Springer Series in Bio-/Neuroinformatics 4

Petia Koprinkova-Hristova
Valeri Mladenov
Nikola K. Kasabov *Editors*

Artificial Neural Networks

Methods and Applications in Bio-/Neuroinformatics

Springer

Springer Series in Bio-/Neuroinformatics

Volume 4

Series editor

Nikola K. Kasabov, Auckland University of Technology, Auckland, New Zealand
e-mail: nkasabov@aut.ac.nz

Aims & Scope

Springer Series in Bio-/Neuroinformatics publishes content on fundamental principles as well as state-of-art theories, methods and applications in the rapidly evolving fields of bioinformatics and neuroinformatics. The series is unique in that it integrates three interdependent disciplines, namely information science, bioinformatics and neuroinformatics. The series covers general informatics methods and techniques, like data mining, database management systems, machine learning, artificial neural networks, evolutionary computation, chaos theory, quantum computation, statistical methods, and image and signal processing, as well as their applications to different areas of research, including big data, functional analysis of the brain, computational systems biology, computational systems neuroscience, health informatics, personalized medicine, rehabilitation medicine, industrial biotechnology, and many more. The series publishes monographs, contributed volumes and conference proceedings, as well as advanced textbooks.

More information about this series at http://www.springer.com/series/10088

Petia Koprinkova-Hristova · Valeri Mladenov
Nikola K. Kasabov
Editors

Artificial Neural Networks

Methods and Applications
in Bio-/Neuroinformatics

Springer

Editors
Petia Koprinkova-Hristova
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Sofia
Bulgaria

Nikola K. Kasabov
Auckland University of Technology
Auckland
New Zealand

Valeri Mladenov
Technical University Sofia
Sofia
Bulgaria

ISSN 2193-9349 ISSN 2193-9357 (electronic)
ISBN 978-3-319-09902-6 ISBN 978-3-319-09903-3 (eBook)
DOI 10.1007/978-3-319-09903-3

Library of Congress Control Number: 2014945765

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The edited book includes chapters that present selected and extended papers from the International Conference on Artificial Neural Networks (ICANN) 2013. Founded in 1991, the ICANN become the Premier annual conference of the European Neural Network Society. Its main goal is to bring together and to facilitate contacts between researchers from information sciences and neurosciences and to provide a high-level international forum for both academic and industrial communities.

The selected and invited to the present book chapters were selected among ICANN papers that received highest scores during the strong peer review assessment (almost 40% rejection rate) and among selected by session chairs best presentations.

The collected in the book chapters are presented in topical sections that cover wide range of contemporary topics varying from neural network theory and models, machine learning and learning algorithms, brain-machine interaction and bio-inspired systems, pattern recognition and classification as well as various applications.

The book chapters include new theoretical developments in recurrent neural networks and reservoir computing, new and improved training algorithms for Deep Boltzmann Machines (DBM), tapped delay feedforward architectures and kernel machines, reinforcement learning and Adaptive Critic Designs (ACD), new bio-inspired models and architectures related to cell assembly mechanisms, visual perception and natural language understanding, new and improved algorithms for pattern recognition with applications to gesture classification, handwritten digit recognition and time series forecasting.

The book will be of interest to all researchers and postgraduate students in the area of computational intelligence, applied mathematics, computer science, engineering, neuroscience, and other related areas.

June 2014 — Editors

Sofia and Auckland

Petia Koprinkova-Hristova
Valeri Mladenov
Nikola K. Kasabov

Contents

Recurrent Neural Networks and Super-Turing Interactive Computation

Jérémie Cabessa and Alessandro E.P. Villa

Abstract. We present a complete overview of the computational power of recurrent neural networks involved in an interactive bio-inspired computational paradigm. More precisely, we recall the results stating that interactive rational- and real-weighted neural networks are Turing-equivalent and super-Turing, respectively. We further prove that interactive evolving neural networks are super-Turing, irrespective of whether their synaptic weights are modeled by rational or real numbers. These results show that the computational powers of neural nets involved in a classical or in an interactive computational framework follow similar patterns of characterization. They suggest that some intrinsic computational capabilities of the brain might lie beyond the scope of Turing-equivalent models of computation, hence surpass the potentialities every current standard artificial models of computation.

1 Introduction

Understanding the computational and dynamical capabilities of biological neural networks represents an issue of central importance. In this context, much interest has been focused on comparing the computational powers of diverse theoretical neural models with those of abstract computing devices. Nowadays, the computational capabilities of neural models is known to be tightly related to the nature of the activation function of the neurons, to the nature of their synaptic connections, to the eventual presence of noise in the model, to the possibility for the networks to evolve over time, and to the computational paradigm performed by the networks.

Jérémie Cabessa
Laboratory of Mathematical Economics (LEMMA), University of Paris 2 – Panthéon-Assas, 4 Rue Blaise Desgoffe, 75006 Paris, France
e-mail: jcabessa@nhrg.org

Alessandro E.P. Villa
Neuroheuristic Research Group, Faculty of Business and Economics, University of Lausanne, CH-1015 Lausanne, Switzerland
e-mail: alessandro.villa@unil.ch

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_1

This comparative approach has been initiated by McCulloch and Pitts who proposed a modeling of the nervous system as a finite interconnection of logical devices [43]. For the first time, neural networks were considered as discrete abstract machines, and the issue of their computational capabilities investigated from the automata-theoretic perspective. In this context, Kleene and Minsky proved that recurrent neural networks with Boolean activation functions are computationally equivalent to classical finite state automata [38, 44]. In Minsky's own words [44]:

> It is evident that each neural network of the kind we have been considering is a finite-state machine. [...] It is interesting and even surprising that there is a converse to this. Every finite-state machine is equivalent to, and can be simulated by, some neural net.

The simulation of finite state machine by various kinds of recurrent neural networks has further been studied for instance in [2, 27, 3, 39, 31, 48].

These foundational works opened up the way to the theoretical approach to neural computation. But the purely discrete and mechanical approach adopted by McCulloch and Pitts quickly appeared too restrictive, far from the biological reality. The neurons that they considered were too similar to classical logic gates, and the structure of the networks was too rigid to allow a biologically plausible implementation of learning.

In 1948, Turing made a step forward by showing the possibility of surpassing the capabilities of finite state machines and reaching Turing universality via neural networks called *B-type unorganized machines* [65]. The networks consisted of a specific interconnection of NAND neurons, and the consideration of infinitely many such cells could simulate the behavior of a Turing machine. The Turing universality of neural networks involving infinitely many binary neurons has further been investigated in many directions, see for instance [49, 28, 20, 19, 52].

Moreover, in the late 50's, von Neumann proposed a particularly relevant approach to the issue of information processing in the brain from the hybrid perspective of *digital* and *analog* computation [47]. He considered that the non-linear character of the operations of the brain emerges from a combination of discrete and continuous mechanisms, and therefore envisioned neural computation as something strictly more powerful than abstract machines, in line with Turing's positions.

Almost at the same time, Rosenblatt proposed the perceptron as a more general computational neural model than the McCulloch-Pitts units [51]. The essential innovation was the introduction of numerical synaptic weights and a special pattern of interconnection. This neural model gave rise to an algorithmic framework of learning achieved by adjusting the synaptic weights of the neuronal connections according to some specific task to be completed. The computational capabilities of the perceptron were further analyzed by Minsky and Papert [45]. These results represent the main achievements concerning the computational power of neural systems until the mid 90's.

In 1995, Siegelmann and Sontag proved the possibility of reaching Turing universality with finite neural networks. By considering rational synaptic weights and by extending the activation functions of the cells from Boolean to linear-sigmoid, the corresponding neural networks have their computational power drastically increased from finite state automata up to Turing machines [60, 32, 46]. Kilian and Siegelmann then generalized the Turing universality of neural networks to a broader class of sigmoidal activation functions [37]. The computational equivalence between so-called *rational recurrent neural networks* and Turing machines has now become standard result in the field.

Moreover, following von Neumann considerations, Siegelmann and Sontag assumed that the variables appearing in the underlying chemical and physical phenomena could be modeled by continuous rather than discrete (rational) numbers, and therefore proposed a precise study of the computational power of recurrent neural networks from the perspective of *analog computation* [56]. They introduced the concept of an *analog recurrent neural network* as a classical linear-sigmoid neural net equipped with real- instead of rational-weighted synaptic connections, and proved that such analog recurrent neural networks are computationally equivalent to Turing machines with advices, hence capable of super-Turing computational power from polynomial time of computation already [59, 58]. This analog information processing model turns out to be capable of capturing non-linear dynamical properties that are most relevant to brain dynamics, such as rich chaotic behaviors [35, 62, 63], as well as dynamical and idealized chaotic systems that cannot be described by the universal Turing machine model [55]. According to these considerations, they formulated the so-called *Thesis of Analog Computation* – an analogous to the Church-Turing thesis, but in the realm of analog computation – stating that no reasonable abstract analog device can be more powerful than first-order analog recurrent neural networks [59, 55]. A proper internal hierarchical classification of analog recurrent neural networks according to the Kolmogorov complexity of their underlying real synaptic weights has further been described in [5].

Besides, central in neural computation is the issue of noise, and a natural question to be addressed concerns the robustness of the computational power of neural networks subjected to various kinds of noise. In this context, it has been shown that the presence of analog noise would generally strongly reduce the computational power of the underlying systems to that of finite state automata, or even below [40, 41, 6]. On the other hand, the incorporation of some discrete source of stochasticity would rather tend to increase or maintain the capabilities of the neural systems [57].

But the neural models considered up to that point were generally oversimplified, lacking many biological features which may be essential to the information processing in the real brain. In particular, the *evolving capabilities* of biological networks had not been taken into consideration in the studies of the computational capabilities of neural models.

In fact, it is nowadays widely admitted that biological mechanisms like synaptic plasticity, cell birth and death, changes in connectivity, etc., are intimately related to the storage and encoding of memory traces in the central nervous system, and provide the basis for most models of learning and memory in neural networks [1, 42]. More precisely, the embryonic nervous system is initially driven by genetic programs that control neural stem cell proliferation, differentiation and migration through the actions of a limited set of trophic factors and guidance cues. After a relatively short period of stable synaptic density, a pruning process begins: synapses are constantly removed, yielding a marked decrease in synaptic density due to apoptosis – genetically programmed cell death – and selective axon pruning [34]. Overproduction of a critical mass of synapses in each cortical area may be essential for their parallel emergence through competitive interactions between extrinsic afferent projections [15]. Background activity and selected patterns of afferent activity are likely to shape deeply the emergent circuit wiring [53]. Synapses can change their strength in response to the activity of both pre- and post-synaptic cells following spike timing dependent plasticity (STDP) rules [50]. Developmental and/or learning processes are likely to potentiate or weaken certain pathways through the network by affecting the number or efficacy of synaptic interactions between the neurons [33]. Despite the plasticity of these phenomena, it is rationale to suppose that whenever the same information is presented in the network, the same pattern of activity is evoked in a circuit of functionally interconnected neurons, referred to as *cell assembly* [29]. In cell assemblies interconnected in this way, some ordered sequences of interspike intervals will recur. Such recurring, ordered, and precise (in the order of few *ms*) interspike interval relationships are referred to as spatiotemporal patterns of discharges or preferred firing sequences. Several evidence exist of spatiotemporal firing patterns in behaving animals, from rats to primates [74, 54], where preferred firing sequences can be associated to specific types of stimuli or behaviors.

In the context of AI, the consideration of such evolving neural architectures in so-called *Evolving Connectionist Systems* (ECoS) has proven to be fruitful, and significantly increased in applications in the recent years [36, 76]. From a theoretical perspective, Turova and Villa showed that neural networks with embedded spike timing-dependent plasticity are able to exhibit a sustained level of activity under special choice of parameters [66, 67]. More recently, Cabessa and Siegelmann, introduced and studied the computational capabilities of a biologically oriented neural model where the synaptic weights, the connectivity pattern, and the number of neurons can evolve rather than stay static [10]. They proved that the so-called *evolving recurrent neural networks* are super-Turing, equivalent to static analog networks, irrespective of whether their synaptic weights are modeled by rational or real numbers. Consequently, the consideration of architectural evolving capabilities in a basic neural model provides an alternative and equivalent way to the incorporation of the power of the continuum towards the achievement of super-Turing computational capabilities.

Besides, in the general context of modern computation, the classical computational approach from Turing [64] has been argued to "no longer fully corresponds to the current notion of computing in modern systems" [73] – especially when it refers to bio-inspired complex information processing systems. In the brain (or in organic life in general), information is rather processed in an interactive way [78, 23], where previous experience must affect the perception of future inputs, and where older memories may themselves change with response to new inputs.

In fact, the cerebral cortex is not a single entity, but an impressive network formed by an order of tens of millions of neurons, most of them excitatory, and by about ten times more glial cells. Ninety percent of the inputs received by a cortical area come from other areas of the cerebral cortex. As a whole, the cerebral cortex can be viewed as a machine talking to itself, and could be seen as one big feedback system subject to the relentless advance of entropy, which subverts the exchange of messages that is essential to continued existence [79]. This concept of interdependent communications systems, also known as systems theory, coupled with Wiener's assertion that a machine that changes its responses based on feedback is a machine that learns, defines the cerebral cortex as a cybernetic machine. Therefore, the focus of investigation is shifted from communication and control to *interaction*. Systems theory has traditionally focused more on the structure of systems and their models, whereas cybernetics has focused more on how systems function, that is to say how they control their actions, how they communicate with other systems or with their own components. However, structure and function of a system cannot be understood in separation, and cybernetics and systems theory should be viewed as two facets of the neuroheuristic approach [4, 61, 75].

Following these considerations, Cabessa and Villa initiated the study of the computational power of recurrent neural networks from the perspective of *interactive computation* [12]. They proved that various kinds of Boolean recurrent neural networks involved in a reactive computational scenario are computationally equivalent to Büchi and Muller automata. From this equivalence, they deduced a transfinite hierarchical classification of Boolean recurrent neural networks based on their attractor properties [12, 8]. Cabessa and Villa also provided a description of the super-Turing computational power of analog recurrent neural networks engaged in a similar reactive computational scenario [13]. Besides, Cabessa and Siegelmann provided a characterization of the Turing and super-Turing capabilities of rational and analog recurrent neural networks involved in a more bio-inspired interactive computational paradigm [11].

Finally, Cabessa proved that neural models combining the two crucial features of *evolvability* and *interactivity* were actually capable of super-Turing computational capabilities, irrespective of whether their synaptic weights are modeled by rational or real numbers [9, 14].

In this chapter, we first review the main results concerning the Turing and super-Turing capabilities of classical and interactive recurrent neural networks, and next provide a detailed proof of these last results stated in [9, 14].

2 Preliminaries

Given some finite alphabet Σ, we let Σ^*, Σ^+, Σ^n, and Σ^ω denote respectively the sets of finite words, non-empty finite words, finite words of length n, and infinite words, all of them over alphabet Σ. We also let $\Sigma^{\leq\omega} = \Sigma^* \cup \Sigma^\omega$ be the set of all possible words (finite or infinite) over Σ. The empty word is denoted λ.

For any $x \in \Sigma^{\leq\omega}$, the *length* of x is denoted by $|x|$ and corresponds to the number of letters contained in x. If x is non-empty, we let $x(i)$ denote the $(i+1)$-th letter of x, for any $0 \leq i < |x|$. The prefix $x(0)\cdots x(i)$ of x is denoted by $x[0{:}i]$, for any $0 \leq i < |x|$. For any $x \in \Sigma^*$ and $y \in \Sigma^{\leq\omega}$, the fact that x is a *prefix* (resp. *strict prefix*) of y is denoted by $x \subseteq y$ (resp. $x \subsetneq y$). If $x \subseteq y$, we let $y - x = y(|x|)\cdots y(|y|-1)$

be the *suffix* of y that is not common to x (if $x = y$, then $y - x = \lambda$). Moreover, the *concatenation* of x and y is denoted by $x \cdot y$.

Given some sequence of finite words $\{x_i : i \in \mathbb{N}\}$ such that $x_i \subseteq x_{i+1}$ for all $i \geq 0$, one defines the *limit* of the x_i's, denoted by $\lim_{i\geq 0} x_i$, as the unique finite or infinite word which is ultimately approached by the sequence of growing prefixes $\{x_i : i \geq 0\}$. Formally, if the sequence $\{x_i : i \in \mathbb{N}\}$ is eventually constant, i.e. there exists an index $i_0 \in \mathbb{N}$ such that $x_j = x_{i_0}$ for all $j \geq i_0$, then $\lim_{i\geq 0} x_i = x_{i_0}$, meaning that $\lim_{i\geq 0} x_i$ corresponds to the smallest finite word containing each word of $\{x_i : i \in \mathbb{N}\}$ as a finite prefix; if the sequence $\{x_i : i \in \mathbb{N}\}$ is not eventually constant, then $\lim_{i\geq 0} x_i$ corresponds to the unique infinite word containing each word of $\{x_i : i \in \mathbb{N}\}$ as a finite prefix.

A function $f : \Sigma^* \to \Sigma^*$ is called *monotone* if the relation $x \subseteq y$ implies $f(x) \subseteq f(y)$, for all $x, y \in \Sigma^*$. It is called *recursive* if it can be computed by some Turing machine. Throughout this paper, any function $\varphi : \Sigma^\omega \to \Sigma^{\leq\omega}$ mapping infinite words to finite or infinite words will be referred to as an *ω-translation*.

Note that any monotone function $f : \{0,1\}^* \to \{0,1\}^*$ induces "in the limit" an ω-translation $f_\omega : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ defined by

$$f_\omega(x) = \lim_{i\geq 0} f(x[0{:}i])$$

for all $x \in \{0,1\}^\omega$. The monotonicity of f ensures that the value $f_\omega(x)$ is well-defined for all $x \in \{0,1\}^\omega$. In words, the value $f_\omega(x)$ corresponds to the finite or infinite word that is ultimately approached by the sequence of growing prefixes $\{f(x[0{:}i]) : i \geq 0\}$.

According to these definitions, an ω-translation $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ will be called *continuous*[1] if there exists a monotone function $f : \{0,1\}^* \to \{0,1\}^*$ such that $f_\omega = \psi$; it will be called *recursive continuous*[2] if there exists a monotone and recursive (i.e. Turing computable) function $f : \{0,1\}^* \to \{0,1\}^*$ such that $f_\omega = \psi$.

3 Interactive Computation

3.1 Historical Background

Interactive computation refers to the computational framework where systems may react or interact with each other as well as with their environment during the computation [78, 23]. This paradigm was theorized in contrast to classical computation [64] which rather proceeds in a function-based transformation of a given input

[1] The choice of this name comes from the fact that continuous functions over the Cantor space $\mathscr{C} = \{0,1\}^\omega$ can be precisely characterized as limits of monotone functions. We extend this definition in the present broader context of functions from $\{0,1\}^\omega$ to $\{0,1\}^{\leq\omega}$ that can also be expressed as limits of monotone functions.

[2] Our notion of a recursive continuous ω-translation $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ is a transposition to the present context of the notion of a limit-continuous function $\varphi : \{0,1\}^\omega \to \{0,1\}^\omega$ defined in [68, Definition 12] and [72, Definition 13].

to a corresponding output (closed-box and amnesic fashion), and has been argued to "no longer fully correspond to the current notions of computing in modern systems" [73]. Interactive computation also provides a particularly appropriate framework for the consideration of natural and bio-inspired complex information processing systems [69, 73, 14].

Wegner first proposed a foundational approach to interactive computation [78]. In his work, he claimed that "interaction is more powerful than algorithms", in the sense that computations performed in an interactive way are capable of handling a wider range of problems than those performed in a classical way, namely by standard algorithms and Turing machines [77, 78].

In this context, Goldin et al. introduced the concept of a *persistent Turing machine* (PTM) as a possible extension of the classical Turing machine model to the framework of interactive computation [21, 22]. A *persistent Turing machine* consists of a multi-tape machines whose inputs and outputs are given as streams of tokens generated in a dynamical and sequential manner, and whose work tape is kept preserved during the transition from one interactive step to the next. In this sense, a PTM computation is sequentially interactive and history dependent. Goldin et al. further provided a transfinite hierarchical classification of PTMs according to their expressive power, and established that PTMs are more expressive (in a precise sense) than amnesic PTMs (an extension of classical Turing machines in their context of interactive computation), and hence also than classical Turing machines [21, 22].

All these consideration led Goldin and Wegner to formulate the so-called *Sequential Interaction Thesis*, a generalization of the Church-Turing Thesis in the realm of interactive computation, claiming that "any sequential interactive computation can be performed by a persistent Turing machine" [22, 24, 25, 26]. They argue that this hypothesis, when combined with their result that PTMs are more expressive than classical TMs, provides a formal proof of Wegner's conjecture that "interaction is more powerful than algorithms" [22, 24, 25, 26], and hence refutes what they call the Strong Church-Turing Thesis – different from the original Church-Turing Thesis –, stating any possible computation can be captured by some Turing machine, or in other words, that "models of computation more expressive than TMs are impossible" [24, 26].

Driven by similar motivations, Van Leeuwen and Wiedermann proposed a slightly different interactive framework where a general component interacts with its environment by translating an incoming input stream of bits into a corresponding output stream of bit in a sequential manner [68, 72]. In their study, they restrict themselves to deterministic components, and provide mathematical characterizations of interactively computable relations, interactively recognizable sets of inputs streams, interactively generated sets of output streams, and interactively computable translations.

In this context, they introduced the concept of an *interactive Turing machine* (I-TM), a relevant translation of the classical Turing machine model in their interactive framework [69]. They further introduced the concept of *interactive Turing machine with advice* (I-TM/A) as a relevant non-uniform computational model in the context of interactive computation [69, 70]. Interactive Turing machines with advice were proven to be strictly more powerful than interactive Turing machines without advice [70, Proposition 5] and [69, Lemma 1], and were shown to be computationally equivalent to several other non-uniform models of interactive computation, like sequences of interactive finite automata, site machines, web Turing machines [69, 70], and more recently to interactive analog neural networks and interactive evolving neural networks [9, 11, 14].

These considerations led van Leeuwen and Wiedermann to formulate an *Interactive Extension of the Church-Turing Thesis* which states that "any (non-uniform interactive) computation can be described in terms of interactive Turing machines with advice" [70].

As opposed to Goldin and Wegner, van Leeuwen and Wiedermann consider that interactivity alone is not sufficient to break the Turing barrier, and rather consists of a different instead of a more powerful paradigm than the classical computational framework [69, 71, 73]. They write [73]:

> "From the viewpoint of computability theory, interactive computing e.g. with I-TMs does not lead to super-Turing computing power. Interactive computing merely extends our view of classically computable functions over finite domains to computable functions (translations) defined over infinite domains. Interactive computers simply compute something different from non-interactive ones because they follow a different scenario."

Here, we follow this point of view and adopt a similar approach to interactive computation as presented in [68, 72].

3.2 The Interactive Paradigm

The general interactive computational paradigm consists of a step by step exchange of information between a system and its environment [68, 72]. In order to capture the unpredictability of next inputs at any time step, the dynamically generated input streams need to be modeled by potentially infinite sequences of symbols (indeed, any interactive computation over a finite input stream can a posteriori be replayed in a non-interactive way producing the same output) [78, 25, 73].

Here, we consider a basic interactive computational scenario similar to that described for instance in [72]. At every time step, the environment first sends a non-empty input bit to the system (full environment activity condition), the system next updates its current state accordingly, and then answers by either producing a corresponding output bit or remaining silent. In other words, the system is not obliged to provide corresponding output bits at every time step, but might instead stay silent for a while (to express the need of some internal computational phase before producing a new output bit), or even staying silent forever (to express the case that it has died). Consequently, after infinitely many time steps, the system will have received an infinite sequence of consecutive input bits and translated it into a corresponding finite or infinite sequence of not necessarily consecutive output bits. In the sequel, we assume that every interactive system is deterministic.

Formally, given some interactive deterministic system $\mathscr{S}$, for any infinite input stream $s \in \{0,1\}^{\omega}$, we define the corresponding output stream $o_s \in \{0,1\}^{\leq \omega}$ of $\mathscr{S}$ as the finite or infinite subsequence of (non-λ) output bits produced by $\mathscr{S}$ after having processed input s. The deterministic nature of $\mathscr{S}$ ensures that the output stream o_s is unique. In this way, any interactive system $\mathscr{S}$ realizes an ω-translation $\varphi_{\mathscr{S}} : \{0,1\}^{\omega} \to \{0,1\}^{\leq \omega}$ defined by $\varphi_{\mathscr{S}}(s) = o_s$, for each $s \in \{0,1\}^{\omega}$.

An ω-translation ψ is then called *interactively deterministically computable*, or simply *interactively computable* iff there exists an interactive deterministic system $\mathscr{S}$ such that $\varphi_{\mathscr{S}} = \psi$. Note that in this definition, we do absolutely not require for the system $\mathscr{S}$ to be driven by a Turing program nor to contain any computable component of whatever kind. We simply require that $\mathscr{S}$ is deterministic and performs ω-translations in conformity with our interactive paradigm, namely in a sequential interactive manner, as precisely described above.

3.3 Interactive Computable Functions

The specific nature of the interactive computational scenario imposes strong conditions on the ω-translations that can be performed by interactive deterministic systems in general. In fact, it can be proven that any interactively computable ω-translation is necessarily continuous. This result will be used in the sequel.

Proposition 1. *Let ψ be some ω-translation. If ψ is interactively computable, then it is continuous.*

Proof. Let ψ be an interactively computable ω-translation. Then by definition, there exists a deterministic interactive system $\mathscr{S}$ such that $\varphi_{\mathscr{S}} = \psi$. Now, consider the function $f : \{0,1\}^* \to \{0,1\}^*$ which maps every finite word u to the unique corresponding finite word produced by $\mathscr{S}$ after exactly $|u|$ steps of computation over input stream u provided bit by bit. Note that the deterministic nature of $\mathscr{S}$ ensures that the finite word $f(u)$ is indeed unique, and thus that the function f is well-defined.

We show that f is monotone. Suppose that $u \subseteq v$. It follow that $v = u \cdot (v - u)$. Hence, according to our interactive scenario, the output strings produced by $\mathscr{S}$ after $|v|$ time steps of computation over input stream v, namely $f(v)$, simply consists of the output strings produced after $|u|$ time steps of computation over input u, namely $f(u)$, followed by the output strings produced after $|v - u|$ time steps of computation over input $v - u$. Consequently, $f(u) \subseteq f(v)$, and therefore f is monotone.

We now prove that the ω-translation $\varphi_{\mathscr{S}}$ performed by the interactive system $\mathscr{S}$ corresponds to the the "limit" (in the sense of Section 2) of the monotone function f, i.e., that $\varphi_{\mathscr{S}} = f_{\omega}$. Towards this purpose, given some infinite input stream $s \in \{0,1\}^{\omega}$, we consider in turn the two possible cases where $\varphi_{\mathscr{S}}(s)$ is either an infinite or a finite word.

First, suppose that $\varphi_{\mathscr{S}}(s) \in \{0,1\}^{\omega}$. By definition, the word $\varphi_{\mathscr{S}}(s)$ corresponds to the output stream produced by $\mathscr{S}$ after having processed the whole infinite input s, and, for any $i \geq 0$, the word $f(s[0{:}i])$ corresponds to the output stream produced by $\mathscr{S}$ after $i+1$ time steps of computation over the input $s[0{:}i]$. According to our interactive scenario, $f(s[0{:}i])$ is a prefix of $\varphi_{\mathscr{S}}(s)$, for all $i \geq 0$ (indeed, once again, what has been produced by $\mathscr{S}$ on s after infinitely many time steps, namely $\varphi_{\mathscr{S}}(s)$, consists of what has been produced by $\mathscr{S}$ on $s[0{:}i]$ after $i+1$ time steps, namely $f(s[0{:}i])$, followed by what has been produced by $\mathscr{S}$ on $s - s[0{:}i]$ after infinitely many time steps). Moreover, since $\varphi_{\mathscr{S}}(s) \in \{0,1\}^\omega$, it means that the sequence of partial output strings produced by $\mathscr{S}$ on input s after i time steps of computation is not eventually constant, i.e., $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$. Hence, the two properties $f(s[0{:}i]) \subseteq \varphi_{\mathscr{S}}(s) \in \{0,1\}^\omega$ for all $i \geq 0$ and $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$ ensure that $\varphi_{\mathscr{S}}(s)$ is the unique infinite word containing each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say by definition that $\varphi_{\mathscr{S}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_\omega(s)$.

Secondly, suppose that $\varphi_{\mathscr{S}}(s) \in \{0,1\}^*$. By the very same argument as in the previous case, $f(s[0{:}i])$ is a prefix of $\varphi_{\mathscr{S}}(s)$, for all $i \geq 0$. Moreover, since $\varphi_{\mathscr{S}}(s) \in \{0,1\}^*$, the sequence of partial output strings produced by $\mathscr{S}$ on input s after i time steps of computation must become stationary from some time step j onwards, i.e. $\lim_{i\to\infty} |f(s[0{:}i])| < \infty$. Hence, the entire finite output stream $\varphi_{\mathscr{S}}(s)$ must necessarily have been produced after a finite amount of time, and thus $\varphi_{\mathscr{S}}(s) \in \{f(s[0{:}i]) : i \geq 0\}$. Consequently, the two properties $f(s[0{:}i]) \subseteq \varphi_{\mathscr{S}}(s) \in \{0,1\}^*$ for all $i \geq 0$ and $\varphi_{\mathscr{S}}(s) \in \{f(s[0{:}i]) : i \geq 0\}$ ensure that $\varphi_{\mathscr{S}}(s)$ is the smallest finite word that contains each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say by definition that $\varphi_{\mathscr{S}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_\omega(s)$. Consequently, $\varphi_{\mathscr{S}}(s) = f_\omega(s)$ for any $s \in \{0,1\}^\omega$, meaning that $\varphi_{\mathscr{S}} = f_\omega$.

We proved that f is a monotone function satisfying $\varphi_{\mathscr{S}} = f_\omega$. This means by definition that $\varphi_{\mathscr{S}}$ is continuous. Since $\varphi_{\mathscr{S}} = \psi$, it follows that ψ is also continuous. $\square$

3.4 Interactive Turing Machines

An *interactive Turing machine* consists of an interactive abstract device driven by a standard Turing machine program. It receives an infinite stream of bits as input and produces a corresponding stream of bits as output, step by step. The input and output bits are processed via corresponding input and output ports rather than tapes. Consequently, at every time step, the machine can no more operate on the output bits that have already been processed.[3] Furthermore, according to our interactive scenario, it is assumed that, at every time step, the environment sends a non-silent input bit to the machine, and the machine either answers by producing some corresponding output bit, or rather chooses to remain silent.

Formally, a deterministic *interactive Turing machine* (I-TM) $\mathscr{M}$ is defined as a tuple $\mathscr{M} = (Q, \Gamma, \delta, q_0)$, where Q is a finite set of states, $\Gamma = \{0,1,\lambda,\sharp\}$ is the alphabet of the machine, where $\sharp$ stands for the blank tape symbol, $q_0 \in Q$ is the initial state, and

$$\delta : Q \times \Gamma \times \{0,1\} \to Q \times \Gamma \times \{\leftarrow, \rightarrow, -\} \times \{0,1,\lambda\}$$

[3] In fact, allowing the machine to erase its previous output bits would lead to the consideration of much more complicated ω-translations.

is the transition function of the machine. The relation $\delta(q,x,b) = (q',x',d,b')$ means that if the machine $\mathcal{M}$ is in state q, the cursor of the tape is scanning the letter $x \in \{0,1,\sharp\}$, and the bit $b \in \{0,1\}$ is currently received at its input port, then $\mathcal{M}$ will go in next state q', it will make the cursor overwrite symbol x by $x' \in \{0,1,\sharp\}$ and then move to direction d, and it will finally output symbol $b' \in \{0,1,\lambda\}$ at its output port, where λ represents the fact the machine is not outputting any bit at that time step. An interactive Turing machine is illustrated in Figure 1.

According to this definition, any I-TM $\mathcal{M}$ induces an ω-translation $\varphi_{\mathcal{M}} : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$ mapping every infinite input stream s to the corresponding finite or infinite output stream o_s produced by $\mathcal{M}$. An ω-translation $\psi : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$ is said to be *realizable* by some interactive Turing machine iff there exists an I-TM $\mathcal{M}$ such that $\varphi_{\mathcal{M}} = \psi$.

Van Leeuwen and Wiedermann also introduced the concept of *interactive Turing machine with advice* as a relevant non-uniform computational model in the context of interactive computation [69, 70].

Formally, a deterministic *interactive Turing machine with advice* (I-TM/A) $\mathcal{M}$ consists of an interactive Turing machine provided with an advice mechanism, which comes in the form of an advice function $\alpha : \mathbb{N} \to \{0,1\}^*$. In addition, the machine $\mathcal{M}$ uses two auxiliary special tapes, an advice input tape and an advice output tape, as well as a designated advice state. During its computation, $\mathcal{M}$ has the possibility to write the binary representation of an integer m on its advice input tape, one bit at a time. Yet at time step n, the number m is not allowed to exceed n. During the computation, if the machine happens to enter its designated advice state at some time step, then the string $\alpha(m)$ is written on the advice output tape in one time step, replacing the previous content of the tape. The machine has the possibility to repeat this process as many time as needed during its infinite computation. An interactive Turing machine with a advice is illustrated in Figure 2.

Once again, according to our interactive scenario, any I-TM/A $\mathcal{M}$ induces an ω-translation $\varphi_{\mathcal{M}} : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$ which maps every infinite input stream s to the corresponding finite or infinite output stream o_s produced by $\mathcal{M}$. Finally, an ω-translation $\psi : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$ is said to be *realizable* by some interactive Turing machine with advice iff there exists an I-TM/A $\mathcal{M}$ such that $\varphi_{\mathcal{M}} = \psi$.

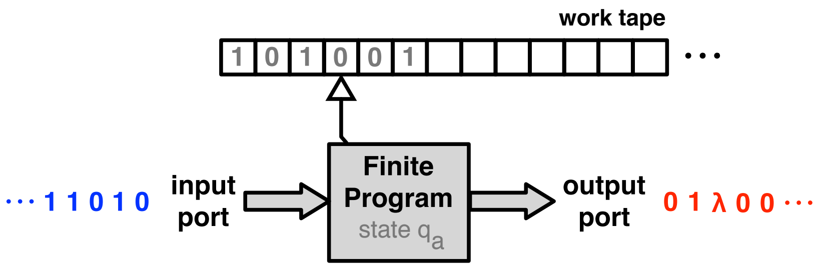


Fig. 1 An interactive Turing machine

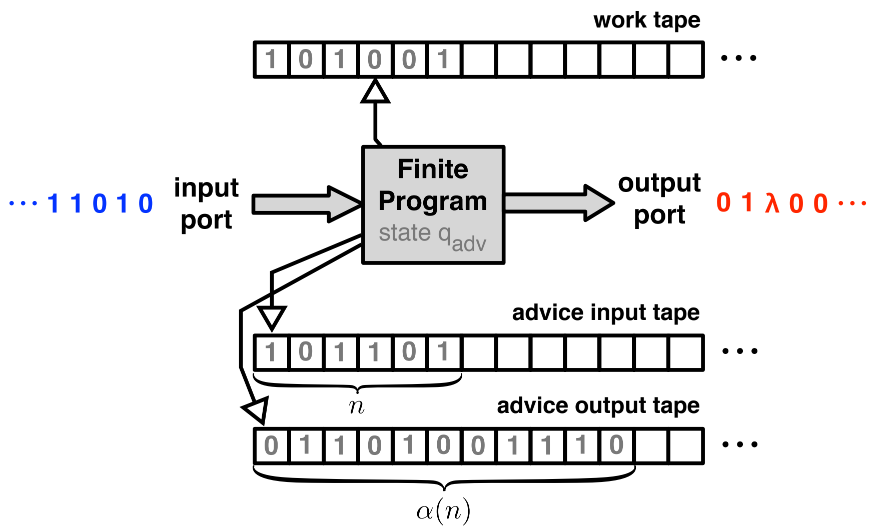


Fig. 2 An interactive Turing machine with advice

For sake of completeness, we provide a proof that I-TM/A are strictly more powerful than I-TM. Accordingly, we say that I-TM/A are *super-Turing*. The result has already been mentioned in [70, Proposition 5] and [69, Lemma 1]

Proposition 2. *I-TM/As are strictly more powerful than I-TMs.*

Proof. We prove that there exists an ω-translation ψ which is realizable by some I-TM/A, yet by no I-TM. Consider a non-Turing computable function $\alpha : \mathbb{N} \to \{0,1\}^*$. Note that such a function obviously exists since there are $2^{\aleph_0}$ (i.e. uncountably many) distinct functions of that form whereas there are only $\aleph_0$ (i.e. countably many) possible Turing machines. Consider the ω-translation $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ which maps every infinite input stream s, necessarily writable of the form $s = 0^* b_0 0^+ b_1 0^+ b_2 0^+ b_3 \cdots$ where b_i's denote the blocks of 1's occurring between the 0's, to the corresponding finite or infinite word given by $\psi(s) = \alpha(|b_0|)\alpha(|b_1|)\alpha(|b_2|)\alpha(|b_3|)\cdots$ (if s has suffix 0^ω, then $\psi(s)$ is finite).

The ω-translation ψ is clearly realizable by some I-TM/A $\mathscr{M}$ with advice function α. Indeed, on every input stream $s \in \{0,1\}^\omega$, the machine $\mathscr{M}$ stores the successive blocks $b_0, b_1, b_2, \ldots$ of 1's occurring in s, and, for every such block b_i, first computes the length $|b_i|$, writes it in binary on its advice tape, then calls the advice value $\alpha(|b_i|)$ (or waits enough time steps in order to have the right to call it), and finally outputs the value $\alpha(|b_i|)$, before reiterating the procedure with respect to the next block b_{i+1}. In this way, $\mathscr{M}$ realizes ψ.

On the other hand, the ω-translation ψ is not realizable by any I-TM. Indeed, towards a contradiction, suppose it is realizable by some I-TM $\mathscr{M}$. Then, consider the classical Turing machine $\mathscr{M}'$ which, on every finite input r of the form $r = 1^k$, proceeds exactly like $\mathcal{M}$ would have on any infinite input beginning by r, thus outputs $\alpha(k)$, and diverges on every other finite input. The existence of this classical TM $\mathcal{M}'$ shows that the function α is Turing computable, a contradiction. □

Moreover, a precise characterization of the computational powers of I-TMs and I-TM/As can be given. In fact, the I-TMs and I-TM/As realize precisely the classes of recursive continuous and continuous ω-translations, respectively. The following results are proven in [11]. Since these proofs can be easily deduced from those of previous Proposition 1 and forthcoming Lemma 1, we can include them hereafter.

Proposition 3. *Let ψ be some ω-translation.*

a) ψ is realizable by some I-TM iff ψ is recursive continuous.
b) ψ is realizable by some I-TM/A iff ψ is continuous.

Proof. The proofs of points (a) and (b) rely on previous Proposition 1 and forthcoming Lemma 1.

Point (a). Let ψ be some ω-translation realized by some I-TM $\mathcal{M}$. This means that $\psi = \varphi_{\mathcal{M}}$. Now, consider the function $f : \{0,1\}^* \to \{0,1\}^*$ which maps every finite word u to the unique corresponding finite word produced by $\mathcal{M}$ after exactly $|u|$ steps of computation over input stream u provided bit by bit. Since $\mathcal{M}$ is driven by a classical TM program, f is recursive. Moreover, by the exact same argument as in the proof of Proposition 1, f is monotone, and $f_\omega = \varphi_{\mathcal{M}} = \psi$. Consequently, ψ is recursive continuous.

Conversely, let ψ be a recursive continuous ω-translation. Then there exists some recursive monotone function $f : \{0,1\}^* \to \{0,1\}^*$ such that $f_\omega = \psi$. Now consider the forthcoming infinite Procedure 1 (proof of Lemma 1) where the three instructions "decode $s[0{:}i]$ from x", "access to the value q_{i+1}", and "decode $f(s[0{:}i])$ from q_{i+1}" are replaced by the following one: "compute $f(s[0{:}i])$". Since f is recursive, this slightly modified version of Procedure 1 can clearly be performed by an I-TM $\mathcal{M}$. The machine $\mathcal{M}$ outputs the current word $v - u$ bit by bit every time it reaches up the instruction "output $v - u$ bit by bit", and otherwise, keeps outputting λ symbols while simulating any other internal computational steps. By the exact same argument as the one presented in the proof of Lemma 1, one has that $\varphi_{\mathcal{M}} = f_\omega = \psi$, meaning that ψ is realized by $\mathcal{M}$.

Point (b). Let ψ be some ω-translation realized by some I-TM/A $\mathcal{M}$. By definition, ψ is interactively computable. By Proposition 1, ψ is continuous.

Conversely, let ψ be a continuous ω-translation. Then there exists some monotone function $f : \{0,1\}^* \to \{0,1\}^*$ such that $f_\omega = \psi$. First of all, we consider the function $\alpha : \mathbb{N} \to \{0,1\}^*$ which maps every integer n to the finite binary word w_n described in the beginning of the proof of forthcoming Lemma 1. Now consider the forthcoming infinite Procedure 1 (proof of Lemma 1) where the three instructions "decode $s[0{:}i]$ from x", "access to the value q_{i+1}", and "decode $f(s[0{:}i])$ from q_{i+1}" are replaced by the two following ones: "query $\alpha(i+1) = w_{i+1}$" and "extract the subword $f(s[0{:}i])$ from w_{i+1}". This slightly modified version of Procedure 1 can clearly be performed by an I-TM/A $\mathcal{M}$ with advice function α. Every time $\mathcal{M}$ encounters the instruction "query $\alpha(i+1) = w_{i+1}$", it makes an extra-recursive call to its advice value $\alpha(i+1)$; otherwise, $\mathscr{M}$ simulates every other recursive step by means of its classical Turing program. Moreover, $\mathscr{M}$ outputs the current word $v-u$ bit by bit every time it reaches up the instruction "output $v-u$ bit by bit", and otherwise keeps outputting λ symbols while simulating any other internal computational steps. By the exact same argument as the one presented in the proof of forthcoming Lemma 1, one has that $\varphi_{\mathscr{M}} = f_\omega = \psi$, meaning that ψ is realized by $\mathscr{M}$. □

4 Interactive Recurrent Neural Networks

4.1 Recurrent Neural Networks

In this work, we consider a classical model of a first-order recurrent neural network, as presented for instance in [59, 60, 55, 56].

A *(first-order) recurrent neural network* (RNN) consists of a synchronous network of neurons (or processors) related together in a general architecture. The network contains a finite number of neurons $(x_i)_{i=1}^N$, M parallel input neurons $(u_i)_{i=1}^M$, and P designated output neurons among the N. The input and output neurons are used to transmit the information from the environment to the network or from the network to the environment, respectively. At each time step, the activation value of every neuron is updated by applying a linear-sigmoid function to some weighted affine combination of values of other neurons or inputs at previous time step.

Formally, given the activation values of the internal and input neurons $(x_j)_{j=1}^N$ and $(u_j)_{j=1}^M$ at time t, the activation value of each neuron x_i at time $t+1$ is then updated by the following equation

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}\cdot x_j(t) + \sum_{j=1}^{M} b_{ij}\cdot u_j(t) + c_i\right),\quad i=1,\ldots,N \qquad (1)$$

where a_{ij}, b_{ij}, and c_i are numbers describing the weighted synaptic connections and weighted bias of the network, and σ is the classical saturated-linear activation function defined by

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \le x \le 1, \\ 1 & \text{if } x > 1. \end{cases}$$

Besides, Cabessa and Siegelmann introduced the model of an *evolving recurrent neural network* (Ev-RNN) as a RNN whose synaptic weights have the possibility to evolve over time rather than remaining static [10]. Formally, an *evolving recurrent neural network* (Ev-RNN) consists of an RNN whose dynamics is given by the following equation:

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t)\cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t)\cdot u_j(t) + c_i(t)\right),\quad i=1,\ldots,N \qquad (2)$$

where $a_{ij}(t)$, $b_{ij}(t)$, and $c_i(t)$ are bounded and time-dependent synaptic weights, and σ is the classical saturated-linear activation function.

The time dependence of the synaptic weights determines the evolving capabilities of the network. The boundedness condition expresses the fact that the synaptic strengths are confined into a certain range of values imposed by the biological constitution of the neurons. It formally states that there exist an upper and a lower bound s and s' such that $a_{ij}(t), b_{ij}(t), c_i(t) \in [s, s']$ for every $t \geq 0$.

Note that this evolving neural model can describe important dynamics other than the sole synaptic plasticity. For instance, creation or deterioration of synapses can be modeled by switching the corresponding synaptic weights on or off, respectively, and cell birth and death are modeled by simultaneously switching on or off the adjacent synaptic weights of a given cell, respectively.

According to these definitions, four models of RNNs can be considered according to whether their underlying synaptic weights are either rational or real numbers of either static or evolving nature. More precisely, a network will be called *rational* if all its weights are rational numbers, and *real* if all its weights are real numbers. It will also be called *static* if all its weights remain static over time, and *evolving* if its weights are time dependent. According to these definitions, the corresponding notions of *static rational* (St-RNN[$\mathbb{Q}$]), *static real* (St-RNN[$\mathbb{R}$]), *evolving rational* (Ev-RNN[$\mathbb{Q}$]), and *evolving real* (Ev-RNN[$\mathbb{R}$]) recurrent neural networks will be employed.

Observe that since rational numbers are included in real numbers, any rational network is also a real network by definition. Moreover, since static weights are particular cases of evolving weights where the evolving patterns remain constant over time, it follows that any static network is also an evolving network. The converses of these two affirmations are obviously not true. Hence, the class of St-RNN[$\mathbb{Q}$]s corresponds precisely to the intersection of the classes of St-RNN[$\mathbb{R}$]s and Ev-RNN[$\mathbb{Q}$]s, and all the latter three classes are included in the class of Ev-RNN[$\mathbb{R}$]s, as illustrated in Figure 3.

4.2 Recurrent Neural Networks and Interactive Computation

In order to stay consistent with the interactive scenario presented in Section 3.2, we define a model of an *interactive recurrent neural network* (I-RNN) which adheres to a rigid encoding of the way inputs and outputs are interactively processed between the environment and the network. This model has already been considered in [11, 9, 14].

An *interactive recurrent neural network* (I-RNN) consists of RNN provided with a single input cell u as well as two binary output cells[4], a data cell y_d and validation cell y_v. The role of the input cell u is to transmit to the network the infinite input stream of bits sent by the environment. At each time step $t \geq 0$, the cell u admits an activation value $u(t)$ belonging to $\{0,1\}$ (the full environment activity condition

[4] The binary requirement of the output cells y_d and y_v means that the network is designed such that for every input and every time step t, one has $y_d(t) \in \{0,1\}$ and $y_v(t) \in \{0,1\}$.

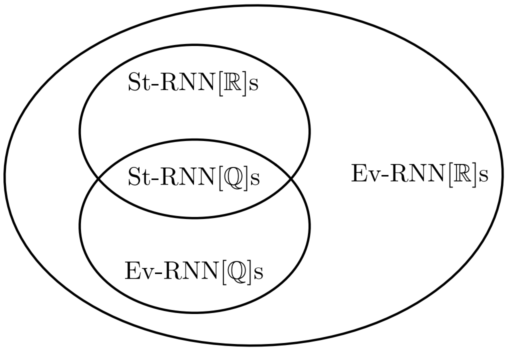


Fig. 3 Inclusion relations between the four classes of St-RNN[ℚ]s, St-RNN[ℝ]s, Ev-RNN[ℚ]s, and Ev-RNN[ℝ]s

forces that $u(t)$ never equals λ). Moreover, the role of the data cell y_d is to carry the output stream of the network to the environment, while the role of the validation cell y_v is to describe when the data cell is active and when it is silent. Accordingly, the output stream transmitted by the network to the environment will be defined as the (finite or infinite) subsequence of successive data bits that occur simultaneously with positive validation bits.

Formally, any I-RNN $\mathcal{N}$ will be supposed to have its initial activation values set to zero, i.e. $x_i(0) = 0$, for $i = 1, \dots, N$. Then any infinite input stream

$$s = s(0)s(1)s(2)\cdots \in \{0,1\}^\omega$$

transmitted to input cell u induces via equations (1) or (2) a corresponding pair of infinite streams transmitted by cells y_d and y_v

$$(y_d(0)y_d(1)y_d(2)\cdots, y_v(0)y_v(1)y_v(2)\cdots) \in \{0,1\}^\omega \times \{0,1\}^\omega.$$

The output stream of $\mathcal{N}$ associated to input s is then given by the finite or infinite subsequence o_s of successive data bits that occur simultaneously with positive validation bits, namely

$$o_s = \langle y_d(i) : i \in \mathbb{N} \text{ and } y_v(i) = 1 \rangle \in \{0,1\}^{\leq\omega}.$$

Hence, any I-RNN $\mathcal{N}$ naturally induces an ω-translation $\varphi_{\mathcal{N}} : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ defined by $\varphi_{\mathcal{N}}(s) = o_s$, for each $s \in \{0,1\}^\omega$. Finally, an ω-translation $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ is said to be *realizable* by some I-RNN iff there exists some I-RNN $\mathcal{N}$ such that $\varphi_{\mathcal{N}} = \psi$.

In this work, four models of I-RNNs will be considered according to whether their underlying synaptic weights are either rational or real numbers of either static or evolving nature. More precisely, the four notions of an *interactive static rational*

(I-St-RNN[$\mathbb{Q}$]), *interactive static real* (I-St-RNN[$\mathbb{R}$]), *interactive evolving rational* (I-Ev-RNN[$\mathbb{Q}$]), *interactive evolving real* (I-Ev-RNN[$\mathbb{R}$]) recurrent neural networks will be employed. An I-RNN is illustrated in Figure 4.

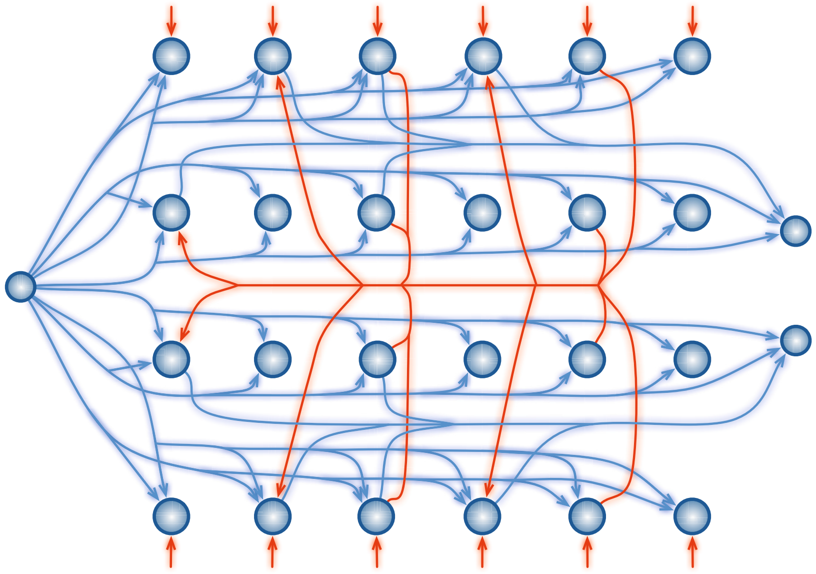


Fig. 4 An interactive recurrent neural network (I-RNN). The single neuron at the very left side represents the input cell. The two little neurons at the very right side represent the output data and validation cells. The forward and recurrent synaptic connections are represented in blue and red, respectively. The background activity connections are represented in red also. The shaded style of the synaptic connections illustrate the fact that the synaptic weights might evolve over time, in the case of an evolving interactive RNN.

5 Computational Power of Classical Neural Networks

For the sake of clarity, we recall the main results concerning the computational powers of recurrent neural networks in the case of classical (i.e. non-interactive) computation. In this context, static rational RNNs were proven to be Turing equivalent [60], whereas static real (or analog) RNNs were proven to be super-Turing [59]. Furthermore, evolving RNNs were shown to be also super-Turing, irrespective of whether their synaptic weights are modeled by rational or real numbers [10]. The three following theorems state these results in details. We now focus on particular each result.

First of all, rational-weighted RNNs were proven to be computationally equivalent to Turing machines (TMs) [60]. Indeed, on the one hand, any function determined by Equation (1) and involving rational weights is necessarily recursive, and thus can be computed by some TM. On the other hand, it was shown that any Turing machine can be simulated in linear time by some rational RNN. The result can be expressed as follows.

Theorem 1. *St-RNN[$\mathbb{Q}$]s are Turing equivalent. More precisely, a language $L \subseteq \{0,1\}^+$ is decidable by some St-RNN[$\mathbb{Q}$] if and only if L is decidable by some TM, i.e., if and only if L is recursive.*

Secondly, real-weighted (or analog) RNNs were shown to be super-Turing, namely strictly more powerful than Turing machines, and hence also than rational RNNs. More precisely, real RNNs are capable of deciding all possible languages in exponential time of computation. When restricted to polynomial time of computation, real RNNs are computationally equivalent to Turing machines with polynomial advice[5] (TM/poly(A)), and hence decide the complexity class of languages **P/poly** [59]. Since **P/poly** strictly includes the class **P** and contains non-recursive languages, it follows that the real networks are capable of super-Turing computational power already from polynomial time of computation. Consequently, the translation from the rational- to the real-weighted context does add to the computational power of the RNNs. These results are summarized in the following theorem.

Theorem 2. *St-RNN[$\mathbb{R}$]s are super-Turing. More precisely, any language $L \subseteq \{0,1\}^+$ can be decided in exponential time by some St-RNN[$\mathbb{R}$]. Moreover, a language $L \subseteq \{0,1\}^+$ is decidable in polynomial time by some St-RNN[$\mathbb{R}$] if and only if L is decidable in polynomial time by some TM/poly(A), i.e., if and only if $L \in$* **P/poly**.

Thirdly, evolving RNNs were shown to be also super-Turing, irrespective of whether their synaptic weights are modeled by rational or real numbers [10]. Hence, the translation from static rational to the evolving rational context does also bring up additional computational power to the networks. However, as opposed to the static context, the translation from the evolving rational to the evolving real does not increase further the capabilities of the networks.

Theorem 3. *Ev-RNN[$\mathbb{Q}$]s and Ev-RNN[$\mathbb{R}$]s are super-Turing equivalent. More precisely, any language $L \subseteq \{0,1\}^+$ can be decided in exponential time by some Ev-RNN[$\mathbb{Q}$] or by some Ev-RNN[$\mathbb{R}$]. Moreover, a language $L \subseteq \{0,1\}^+$ is decidable in polynomial time by some Ev-RNN[$\mathbb{Q}$] or by some Ev-RNN[$\mathbb{R}$] if and only if L is decidable in polynomial time by some TM/poly(A), i.e., if and only if $L \in$* **P/poly**.

The computational capabilities of classical RNNs stated in previous theorems 1, 2, and 3 are summarized in Table 1 below.

[5] We recall that a *Turing machine with advice* (TM/A) consists of a classical Turing machine provided with an additional advice function $\alpha : \mathbb{N} \to \{0,1\}^+$ as well as an additional advice tape, and such that, on every input u of length n, the machine first copies the advice word $\alpha(n)$ on its advice tape and then continues its computation according to its finite Turing program. A *Turing machine with polynomial-bounded advice* (TM/poly(A)) consists of a TM/A whose advice length is bounded by some polynomial. The complexity classes **P** and **P/poly** represents the set of all languages decidable in polynomial time by some TM and some TM/poly(A), respectively.

Table 1 Computational power of static and evolving RNNs according to the nature of their synaptic weights

	Static RNNs	Evolving RNNs
$\mathbb{Q}$	Turing	super-Turing
$\mathbb{R}$	super-Turing	super-Turing

6 Computational Power of Interactive Static Neural Networks

Cabessa and Villa initiated the study of the computational power of RNNs involved in a reactive computational context [13]. They proved that deterministic and non-deterministic real RNNs working on infinite input streams are strictly more expressive than Turing machines equipped with Büchi or Muller conditions, respectively. More recently, Cabessa and Siegelmann studied the computational power of RNNs involved in an interactive computational framework similar the one presented here.

First, they proved that interactive rational RNNs are Turing-equivalent, and hence realize the class of recursive continuous ω-translations [11]. The results is formally expressed as follows.

Theorem 4. *I-St-RNN[$\mathbb{Q}$] are Turing-equivalent. More precisely, for any ω-translation $\psi : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$, the following conditions are equivalent:*

a) ψ is realizable by some I-St-RNN[$\mathbb{Q}$];
b) ψ is realizable by some I-TM;
c) ψ is recursive continuous.

Second, they showed that interactive real RNNs are super-Turing. They are computationally equivalent to I-TM/A, and realize the class of continuous ω-translations [11]. Hence, similarly to the classical case, the translation from the rational- to the real-weighted context does bring additional computational power to the neural networks.

Theorem 5. *I-St-RNN[$\mathbb{R}$] are super-Turing. More precisely, for any ω-translation $\psi : \{0,1\}^{\omega} \to \{0,1\}^{\leq\omega}$, the following conditions are equivalent:*

a) ψ is realizable by some I-St-RNN[$\mathbb{R}$];
b) ψ is realizable by some I-TM/A;
c) ψ is continuous.

Theorems 4 and 5 provide a generalization of theorems 1 and 2 to the interactive context. Note that the equivalences between point b and c of theorems 4 and 5 are given by Proposition 3. According to these results, for the classical as for the interactive computational framework, the translation from the static rational- to the static real-weighted context, or in other words, the incorporation of some power of continuum in the model, does bring additional capabilities to the neural networks.

7 Computational Power of Interactive Evolving Neural Networks

In this section, we prove that interactive evolving RNNs are super-Turing, irrespective of whether their synaptic weights are modeled by rational or real numbers. More precisely, both models of interactive rational and interactive real RNNs are computationally equivalent to interactive Turing machines with advice, and realize the class of continuous ω-translations. Consequently, in both classical and interactive frameworks, the translation from static rational to the evolving rational context does bring additional computational power to the networks. Once again, the translation from the evolving rational to the evolving real does not increase further the capabilities of the networks. These results provide a generalization of Theorem 3 to the context of interactive computation. They show that the power of evolution provides the possibility to break the Turing barrier of computation. A concise form of these results has already appeared in [9, 14]. The results are proven here in details.

Theorem 6. *I-Ev-RNN[$\mathbb{Q}$]s and I-Ev-RNN[$\mathbb{R}$]s are super-Turing. More precisely, for any ω-translation $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$, the following conditions are equivalent:*

a) ψ is realizable by some I-Ev-RNN[$\mathbb{Q}$];
b) ψ is realizable by some I-Ev-RNN[$\mathbb{R}$];
c) ψ is realizable by some I-TM/A;
d) ψ is continuous.

Proof. The implication "a $\to$ b" holds by definition. The three implications "a $\to$ d", "b $\to$ d", and "c $\to$ d" are given by Proposition 1. The equivalence "d $\leftrightarrow$ c" is provided by Proposition 3(a). The implication "d $\to$ a" is given by forthcoming Lemma 1. By combining all these implications and equivalences, the equivalences between points a, b, c and d are obtained. □

Lemma 1. *Let $\psi : \{0,1\}^\omega \to \{0,1\}^{\leq\omega}$ be some continuous ω-translation. Then ψ is realizable by some I-Ev-RNN[$\mathbb{Q}$].*

Proof. Let ψ be a continuous function. Then there exists some monotone function $f : \{0,1\}^* \to \{0,1\}^*$ such such that $f_\omega = \psi$. We begin by encoding all possible values of f into successive distinct rational numbers. Towards this purpose, for any $n > 0$, we let $w_{n,1}, \dots, w_{n,2^n}$ be the lexicographical enumeration of all binary words of length n, and we let $w_n \in \{0,1,2\}^*$ be the finite word given by $w_n = 2 \cdot f(w_{n,1}) \cdot 2 \cdot f(w_{n,2}) \cdot 2 \cdots 2 \cdot f(w_{n,2^n}) \cdot 2$. Then, we consider the following rational encoding of the word w_n

$$q_n = \sum_{i=1}^{|w_n|} \frac{2 \cdot w_n(i) + 1}{6^i}.$$

Note that $q_n \in]0,1[$ for all $n > 0$. Also, the encoding procedure ensures that $q_n \neq q_{n+1}$, since $w_n \neq w_{n+1}$, for all $n > 0$. Moreover, it can be shown that the finite word w_n can be decoded from the value q_n by some Turing machine, or equivalently, by some rational recurrent neural network [59, 60]. In this way, for any $n > 0$, the number q_n provides a rational encoding of the images by f of all words of length n.

Now, we consider the infinite Procedure 1 described below. This procedure receives as input an infinite stream $s = s(0)s(1)s(2)\cdots \in \{0,1\}^\omega$ provided bit by bit, and eventually produces as output a corresponding finite or infinite stream of bits. The procedure consists of two infinite subroutines running in parallel. The first subroutine stores each input bit $s(t)$ occurring at every time step t. The second subroutine performs an infinite loop. More precisely, at stage $i+1$, the procedure considers the value $f(s[0{:}i{+}1])$. By monotonicity of f, the word $f(s[0{:}i{+}1])$ extends $f(s[0{:}i])$. If this extension is strict, the procedure output the difference word $f(s[0{:}i{+}1]) - f(s[0{:}i])$ bit by bit. Otherwise, the procedure simply outputs the empty word λ. Note that the only non-recursive instruction of Procedure 1 is "access to the value q_{i+1}".

Procedure 1.

```
input: infinite input stream $s = s(0)s(1)s(2)\cdots \in \{0,1\}^\omega$ provided bit by bit
initialization: $i \leftarrow 0, x \leftarrow \lambda, u \leftarrow \lambda, v \leftarrow \lambda$

SUBROUTINE 1:
for all $t \geq 0$ do
    $x \leftarrow x \cdot s(t)$        // concatenation of the current bit $s(t)$ to $x$
end for

SUBROUTINE 2:
loop
    decode $s[0{:}i]$ from $x$
    access to the value $q_{i+1}$        // non-recursive instruction
    decode $f(s[0{:}i])$ from $q_{i+1}$
    $v \leftarrow f(s[0{:}i])$
    if $u \subsetneq v$ then
        output $v - u$ bit by bit
    else
        output $\lambda$
    end if
    $i \leftarrow i + 1$
    $u \leftarrow v$
end loop
```

We now show that there indeed exists some I-Ev-RNN[$\mathbb{Q}$] $\mathcal{N}$ which performs Procedure 1. The network $\mathcal{N}$ consists of one evolving and one static rational sub-network connected together. The evolving sub-network will be in charge of the execution of the only non-recursive instruction "access to the value q_{i+1}", and the static sub-network will be in charge of the execution of all other recursive instructions of Procedure 1.

More precisely, the evolving rational-weighted part of $\mathcal{N}$ is made up of a single designated processor x_e. The neuron x_e receives as sole incoming synaptic connection a background activity of evolving intensity $c_e(t)$. The synaptic weight $c_e(t)$ successively takes the rational bounded values $q_1, q_2, q_3, \ldots$, by switching from value q_k to q_{k+1} after every N_k time steps, for some large enough $N_k > 0$ to be described. In this way, every time some new value q_{i+1} appears as a background activity of neuron x_e, the network stores it in a designated neuron in order to be able to perform the instruction "access to the value q_{i+1}" when required.

The static rational-weighted part of $\mathcal{N}$ is designed in order to perform the successive recursive steps of Procedure 1, every time some new value q_{i+1} has appeared by means of the activation value of neuron x_e. The equivalence result between rational-weighted RNNs and TMs ensures that such a static rational-weighted sub-network of $\mathcal{N}$ performing these recursive step can indeed always be constructed [59]. Moreover, for each $k > 0$, the time interval N_k between the apparition of the synaptic weights q_k and q_{k+1} is chosen large enough in order to be able to perform all the aforementioned recursive steps.

Finally, the network $\mathcal{N}$ is designed in such a way that it outputs via its data and validation cells y_d and y_v the finite word $v - u$ every time it simulates the instruction "output $v - u$ bit by bit" of Procedure 1. The network keeps outputting λ symbols every time it simulates any other internal instruction of Procedure 1.

It remains to prove that the network $\mathcal{N}$ realizes ψ, i.e. that $\varphi_{\mathcal{N}} = \psi$. Note that, for any input stream $s \in \{0,1\}^{\omega}$, the finite word that has been output at the end of each instruction "output $v - u$ bit by bit" corresponds precisely to the finite word $f(s[0{:}i])$ currently stored in the variable v. Hence, after infinitely many time steps, the finite or infinite word $\varphi_{\mathcal{N}}(s)$ output by $\mathcal{N}$ contains each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix. In other words, $f(s[0{:}i]) \subseteq \varphi_{\mathcal{N}}(s)$ for all $i \geq 0$.

We now consider in turn the two possible cases where $\varphi_{\mathcal{N}}(s)$ is either infinite or finite. First, if $\varphi_{\mathcal{N}}(s)$ is infinite, then it means that Procedure 1 has never stopped outputting new bits from some time step onwards, i.e., $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$. Consequently, the two properties $f(s[0{:}i]) \subseteq \varphi_{\mathcal{N}}(s) \in \{0,1\}^{\omega}$ for all $i \geq 0$ and $\lim_{i\to\infty} |f(s[0{:}i])| = \infty$ ensure that $\varphi_{\mathcal{N}}(s)$ is the unique infinite word containing each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say by definition that $\varphi_{\mathcal{N}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$. Second, if $\varphi_{\mathcal{N}}(s)$ is finite, it means that Procedure 1 has stopped outputting new bits from some time step onwards, and hence $\varphi_{\mathcal{N}}(s) = f(s[0{:}j])$ for some $j \geq 0$. In this case, the two properties $f(s[0{:}i]) \subseteq \varphi_{\mathcal{N}}(s) \in \{0,1\}^{*}$ for all $i \geq 0$ and $\varphi_{\mathcal{N}}(s) \in \{f(s[0{:}i]) : i \geq 0\}$ ensure that $\varphi_{\mathcal{N}}(s)$ is the smallest finite word that contains each word of $\{f(s[0{:}i]) : i \geq 0\}$ as a finite prefix, which is to say by definition that $\varphi_{\mathcal{N}}(s) = \lim_{i\geq 0} f(s[0{:}i]) = f_{\omega}(s)$.

Therefore, $\varphi_{\mathcal{N}} = f_{\omega}$, and since $f_{\omega} = \psi$, it follows that $\varphi_{\mathcal{N}} = \psi$, meaning that ψ is realized by $\mathcal{N}$. This concludes the proof. □

Finally, the computational capabilities of interactive RNNs, stated by previous theorems 4, 5, and 6 follow the same pattern as those of classical RNNs. The results are summarized in Table 2 below.

Table 2 Computational power of interactive static and evolving RNNs according to the nature of their synaptic weights

	Interactive Static RNNs	Interactive Evolving RNNs
$\mathbb{Q}$	Turing	super-Turing
$\mathbb{R}$	super-Turing	super-Turing

8 Universality

Theorems 5 and 6 together with Proposition 1 show that the four models of I-St-RNN[$\mathbb{R}$]s, I-Ev-RNN[$\mathbb{Q}$]s, I-Ev-RNN[$\mathbb{R}$]s, and I-TM/As are capable to capture all possible computations performable by some deterministic interactive system. More precisely, for any possible interactive deterministic systems $\mathscr{S}$, there exists an I-St-RNN[$\mathbb{R}$] $\mathscr{N}_1$, an I-Ev-RNN[$\mathbb{Q}$] $\mathscr{N}_2$, an I-Ev-RNN[$\mathbb{R}$] $\mathscr{N}_3$, and an I-TM/A $\mathscr{M}$ such that $\varphi_{\mathscr{N}_1} = \varphi_{\mathscr{N}_2} = \varphi_{\mathscr{N}_3} = \varphi_{\mathscr{M}} = \varphi_{\mathscr{S}}$. In this sense, those four models of interactive computation are called *universal*.

Theorem 7. *The four models of computations that are I-St-RNN[$\mathbb{R}$]s, I-Ev-RNN[$\mathbb{Q}$]s, I-Ev-RNN[$\mathbb{R}$]s, and I-TM/As, are super-Turing universal.*

Proof. Let $\mathscr{S}$ be some deterministic interactive system. By Proposition 1, $\varphi_{\mathscr{S}}$ is continuous. By Theorems 5 and 6, $\varphi_{\mathscr{S}}$ is realizable by some I-St-RNN[$\mathbb{R}$], by some I-Ev-RNN[$\mathbb{Q}$], by some I-Ev-RNN[$\mathbb{R}$], and by some I-TM/A. $\square$

These results can be understood as follows: similarly to the classical framework, where every possible partial function from integers to integers can be computed by some Turing machine with oracle [64], in the interactive framework, every possible ω-translation performed in an interactive way can be computed by some interactive Turing machine with advice, or equivalently, by some interactive analog or evolving recurrent neural network. Alternatively put, as in the classical framework, where the model of a Turing machine with oracle exhausts the class of all possible partial functions from integers to integers, in the interactive framework, the model of an interactive Turing machine with advice or those of an interactive analog or evolving recurrent neural network also exhaust the class of all possible ω-translations performed in an interactive way.

9 Discussion

We showed that interactive rational- and real-weighted RNNs are Turing-equivalent and super-Turing, respectively (theorems 4, 5). Furthermore, interactive evolving RNNs are also super-Turing, irrespective of whether their synaptic weights are modeled by rational or real numbers (Theorem 6). The comparison between theorems 1, 2, 3 and theorems 4, 5, 6 shows that the computational powers of RNNs involved in a classical or in an interactive computational framework follow similar patterns of characterization. These results are summarized in tables 1 and 2, respectively.

These achievements show that in both classical and interactive computational framework, the translations from the static rational to the static real context, as well as from the static rational to the evolving rational context, do bring additional power to the underlying neural networks. By contrast, the two other translations from the evolving rational to the evolving real context, as well as from the static real to the evolving real context, do not increase further the capabilities of the neural networks.

Furthermore, according to theorems 1, 2, 3, 4, 5, 6, the computational capabilities of all neural models studied so far are shown to be upper bounded by those of the Turing machine with advice model. In the classical computational context, these considerations support the *Thesis of Natural Computation*, which states that every natural computational phenomenon can be captured by the Turing machine with polynomial advice model [59]. In the interactive framework, they support the *Church-Turing Thesis of Interactive Computation* which claims that "any (non-uniform interactive) computation can be described in terms of interactive Turing machines with advice" [70].

Hence, similarly to the Turing machine model which represents a definitely relevant conceptualization of current algorithmic, the interactive Turing machine with advice model also seems to encompass a particularly suitable conceptualization of brain computation, or even of natural computation in general[55, 7, 73], since it is capable to capture crucial features, like analogue considerations [60], evolvability[10, 9, 14], chaotic behaviors [63], that are impossible to be achieved via the simple Turing machine model.

The results also show that the incorporation of general *evolving capabilities* in a neuronal-based computational model naturally leads to the emergence of super-Turing computational capabilities. In fact, tables 1 and 2 show that the incorporation of either *evolving capabilities* or some *power of the continuum* in a basic neural model provides an alternative and equivalent way towards the achievement of super-Turing computational capabilities. Although being mathematically equivalent in this sense, these two features are nevertheless conceptually well distinct. While the power of the continuum is a pure conceptualisation of the mind, the evolving capabilities of the networks are, by contrast, observable in nature.

These achievements support the claim that the general mechanism of plasticity is crucially involved in the computational and dynamical capabilities of biological neural networks, and in this sense, provides a new theoretical complement to the numerous experimental studies emphasizing the importance of the general mechanism of plasticity in brain's information processing [1, 18, 30]. They further suggest that some intrinsic computational capabilities of the brain might lie beyond the scope of Turing-equivalent models of computation, and hence surpass the potentialities every current standard artificial models of computation.

Finally, we believe that the present work presents some interest far beyond the question of the existence of hypercomputational capabilities in nature [16, 17]. Comparative studies about the computational power of more and more biologically oriented neural models might ultimately bring further insight to the understanding of the intrinsic natures of biological as well as artificial intelligences. Furthermore, foundational approaches to alternative models of computation might in the long term not only lead to relevant theoretical considerations, but also to practical applications. Similarly to the theoretical work from Turing which played a crucial role in the practical realization of modern computers, further foundational considerations of alternative models of computation will certainly contribute to the emergence of novel computational technologies and computers, and step by step, open the way to the next computational era.

References

1. Abbott, L.F., Nelson, S.B.: Synaptic plasticity: taming the beast. Nat. Neurosci. 3(suppl.), 1178–1183 (2000)
2. Alon, N., Dewdney, A.K., Ott, T.J.: Efficient simulation of finite automata by neural nets. J. ACM 38(2), 495–514 (1991)
3. Alquźar, R., Alberto, S.: An algebraic framework to represent finite state machines in single-layer recurrent neural networks. Neural Computation 7(5), 931–949 (1995)
4. Arbib, M.A.: On Modelling the Nervous System. In: von Gierke, H.E., Keidel, W.D., Oestreicher, H.L. (eds.) Principles and Practice of Bionics, Proc. 44th. AGARD—Conference Brüssel, ch. 1-2, pp. 43–58. The Advisory Group for Aerospace Research and Development, NATO (1970)
5. Balcázar, J.L., Gavaldà, R., Siegelmann, H.T.: Computational power of neural networks: a characterization in terms of kolmogorov complexity. IEEE Transactions on Information Theory 43(4), 1175–1183 (1997)
6. Ben-Hur, A., Roitershtein, A., Siegelmann, H.T.: On probabilistic analog automata. Theor. Comput. Sci. 320(2-3), 449–464 (2004)
7. Bournez, O., Cosnard, M.: On the computational power of dynamical systems and hybrid systems. Theoretical Computer Science 168(2), 417–459 (1996)
8. Cabessa, J., Villa, A.E.P.: An attractor-based complexity measurement of boolean recurrent neural networks. Plos One (to appear, 2014)
9. Cabessa, J.: Interactive evolving recurrent neural networks are super-Turing. In: Filipe, J., Fred, A.L.N. (eds.) ICAART (1), pp. 328–333. SciTePress (2012)
10. Cabessa, J., Siegelmann, H.T.: Evolving recurrent neural networks are super-Turing. In: IJCNN, pp. 3200–3206. IEEE (2011)
11. Cabessa, J., Siegelmann, H.T.: The computational power of interactive recurrent neural networks. Neural Computation 24(4), 996–1019 (2012)
12. Cabessa, J., Villa, A.E.P.: A hierarchical classification of first-order recurrent neural networks. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 142–153. Springer, Heidelberg (2010)
13. Cabessa, J., Villa, A.E.P.: The expressive power of analog recurrent neural networks on infinite input streams. Theor. Comput. Sci. 436, 23–34 (2012)

14. Cabessa, J., Villa, A.E.P.: The super-Turing computational power of interactive evolving recurrent neural networks. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 58–65. Springer, Heidelberg (2013)
15. Chechik, G., Meilijson, I., Ruppin, E.: Neuronal regulation: A mechanism for synaptic pruning during brain maturation. Neural Comput. 11, 2061–2080 (1999)
16. Copeland, B.J.: Hypercomputation. Minds Mach. 12(4), 461–502 (2002)
17. Copeland, B.J.: Hypercomputation: philosophical issues. Theor. Comput. Sci. 317(1-3), 251–267 (2004)
18. Destexhe, A., Marder, E.: Plasticity in single neuron and circuit computations. Nature 431(7010), 789–795 (2004)
19. Franklin, S., Garzon, M.: Neural computability. In: Omidvar, O. (ed.) Progress in Neural Networks, pp. 128–144. Ablex, Norwood (1989)
20. Garzon, M., Franklin, S.: Neural computability II. In: Omidvar, O. (ed.) Proceedings of the Third International Joint Conference on Neural Networks, pp. 631–637. IEEE (1989)
21. Goldin, D.Q.: Persistent Turing machines as a model of interactive computation. In: Schewe, K.-D., Thalheim, B. (eds.) FoIKS 2000. LNCS, vol. 1762, pp. 116–135. Springer, Heidelberg (2000)
22. Goldin, D., Smolka, S.A., Attie, P.C., Sonderegger, E.L.: Turing machines, transition systems, and interaction. Inf. Comput. 194, 101–128 (2004)
23. Goldin, D., Smolka, S.A., Wegner, P.: Interactive Computation: The New Paradigm. Springer-Verlag New York, Inc., Secaucus (2006)
24. Goldin, D., Wegner, P.: The Church-Turing thesis: Breaking the myth. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 152–168. Springer, Heidelberg (2005)
25. Goldin, D., Wegner, P.: Principles of interactive computation. In: Goldin, D., Smolka, S.A., Wegner, P. (eds.) Interactive Computation, pp. 25–37. Springer, Heidelberg (2006)
26. Goldin, D., Wegner, P.: The interactive nature of computing: Refuting the strong Church-Turing thesis. Minds Mach. 18, 17–38 (2008)
27. Goudreau, M.W., Giles, C.L., Chakradhar, S.T., Chen, D.: First-order versus second-order single-layer recurrent neural networks. IEEE Transactions on Neural Networks 5(3), 511–513 (1994)
28. Hartley, R., Szu, H.: A comparison of the computational power of neural network models. In: Butler, C. (ed.) Proceedings of the IEEE First International Conference on Neural Networks, pp. 17–22. IEEE (1987)
29. Hebb, D.O.: The organization of behavior: a neuropsychological theory. John Wiley & Sons Inc. (1949)
30. Holtmaat, A., Svoboda, K.: Experience-dependent structural synaptic plasticity in the mammalian brain. Nat. Rev. Neurosci. 10(9), 647–658 (2009)
31. Horne, B.G., Hush, D.R.: Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Networks 9(2), 243–252 (1996)
32. Hyötyniemi, H.: Turing machines are recurrent neural networks. In: Proceedings of STEP 1996, pp. 13–24. Finnish Artificial Intelligence Society (1996)
33. Iglesias, J., Villa, A.E.P.: Emergence of preferred firing sequences in large spiking neural networks during simulated neuronal development. Int. J. Neural Syst. 18(4), 267–277 (2008)

34. Innocenti, G.M., Price, D.J.: Exuberance in the development of cortical networks. Nature Rev. Neurosci. 6, 955–965 (2005)
35. Kaneko, K., Tsuda, I.: Chaotic itinerancy. Chaos 13(3), 926–936 (2003)
36. Kasabov, N.: Evolving connectionist systems - the knowledge engineering approach, 2nd edn. Springer (2007)
37. Kilian, J., Siegelmann, H.T.: The dynamic universality of sigmoidal neural networks. Inf. Comput. 128(1), 48–56 (1996)
38. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, pp. 3–41. Princeton University Press, Princeton (1956)
39. Kremer, S.C.: On the computational power of elman-style recurrent networks. IEEE Transactions on Neural Networks 6(4), 1000–1004 (1995)
40. Maass, W., Orponen, P.: On the effect of analog noise in discrete-time analog computations. Neural Comput. 10(5), 1071–1095 (1998)
41. Maass, W., Sontag, E.D.: Analog neural nets with gaussian or other common noise distributions cannot recognize arbitary regular languages. Neural Comput. 11(3), 771–782 (1999)
42. Martin, S.J., Grimwood, P.D., Morris, R.G.M.: Synaptic plasticity and memory: An evaluation of the hypothesis. Annu. Rev. Neurosci. 23(1), 649–711 (2000); PMID: 10845078
43. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysic 5, 115–133 (1943)
44. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Englewood Cliffs (1967)
45. Minsky, M.L., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge (1969)
46. Neto, J.P., Siegelmann, H.T., Costa, J.F., Araujo, C.P.S.: Turing universality of neural nets (revisited). In: Moreno-Díaz, R., Pichler, F. (eds.) EUROCAST 1997. LNCS, vol. 1333, pp. 361–366. Springer, Heidelberg (1997)
47. von Neumann, J.: The computer and the brain. Yale University Press, New Haven (1958)
48. Omlin, C.W., Giles, C.L.: Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants. Neural Computation 8(4), 675–696 (1996)
49. Pollack, J.B.: On Connectionist Models of Natural Language Processing. PhD thesis, Computing Reseach Laboratory, New Mexico State University, Las Cruces, NM (1987)
50. Roberts, P.D., Bell, C.C.: Spike timing dependent synaptic plasticity in biological systems. Biol. Cybern. 87, 392–403 (2002)
51. Rosenblatt, F.: The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York (1957)
52. Schmidhuber, J.: Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem (Dynamic neural nets and the fundamental spatio-temporal credit assignment problem). PhD thesis, Institut für Informatik, Technische Universität München (1990)
53. Shatz, C.J.: Impulse activity and the patterning of connections during CNS development. Neuron 5, 745–756 (1990)
54. Shmiel, T., Drori, R., Shmiel, O., Ben-Shaul, Y., Nadasdy, Z., Shemesh, M., Teicher, M., Abeles, M.: Neurons of the cerebral cortex exhibit precise interspike timing in correspondence to behavior. Proc. Natl. Acad. Sci.d U S A 102(51), 18655–18657 (2005)
55. Siegelmann, H.T.: Computation beyond the Turing limit. Science 268(5210), 545–548 (1995)

56. Siegelmann, H.T.: Neural networks and analog computation: beyond the Turing limit. Birkhauser Boston Inc., Cambridge (1999)
57. Siegelmann, H.T.: Stochastic analog networks and computational complexity. J. Complexity 15(4), 451–475 (1999)
58. Siegelmann, H.T.: Neural and super-Turing computing. Minds Mach. 13(1), 103–114 (2003)
59. Siegelmann, H.T., Sontag, E.D.: Analog computation via neural networks. Theor. Comput. Sci. 131(2), 331–360 (1994)
60. Siegelmann, H.T., Sontag, E.D.: On the computational power of neural nets. J. Comput. Syst. Sci. 50(1), 132–150 (1995)
61. Taylor, J.G., Villa, A.E.P.: The "Conscious I": A Neuroheuristic Approach to the Mind. In: Baltimore, D., Dulbecco, R., Jacob, F., Montalcini, R.L. (eds.) Frontiers of Life, vol. III, pp. 349–270. Academic Press (2001) ISBN: 0-12-077340-6
62. Tsuda, I.: Chaotic itinerancy as a dynamical basis of hermeneutics of brain and mind. World Futures 32, 167–185 (1991)
63. Tsuda, I.: Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. Behav. Brain Sci. 24(5), 793–847 (2001)
64. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. 2(42), 230–265 (1936)
65. Turing, A.M.: Intelligent machinery. Technical report, National Physical Laboratory, Teddington, UK (1948)
66. Turova, T.S.: Structural phase transitions in neural networks. Math. Biosci. Eng. 11(1), 139–148 (2014)
67. Turova, T.S., Villa, A.E.P.: On a phase diagram for random neural networks with embedded spike timing dependent plasticity. Biosystems 89(1-3), 280–286 (2007)
68. van Leeuwen, J., Wiedermann, J.: On algorithms and interaction. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 99–113. Springer, Heidelberg (2000)
69. van Leeuwen, J., Wiedermann, J.: Beyond the Turing limit: Evolving interactive systems. In: Pacholski, L., Ružička, P. (eds.) SOFSEM 2001. LNCS, vol. 2234, pp. 90–109. Springer, Heidelberg (2001)
70. van Leeuwen, J., Wiedermann, J.: The Turing machine paradigm in contemporary computing. In: Engquist, B., Schmid, W. (eds.) Mathematics Unlimited - 2001 and Beyond. LNCS, pp. 1139–1155. Springer, Heidelberg (2001)
71. van Leeuwen, J., Wiedermann, J.: The emergent computational potential of evolving artificial living systems. AI Commun. 15, 205–215 (2002)
72. van Leeuwen, J., Wiedermann, J.: A theory of interactive computation. In: Goldin, D., Smolka, S.A., Wegner, P. (eds.) Interactive Computation, pp. 119–142. Springer, Heidelberg (2006)
73. Wiedermann, J., van Leeuwen, J.: How we think of computing today. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 579–593. Springer, Heidelberg (2008)
74. Villa, A.E.P., Tetko, I.V., Hyland, B., Najem, A.: Spatiotemporal activity patterns of rat cortical neurons predict responses in a conditioned task. Proc. Natl. Acad. Sci. U S A 96(3), 1106–1111 (1999)
75. Villa, A.E.P.: Neural Coding in the Neuroheuristic Perspective. In: Barbieri, M. (ed.) The Codes of Life: The Rules of Macroevolution, ch. 16. Biosemiotics, vol. 1, pp. 357–377. Springer, Berlin (2008)
76. Watts, M.J.: A decade of kasabov's evolving connectionist systems: A review. IEEE Transactions on Systems, Man, and Cybernetics, Part C 39(3), 253–269 (2009)

77. Wegner, P.: Why interaction is more powerful than algorithms. Commun. ACM 40, 80–91 (1997)
78. Wegner, P.: Interactive foundations of computing. Theor. Comput. Sci. 192, 315–351 (1998)
79. Wiener, N.: Cybernetics Or Control And Communication In The Animal And The Machine. John Wiley & Sons Inc. (1948)

Image Classification with Nonnegative Matrix Factorization Based on Spectral Projected Gradient

Rafał Zdunek, Anh Huy Phan, and Andrzej Cichocki

Abstract. Nonnegative Matrix Factorization (NMF) is a key tool for model dimensionality reduction in supervised classification. Several NMF algorithms have been developed for this purpose. In a majority of them, the training process is improved by using discriminant or nearest-neighbor graph-based constraints that are obtained from the knowledge on class labels of training samples. The constraints are usually incorporated to NMF algorithms by l_2-weighted penalty terms that involve formulating a large-size weighting matrix. Using the Newton method for updating the latent factors, the optimization problems in NMF become large-scale. However, the computational problem can be considerably alleviated if the modified Spectral Projected Gradient (SPG) that belongs to a class of quasi-Newton methods is used. The simulation results presented for the selected classification problems demonstrate the high efficiency of the proposed method.

1 Introduction

Nonnegative Matrix Factorization (NMF) [20] decomposes a nonnegative matrix into lower-rank factor matrices that have nonnegative entries and usually some physical meaning. When NMF is applied to the matrix of training samples, we obtain sparse nonnegative feature vectors and coefficients of their nonnegative combinations. The vectors of the coefficients, which are here referred to as encoding

Rafał Zdunek
Department of Electronics, Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
e-mail: rafal.zdunek@pwr.wroc.pl

Anh Huy Phan
Laboratory for Advanced Brain Signal Processing
RIKEN BSI, Wako-shi, Japan
e-mail: phan@brain.riken.jp

Andrzej Cichocki
Laboratory for Advanced Brain Signal Processing
RIKEN BSI, Wako-shi, Japan
e-mail: a.cichocki@riken.jp

Systems Research Institute, Polish Academy of Science (PAN) Warsaw, Poland

vectors, lie in a low-dimensional latent component space. Hence, NMF is often regarded as a dimensionality reduction technique, and it has been widely applied for classification of various objects [2, 6, 12, 29, 33].

As reported in [8], the factor matrices obtained with NMF are generally non-unique. Several attempts have been done to impose additional constraints on the estimated factors. The sparsity constraints are probably the most frequently used. It is well-known that the factors should be somehow sparse to expect the uniqueness [16]. Hoyer [14] demonstrated that the sparsity can be readily controlled in NMF by the l_1-norm based penalty term in an objective function. When l_2-norm is used instead, the smoothness is enforced in one factor, and intrinsically the sparsity level increases in the other [26]. The sparse NMF that minimizes the l_p-diversity measure was proposed in [37]. This approach was also developed by Kim and Park in [17] using the active-set minimization strategy. The sparsity constraints were also analyzed in the context of supervised classification. Li *et al* [21] proposed the Local NMF (LNMF) that combines several kinds of constraints. It enforces orthogonality between basis vectors and selects the most meaningful encoding vectors. As a result, the basis vectors contain more localized features, which usually leads to better classification results.

The training process can be also improved by using the Fisher's discriminant information. The examples include hybrid methods, such as FNMF (*Fisher NMF*) or DNMF (*Discriminant NMF*), which combine NMF with FDA (*Fisher's Discriminant Analysis*) or LDA (*Linear Discriminant Analysis*). The first was proposed by Wang *et al.* [31], and the second developed by Zafeiriou *et al.* [33]. In these hybrids, the penalty terms are constructed in such a way to minimize the inner-class scattering and simultaneously to maximize intra-class scattering. The inner-class scatter models the dispersion of vectors that belong to the same class around their mean, while the intra-class scatter expresses the distances of the local class means from the global mean. As a result, the well-known Fisher discriminant criterion is maximized. Both FNMF and DNMF are based on the multiplicative algorithm that minimizes the penalized Kullback-Leibler (KL) divergence.

Another group contains the NMF algorithms which explore a geometrical structure of observed data. Cai *et al.* [4, 5] noticed that samples in a latent space lie on a low-dimensional manifold embedded in a high-dimensional ambient space. Thus, the projection from a high-dimensional observation space to a low-dimensional latent space should preserve a data geometrical structure. The neighboring samples should belong to the same class in both spaces. Thus, they proposed Graph regularized NMF (GNMF) [5] that constrains one of the factor matrices with the information on the data geometric structure encoded in a nearest-neighbor graph of training samples. This constraint was imposed to NMF by a specifically designed regularization term in an objective function that was then minimized with the standard multiplicative algorithm. This approach combines NMF with the Laplacian eigenmaps [1] and locality-preserving projections [13]. Yang *et al.* [32] divided the coefficients of the encoding vectors into two parts to which the discriminant information is incorporated accordingly. The first part is enforced by the intrinsic graph that characterizes the favorite relationships among the training data. The other is affected by the penalty graph that expresses the unfavorable relationships. This approach was then computationally improved and extended to tensor decompositions by Wang *et al.* [30], and to the projective NMF by Liu *et al.* [24].

An interesting property of nearly all the above mentioned learning methods is the common way of introducing the prior knowledge to the training process. The penalty terms in the regularized objective function can be expressed in terms of the weighted Frobenius norm. The weighting matrix reflects the group sparsity, discriminant information or graph-based embedding. Its size is usually equal to the number of training samples, so it is large and usually sparse matrix.

The penalty term expressed by the the weighted Frobenius norm can be easily considered in an optimization process when the multiplicative algorithm is used. Hence, this approach is explored in a large number of research papers on NMF. However, multiplicative algorithms are slowly convergent, and in the basic version they do not guarantee convergence to a stationary point. Using the simple numerical improvements, as proposed in [22], the algorithms can be numerically stable but the slow convergence is still an open problem.

To tackle the convergence problems, several other computational strategies have been proposed in the literature. Guan *et al.* [10] considerably accelerated the convergence of GNMF by using additive quasi-Newton gradient descent updates. In the next paper, Guan *et al.* [11] proposed the NeNMF that is based on the Nesterovs optimal gradient approach. This method can be used for minimization of the above-mentioned penalized objective functions, provided that the Lipschitz constant can be easily calculated. A simple version of the Projected Gradient (PG) algorithm was also used for classification problems in [18].

The optimization problems that include the weighted Frobenius norm-based penalty terms can be also efficiently solved using the Hierarchical Alternating Least Squares (HALS) algorithm [6]. Phan *et al.* [27, 28] applied the HALS algorithm to obtain multi-way array decomposition with higher-order discriminant analysis. A similar computational strategy, known as the Sequential Coordinate-Wise (SCW) [9], was used in [38] for updating lateral factors in DNMF. As a result, the SCW-DNMF substantially outperforms the basic DNMF and LNMF algorithms but at the cost of an increased computational complexity.

GNMF can be also efficiently obtained with the Spectral Gradient Projection (SPG) method. This approach was proposed in [39] in the context of facial image classification. The SPG [25] belongs to a class of quasi-Newton methods. It approximates the Hessian matrix with the scalar that is estimated from the secant equation, separately for each column vector of the encoding matrix. Such computations can be easily parallelized, which leads to a high performance with a low computational cost. Unfortunately, the steplengths in gradient descent updates cannot be so easily determined for the penalty terms used in GNMF. Hence, the Armijo rule was used in [39], similarly as in the PG NMF algorithm proposed by Lin [23].

In this chapter, we extend the SPG algorithm discussed in [39] in several aspects: (1) we propose a better computational strategy for estimating the steplengths, separately for each column vector of the encoding matrix; (2) we adapt this algorithm for solving generalized weighted Frobenius norm-based penalty terms, including sparsity and discriminant information; (3) we present the results for more classification problems and demonstrate the efficiency of using preprocessing based on the wavelet transform.

The chapter is organized in the following way. The next section discusses the penalty terms in NMF that are expressed by the weighted Frobenius norm. Section 3 is concerned with the optimization algorithms. The numerical experiments for image classification problems are presented in Section 4. Finally, the conclusions are drawn in Section 5.

2 Penalty Terms

Let $Y = [y_1, \ldots, y_T] \in \mathbb{R}_+^{I \times T}$, where $y_t \in \mathbb{R}_+^I$ is the t-th training sample. Applying NMF to Y, we get $Y \cong AX$, where the columns of the matrix $A \in \mathbb{R}_+^{I \times J}$ represent the feature or basis vectors, and the columns of the matrix $X \in \mathbb{R}_+^{J \times T}$ are encoding vectors. The parameter J is the rank of factorization.

In several variants of NMF, the objective function can be expressed by the quadratic function:

$$\begin{aligned}\Psi(A,X) &= \frac{1}{2}||Y - AX||_F^2 + \frac{\alpha_A}{2}\operatorname{tr}(A^T L_A A) + \frac{\alpha_X}{2}\operatorname{tr}(X L_X X^T) \\ &= \frac{1}{2}||Y - AX||_F^2 + \frac{\alpha_A}{2}||L_A^{\frac{1}{2}} A||_F^2 + \frac{\alpha_X}{2}||X L_X^{\frac{1}{2}}||_F^2, \end{aligned} \tag{1}$$

where $\frac{\alpha_A}{2}||L_A^{\frac{1}{2}} A||_F^2$ and $\frac{\alpha_X}{2}||X L_X^{\frac{1}{2}}||_F^2$ are *penalty terms*, expressed in terms of the weighted Frobenius norm, and $\alpha_A, \alpha_X \geq 0$ are penalty parameters that control the amounts of introduced *a priori* information. The weighting matrices $L_A \in \mathbb{R}^{I \times I}$ and $L_X \in \mathbb{R}^{T \times T}$ are symmetric and nonnegative definite. Both matrices are determined on the basis of the prior knowledge on the estimated factors. The matrix L_A enforces column profiles in the matrix A, and L_X – row profiles in X. Below we present a short survey of the typical penalty terms.

2.1 Sparse NMF

The sparsity in the factor X can be modeled in many ways, e.g. by the l_p-diversity measure [7]: $J^{(p,q)} = \sum_{j=1}^J (||\underline{x}_j||_q)^p$ for $p \geq 0$, $q \geq 1$, where $\underline{x}_j$ is the j-th row vector of X. In [37], we assumed $q = 1$ and $p = 2$ to enforce sparsity in the columns of X. Note that the sparsity can be also enforced in the rows, if $q = 2$ and $p = 1$. This leads to the term: $J^{(1,2)} = \sum_{j=1}^J (||\underline{x}_j||_1)^2 = \operatorname{tr}\{X E_T X^T\}$, where $E_T \in \mathbb{R}^{T \times T}$ is a matrix of all ones. Considering (1), we have $L_X = E_T$. The sparsity can be also modeled in a similar way in the column vectors of A, which leads to $L_A = E_I \in \mathbb{R}^{I \times I}$. Such sparsity measures were also used in the SNMF/L and SNMF/R algorithms [17].

Applying the Hoyer's sparsity measure [15] to the rows of X, we have:

$$\sigma = \frac{\sqrt{T} - \frac{||\underline{x}_j||_1}{||\underline{x}_j||_2}}{\sqrt{T}-1}. \tag{2}$$

Following [19], this measure can be approximated by the additive form:

$$J = \frac{1}{2}\sum_{j=1}^{J} ||\underline{x}_j||_1^2 - ||\underline{x}_j||_2^2 = \mathrm{tr}\{XL_X X^T\}, \tag{3}$$

where $L_X = E_T - I_T$, where $I_T \in \mathbb{R}^{T\times T}$ is an identity matrix. Similarly, $L_A = E_I - I_I$. In this case, $\mathrm{rank}(L_X) = T$, which leads to better numerical properties than for the previous case ($\mathrm{rank}(E_T) = 1$).

2.2 DNMF

The objective function in DNMF [31, 33] has the following form:

$$\Psi(A,X) = D_{KL}(Y||AX) + \gamma\mathrm{tr}\{S_X\} - \delta\,\mathrm{tr}\{S_{\bar{X}}\}, \tag{4}$$

where $D_{KL}(Y||AX)$ is the KL divergence between Y and AX. The matrices S_X and $S_{\bar{X}}$ represent the inner- and intra-scattering, respectively. They are defined in the following way:

$$S_X = \sum_{k=1}^{K}\sum_{t_k=1}^{|\mathcal{N}_k|}(x_{t_k} - \bar{x}_k)(x_{t_k} - \bar{x}_k)^T, \tag{5}$$

$$S_{\bar{X}} = \sum_{k=1}^{K} |\mathcal{N}_k|(\bar{x}_k - \bar{x})(\bar{x}_k - \bar{x})^T, \tag{6}$$

where K is the number of classes, $\mathcal{N}_k$ is the set of indices of the samples x_t that belong to the k-th class, $|\mathcal{N}_k|$ is the number of samples in the k-th class, x_{t_k} is the t_k-th image in the k-th class, $\bar{x}_k$ is the mean vector of the k-th class, and $\bar{x}$ is the mean vector over all the column vectors in X.

Let $C = [c_{st}] \in \mathbb{R}^{T\times T}$, where

$$c_{st} = \begin{cases} \frac{1}{\mathcal{N}_k} & \text{if } (s,t) \in \mathcal{N}_k \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

for $k = 1,\ldots,K$. Considering (7), the matrix S_X in (5) can be rearranged as follows:

$$S_X = (X - XC)(X - XC)^T = XL_X^{(1)}X^T, \tag{8}$$

where $L_X^{(1)} = (I_T - C)(I_T - C)^T$. Similarly, the matrix $S_{\bar{X}}$ in (6) can be rewritten as

$$S_{\bar{X}} = (XC - X\tilde{E}_T)(XC - X\tilde{E}_T)^T = XL_X^{(2)}X^T, \tag{9}$$

where $\tilde{E}_T = \frac{1}{T}11^T$, $1 = [1,1,\ldots,1]^T \in \mathbb{R}^T$, and $L_X^{(2)} = (C - \tilde{E}_T)(C - \tilde{E}_T)^T$.

Thus, the penalty terms in (4) can be reformulated as

$$\gamma \operatorname{tr}\{S_X\} - \delta \operatorname{tr}\{S_{\bar{X}}\} = \operatorname{tr}\{XL_XX^T\} = ||XL_X^{\frac{1}{2}}||_F^2, \tag{10}$$

where $L_X = \gamma L_X^{(1)} - \delta L_X^{(2)}$. The penalty term in (10) can be combined not only with the KL divergence but also with other disimilarity measures. In [39], this modeling was used for deriving the SCW-DNMF algorithm.

2.3 GNMF

GNMF [5] integrates NMF with the Laplacian eigenmaps [1] and locality-preserving projections [13]. The matrix L_X in GNMF is expressed by the graph Laplacian matrix that represents a data geometrical structure in the observation space. It takes form: $L_X = D - W$, where $W = [w_{nm}] \in \mathbb{R}_+^{T\times T}$ contains the entries that determine the edges in the nearest neighbor graph of the observed points, and $D = \operatorname{diag}\left(\sum_{m\neq n} w_{nm}\right) \in \mathbb{R}_+^{T\times T}$. The edges can be determined by the hard connections:

$$w_{nm} = \begin{cases} 1, \text{ if } & y_n \in \mathcal{N}_p(y_m), \text{ or } y_m \in \mathcal{N}_p(y_n), \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

where $\mathcal{N}_p(y_t)$ is the p nearest neighbor of the sample y_t. We can also use the Heat kernel weighting:

$$w_{nm} = \begin{cases} \exp\left\{-\frac{||y_n - y_m||_2^2}{2\sigma^2}\right\}, \text{ if } & y_n \in \mathcal{N}_p(y_m), \text{ or } y_m \in \mathcal{N}_p(y_n), \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

or the cosine measure:

$$w_{nm} = \begin{cases} \frac{y_n^T y_m}{||y_n|| ||y_m||}, \text{ if } & y_n \in \mathcal{N}_p(y_m), \text{ or } y_m \in \mathcal{N}_p(y_n), \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

The graph-based regularization helps to preserve the data geometrical structure in the low dimensional latent space that contains the samples $\{x_t\}$. If any two samples y_{t_1} and y_{t_2} belong to one cluster, the corresponding samples x_{t_1} and x_{t_2} should also belong to the same cluster. Thus, this approach seems to be very useful for clustering but not necessarily for classification problems. For the latter, the discriminant information is more relevant.

The discriminant constraints can be also combined with the graph-based ones. In this approach, the graph Laplacian matrices can be defined separately for the inner- and intra-class samples. Guan *et al.* [10] model k_1 nearest-neighbor inner class samples with the graph Laplacian matrix $L_{(inner)}$, and k_2 nearest-neighbor intra class samples with the graph Laplacian matrix $L_{(intra)}$. The Laplacian eigenmaps can be combined in the following way:

$$L_X = \left(\tilde{L}_{(intra)}^{-\frac{1}{2}}\right)^T L_{(inner)} \tilde{L}_{(intra)}^{-\frac{1}{2}}, \qquad (14)$$

where $\tilde{L}_{(intra)} = L_{(intra)} + \xi I$. The regularization parameter $\xi > 0$ should be selected in such a way to guarantee the inversion of $L_{(intra)}$. Note that the parameter ξ also controls a ratio of inner-to-intra class information. Hence it has a discriminant property, and can be treated as a penalty parameter.

Note that the matrices $L_X^{(1)}$ and $L_X^{(2)}$ given in Section 2.2 can be also integrated using the formula in (14). In this approach, the parameter ξ refers to the ratio $\frac{\gamma}{\delta}$.

3 Algorithm

The penalty term $||XL_X^{\frac{1}{2}}||_F^2$ in (1) can be reformulated as follows:

$$\Psi_r(X) = ||XL_X^{\frac{1}{2}}||_F^2 = ||(L_X^{\frac{1}{2}} \otimes I_J)x||_2^2 = x^T (L_X \otimes I_J)x, \qquad (15)$$

where $x = \mathrm{vec}(X) \in \mathbb{R}^{JT}$ is a vectorized form of X, and $\otimes$ stands for the Kronecker product.

Assuming $L_A = I_I$, the Hessian matrices of $\Psi(A,X)$ in (1) with respect to A and X have the following forms:

$$H_A = \nabla_A^2 \Psi(A,X) = (XX^T + \alpha_A I_J) \otimes I_I \in \mathbb{R}^{IJ \times IJ}, \qquad (16)$$

$$H_X = \nabla_X^2 \Psi(A,X) = I_T \otimes A^T A + \alpha_X L_X \otimes I_J \in \mathbb{R}^{JT \times JT}. \qquad (17)$$

For $\alpha_A > 0$, the matrix H_A is positive definite. Under the assumption of positive definiteness of the matrix L_X, the matrix H_X is also symmetric and positive definite.

The matrix H_A has a block-diagonal structure, and hence the updates of A might be considerably accelerated by transforming the nonnegative least-squares problem: $\min_{A \geq 0} \frac{1}{2}||Y - AX||_F^2 + \frac{\alpha_A}{2}||A||_F^2$ to the normal equations $XX^T A^T = XY^T$ subject to the nonnegativity constraints $A \geq 0$. Then, the solution can be efficiently searched with any iterative solver for solving this kind of problems, e.g. HALS [6], PG [23], FCNNLS [17], etc.

Remark 1. Assuming $(XX^T)^{-1}$ is calculated with the $O(J^3)$ complexity, then one iterative step of the Newton method for updating the matrix A entails the complexity about $O(IJT) + O(J^3 + TJ^2) + O(IJ^2)$. Assuming $J << \min\{I, T\}$, an overall complexity is dominated by $O(IJT)$.

Remark 2. The Hessian H_X is no longer a block-diagonal matrix, hence the Newton updates for X cannot be simplified considerably. The computational complexity of one step of the Newton algorithm is $O(IJT) + O(J^3T^3) + O(JT)$. Thus it is strongly affected by the computational cost of the Hessian inverse.

The updates for X cannot be accelerated in the similar way, however, there is still a possibility of applying some quasi-Newton method without formulating the Hessian H_X directly. Note that the matrix H_X is very large when the number of training samples is large, and it is rather a dense matrix due to the matrix L_X. One of these possibilities is to use the SPG method [3] that combines the standard gradient projection scheme with the nonmonotonic Barzilai-Borwein (BB) method [25]. It is used for minimization of convex functions subject to box-constraints.

In the SPG method, the descent direction $p_t^{(k)}$ for updating the vector x_t in the k-th iteration is defined as follows:

$$p_t^{(k)} = \left[x_t^{(k)} - (\alpha_t^{(k)})^{-1} \nabla_{x_t} \Psi(A, x_t^{(k)}) \right]_+ - x_t^{(k)}, \tag{18}$$

for $\alpha_t^{(k)} > 0$ selected in such a way that the matrix $\alpha_t^{(k)} I_J$ approximates the Hessian matrix.

In [6], this method was adopted to parallel processing of all column vectors in X. Using this approach, we have the update rule:

$$X^{(k+1)} = X^{(k)} + P^{(k)} Z^{(k)}, \tag{19}$$

where $Z^{(k)} = \mathrm{diag}\{\eta^{(k)}\}$. The column vectors of $P^{(k)} \in \mathbb{R}^{J \times T}$ and the entries of the vector $\eta^{(k)} \in \mathbb{R}^T_+$ are descent directions and steplengths for updating the vectors $\{x_t\}$, respectively. According to (18), the matrix $P^{(k)}$ has the form:

$$P^{(k)} = \left[X^{(k)} - G_X^{(k)} D^{(k)} \right]_+ - X^{(k)}, \tag{20}$$

where $G_X^{(k)} = \nabla_X \Psi(A, X^{(k)}) \in \mathbb{R}^{J \times T}$ and $D^{(k)} = \mathrm{diag}\{(\alpha_t^{(k)})^{-1}\} \in \mathbb{R}^{T \times T}$.

The coefficients $\{\alpha_t^{(k)}\}$ can be obtained from the secant equation that is given by $S^{(k)} \mathrm{diag}\{\alpha_t^{(k+1)}\} = W^{(k)}$, where $S^{(k)} = X^{(k+1)} - X^{(k)}$ and $W^{(k)} = \nabla_X \Psi(A, X^{(k+1)}) - \nabla_X \Psi(X^{(k)})$. For the minimization of the objective function (1) with respect to X, the matrix $W^{(k)}$ takes the form: $W^{(k)} = A^T A S^{(k)} + \alpha_X S^{(k)} L_X$. From (19) we have: $S^{(k)} = P^{(k)} Z^{(k)}$. In consequence, the secant equation leads to:

$$\begin{aligned}\alpha^{(k+1)} &= \frac{\operatorname{diag}\left\{(S^{(k)})^T W^{(k)}\right\}}{\operatorname{diag}\left\{(S^{(k)})^T S^{(k)}\right\}} = \frac{\operatorname{diag}\left\{(S^{(k)})^T A^T A S^{(k)} + \alpha_X (S^{(k)})^T S^{(k)} L_X\right\}}{\operatorname{diag}\left\{(S^{(k)})^T S^{(k)}\right\}} \\ &= \frac{\operatorname{diag}\left\{(P^{(k)})^T A^T A P^{(k)} + \alpha_X (P^{(k)})^T P^{(k)} Z^{(k)} L_X (Z^{(k)})^{-1}\right\}}{\operatorname{diag}\left\{(P^{(k)})^T P^{(k)}\right\}} \\ &= \frac{1_J^T \left[P^{(k)} \circledast \left(A^T A P^{(k)} + \alpha_X P^{(k)} Z^{(k)} L_X (Z^{(k)})^{-1}\right)\right]}{1_J^T \left[P^{(k)} \circledast P^{(k)}\right]}, \end{aligned} \tag{21}$$

where $\circledast$ stands for the Hadamard product, and the operation $\operatorname{diag}\{M\}$ creates a vector containing the main diagonal entries of a matrix M. Note that the matrix $Z^{(k)}$ is diagonal, so the product $Z^{(k)} L_X (Z^{(k)})^{-1}$ can be readily calculated.

The steplengths can be estimated by solving the minimization problem:

$$\eta_*^{(k)} = \arg\min_{\eta^{(k)}} \Psi\left(A, X^{(k)} + P^{(k)} \operatorname{diag}\{\eta^{(k)}\}\right). \tag{22}$$

For $\alpha_X = 0$, the objective function $\Psi(A, X^{(k+1)})$ takes the form:

$$\begin{aligned}\Psi(A, X^{(k+1)}) &= \frac{1}{2}||Y - A(X^{(k)} + P^{(k)} D_\eta^{(k)})||_F^2 = \frac{1}{2}\operatorname{tr}\left\{D_\eta^{(k)} (P^{(k)})^T A^T A P^{(k)} D_\eta^{(k)}\right\} \\ &\quad - \operatorname{tr}\left\{(Y - AX^{(k)})^T A P^{(k)} D_\eta^{(k)}\right\} + \text{const}, \end{aligned} \tag{23}$$

where $D_\eta^{(k)} = \operatorname{diag}\{\eta^{(k)}\}$. From the stationarity of $\Psi(A, X)$ with respect to X, we have:

$$\frac{\partial}{\partial \eta}\Psi(A, X^{(k+1)}) = \operatorname{diag}\left\{(P^{(k)})^T A^T A P^{(k)} D_\eta^{(k)}\right\} + \operatorname{diag}\left\{(G_X^{(k)})^T P^{(k)}\right\} \triangleq 0. \tag{24}$$

Hence, the solution to (22) can be presented in a closed-form:

$$\eta_*^{(k)} = -\frac{\operatorname{diag}\left\{(G_X^{(k)})^T P^{(k)}\right\}}{\operatorname{diag}\left\{(P^{(k)})^T A^T A P^{(k)} D_\eta^{(k)}\right\}} = -\frac{1_J^T \left[G_X^{(k)} \circledast P^{(k)}\right]}{1_J^T \left[P^{(k)} \circledast (A^T A P^{(k)})\right]}. \tag{25}$$

For $\alpha_X > 0$, we have:

$$\begin{aligned}\Psi(A, X^{(k+1)}) &= \frac{1}{2}||Y - A(X^{(k)} + P^{(k)} D_\eta^{(k)})||_F^2 + \frac{\alpha_X}{2}||(X^{(k)} + P^{(k)} D_\eta^{(k)}) L_X^{\frac{1}{2}}||_F^2 \\ &= \frac{1}{2}\operatorname{tr}\left\{D_\eta^{(k)} (P^{(k)})^T A^T A P^{(k)} D_\eta^{(k)}\right\} - \operatorname{tr}\left\{(Y - AX^{(k)})^T A P^{(k)} D_\eta^{(k)}\right\} \\ &\quad + \operatorname{tr}\left\{X^{(k)} L_X D_\eta^{(k)} (P^{(k)})^T\right\} + \frac{1}{2}\operatorname{tr}\left\{P^{(k)} D_\eta^{(k)} L_X D_\eta^{(k)} (P^{(k)})^T\right\} \\ &\quad + \text{const}. \end{aligned} \tag{26}$$

Thus

$$\frac{\partial}{\partial \eta}\Psi(A,X^{(k+1)}) = \operatorname{diag}\left\{(P^{(k)})^T A^T A P^{(k)} D_\eta^{(k)}\right\} + \operatorname{diag}\left\{(G_X^{(k)})^T P^{(k)}\right\}$$
$$+ \operatorname{diag}\left\{(P^{(k)})^T X^{(k)} L_X\right\} + \frac{1}{2}\operatorname{diag}\left\{(P^{(k)})^T P^{(k)} D_\eta^{(k)} L_X\right\}$$
$$+ \frac{1}{2}\operatorname{diag}\left\{L_X D_\eta^{(k)} (P^{(k)})^T P^{(k)}\right\} \triangleq 0. \tag{27}$$

From (27), we have:

$$\operatorname{diag}\left\{\operatorname{diag}\{(P^{(k)})^T A^T A P^{(k)}\}\right\} \eta_*^{(k)} + \left(\left[(P^{(k)})^T P^{(k)}\right] \circledast L_X\right) \eta_*^{(k)}$$
$$= -\operatorname{diag}\left\{(G_X^{(k)})^T P^{(k)} + (P^{(k)})^T X^{(k)} L_X\right\},$$

which leads to the following system of linear equations:

$$\tilde{T}^{(k)} \eta_*^{(k)} = -b^{(k)}, \tag{28}$$

where $\tilde{T}^{(k)} = T^{(k)} + \tilde{\xi} I_T$ for $\tilde{\xi} > 0$,

$$T^{(k)} = \left[(P^{(k)})^T P^{(k)}\right] \circledast L_X + \operatorname{diag}\left\{1_I^T (A P^{(k)})^2\right\}, \tag{29}$$

$$b^{(k)} = 1_J^T \left(P^{(k)} \circledast (P^{(k)} + X^{(k)} L_X)\right). \tag{30}$$

The matrix $T^{(k)}$ in (28) is symmetric and at least nonnegative definite. Introducing a small positive constant $\tilde{\xi}$, the positive definiteness is enforced, which allows us to use the Cholesky factorization to solve the system (28). Using the Matlab's function $[R,p] = \texttt{chol}(\tilde{T}^{(k)})$, we can easily control the positive definiteness. If $p = 0$, the matrix $\tilde{T}^{(k)}$ is positive definite, and $R^T R = \tilde{T}^{(k)}$. When $p > 0$, the parameter $\tilde{\xi}$ should be set up to a positive value.

The solution $\eta_*^{(k)}$ must satisfy the box constraints: $0 < \eta_*^{(k)} \leq 1$. Hence, the update for η in the k-th iterative step is determined by $\eta^{(k)} = \max\{\varepsilon, \min\{1, \eta_*^{(k)}\}\}$, where $\eta_*^{(k)} = -R\backslash(R^T\backslash b^{(k)})$ for a small positive constant ε. The operator $\backslash$ denotes the back-substitution.

The final form of the modified SPG algorithm is given by Algorithm 1. It is a fundamental part of the NMF algorithm used in the training process (see Algorithm 2).

In the training process, we obtain the nonnegative matrices A and X. To classify the test sample $\tilde{y}$, first we need to project it onto the subspace spanned by the column vectors of the matrix A. As a result, we obtain $\tilde{x} \in \mathbb{R}_+^J$. This step can be carried out with the SPG, assuming $\alpha_X = 0$. Then, the following problem is solved: $t_* =$

Algorithm 1. SPG algorithm

```
Input  : $Y \in \mathbb{R}_+^{I\times T}$, $A \in \mathbb{R}_+^{I\times J}$, $X^{(0)} \in \mathbb{R}_+^{J\times T}$ - initial guess, $L_X \in \mathbb{R}^{T\times T}$, $\alpha_{min} > 0$,
         $\alpha_{max} > 0$, $\forall t : \bar{\alpha}_t^{(0)} = \frac{1}{2}\alpha_{max}$, $\tilde{\xi} = 10^{-12}$, $k = 0$,
Output: $\hat{X}$ - estimated factor matrices,
1  repeat
2      $k \leftarrow k+1$;
3      $G_X^{(k)} = \nabla_X \Psi(A, X^{(k)}) = A^T(AX^{(k)} - Y) + \alpha_X X^{(k)} L_X$ ;            // Gradient
4      $P^{(k)} = \left[X^{(k)} - G_X^{(k)} \operatorname{diag}\{(\bar{\alpha}_t^{(k)})^{-1}\}\right]_+ - X^{(k)}$ ;      // Descent direction
5      $[R, p] = \mathtt{chol}(T^{(k)})$ ;                      // where $T^{(k)}$ is given by (29)
6      while $p > 0$ do
7          $\tilde{\xi} \leftarrow 2\tilde{\xi}$;
8          $\tilde{T}^{(k)} = T^{(k)} + \tilde{\xi} I_T$;
9          $[R, p] = \mathtt{chol}(\tilde{T}^{(k)})$;
10     $\eta_*^{(k)} = -R\backslash(R^T\backslash b^{(k)})$;                   // where $b^{(k)}$ is given by (30)
11     $\bar{\eta}^{(k)} = \max\{\varepsilon, \min\{1, \eta_*^{(k)}\}\}$;                         // Steplengths
12     $X^{(k+1)} = X^{(k)} + P^{(k)} \operatorname{diag}\{\bar{\eta}^{(k)}\}$;
13     $\bar{\alpha}^{(k+1)} = \max\{\alpha_{min}, \min\{\alpha_{max}, \alpha^{(k+1)}\}\}$;      // where $\alpha^{(k+1)}$ is set to
       (21)
14 until Stop criterion is satisfied;
```

$\arg\min_{1\le t\le T} ||\tilde{x} - x_t||_2$, which gives us the index t_* of the class to which the sample $\tilde{y}$ is classified.

Remark 3. The complexity of one iterative step of the SPG algorithm for updating the matrix A is only $O(IJT)$. It increases considerably for updating the matrix X when $\alpha_X > 0$. In this case, it can be roughly estimated as $O(IJT) + O(J^2I) + O(J^2T) + O(JT^2) + O(T^3)$, assuming that the term $X^{(k)}L_X$ needs $O(JT^2)$, and the solution of the system (28) is obtained in $O(T^3)$. Using the Cholesky factorization, the computational cost for calculating $\tilde{T}^{(k)}$ can be diminished to $\frac{T^3}{6}$ elementary operations. Despite this computational cost predominates in the calculations, the overall complexity for the regularized SPG is still significantly lower than for the standard Newton method that needs $O(IJT) + O(J^3T^3) + O(JT)$ (Remark 2).

4 Experiments

In the experiments, the selected NMF algorithms are compared in the context of their usefulness for supervised classification of various images. The algorithms are evaluated with respect to the classification accuracy, normalized residual error and runtime.

Algorithm 2. SPG-NMF Algorithm

```
   Input  : $Y \in \mathbb{R}^{I \times T}$, $J$ - lower rank, $L_X \in \mathbb{R}^{T \times T}$ - weighting matrix, $\alpha_X$ - penalty
            parameter,
   Output: Factor matrices: $A \in \mathbb{R}_+^{I \times J}$ and $X \in \mathbb{R}_+^{J \times T}$
1  Initialize: $A$ and $X$ with nonnegative random numbers;
2  Replace negative entries (if any) in $Y$ with zero-value, $k = 0$ ;
3  repeat
4      $X^{(k+1)} = \mathtt{SPG}(Y, A^{(k)}, X^{(k)}, L_X, \alpha_X)$;
5      $\bar{d}_j^{(k+1)} = \sum_{t=1}^{T} x_{jt}^{(k+1)}$,
       $X^{(k+1)} \leftarrow \text{diag}\left\{\left(\bar{d}_j^{(k+1)}\right)^{-1}\right\} X^{(k+1)}$,   $A^{(k)} \leftarrow A^{(k)}\text{diag}\left\{\bar{d}_j^{(k+1)}\right\}$;
6      $\bar{A}^{(k+1)} = \mathtt{SPG}(Y^T, (X^{(k+1)})^T, (A^{(k)})^T)$;
7      $A^{(k+1)} = (\bar{A}^{(k+1)})^T$;
8      $\bar{\bar{d}}_j^{(k+1)} = \sum_{i=1}^{I} a_{ij}^{(k+1)}$,
       $X^{(k+1)} \leftarrow \text{diag}\left\{\bar{\bar{d}}_j^{(k+1)}\right\} X^{(k+1)}$,   $A^{(k+1)} \leftarrow A^{(k+1)}\text{diag}\left\{\left(\bar{\bar{d}}_j^{(k+1)}\right)^{-1}\right\}$;
9      $k \leftarrow k+1$;
10 until Stop criterion is satisfied;
```

The classification accuracy is statistically evaluated using n-fold Cross-Validation (CV). The mean-accuracy averaged over all CV-folds is expressed as the recognition rate. The normalized residual error in the k-iterative step is calculated as $r^{(k)} = \frac{||Y - A^{(k)}X^{(k)}||_F}{||Y||_F}$. The runtime is calculated in Matlab as the elapsed time of processing the algorithm until its stop criterion is satisfied.

We used the following data:

- *Dataset A*: It contains log-magnitude spectrograms created from the audio recordings of 6 musical instruments (cello, soprano saxophone, violin, bassoon, flute, and piano). The recordings are taken from the MIS database[1] of the University of Iowa. The sampling frequency is 44.1kHz. The audio samples are created from 4 sec. excerpts that contain meaningful information in the frequency range from 86Hz to 10.9kHz. Then, the spectrogram are downsampled to 64 frequencies $\times$ 128 time intervals. Each class is represented by 12 samples. The samples are divided into the training and testing sets according to the regular 6-fold CV rule.
- *Dataset B*: It is created from images of hand-written digits from 0 to 9. They were prepared by one student from Wroclaw University of Technology, and used in [34]. The images are downsampled to the resolution of 64×64 pixels. Each class contains 10 images. For testing the algorithms with this dataset, we used the regular 5-fold CV rule.

[1] `http://theremin.music.uiowa.edu`

Table 1 Mean recognition rates, standard deviations (in parenthesis), and elapsed time averaged over CV-folds for $J = 30$, and the datasets: A, B1 (dataset B without processing), B2 (dataset B with WT processing) and C.

Algorithm	A		B1		B2		C	
	Rec. rate	Time	Rec. rate	Time	Rec. rate	Time	Rec. rate	Time
MKL	78.6 (12.0)	0.71	70 (11.2)	0.4	96 (4.2)	0.79	69.75 (3.8)	3.95
MUE	91.7 (10.5)	6.19	90 (5.0)	2.18	96 (4.2)	1.86	94.5 (3.0)	14.45
ALS	90.3 (12.3)	1.05	21 (24.6)	1.82	96 (4.2)	0.81	92.5 (1.8)	2.68
LPG	94.4 (8.6)	8.94	91 (7.4)	3.13	93 (2.7)	2.99	95.75 (1.7)	13.44
FCNNLS	92.9 (8.2)	183.3	84 (6.5)	41.1	94 (2.2)	58.3	95.25 (1.6)	201.6
LNMF	94.4 (8.6)	0.71	25 (7.9)	0.39	95 (3.5)	1.32	91 (6.3)	2.49
DNMF	84.1 (5.5)	0.67	68 (2.7)	0.4	94 (4.2)	0.41	81 (4.2)	2.84
GNMF	93.3 (7.0)	4.51	90 (5.0)	2.07	**98** (2.7)	1.68	96 (2.1)	15.06
SPG	95.8 (7.0)	6.66	91 (9.6)	2.89	97 (2.7)	4.24	96.25 (1.8)	9.88
SPG-MD	95.8 (7.0)	14.8	87 (5.7)	2.05	97 (2.7)	2.59	94.75 (3.0)	13.28
SPG-DNMF	**97.2** (6.8)	11.76	**93** (7.6)	1.71	98 (2.7)	4.65	**97 (1.4)**	9.67

- *Dataset C*: This dataset is obtained from facial images taken from the ORL database[2]. It contains 400 frontal facial images of 40 people (10 pictures per person). The resolution of each image is 112×92 pixels. The 5-fold CV rule is used for testing the algorithms using this dataset.

Additionally, the tests on the dataset B are carried out for two cases: (B1) the original downsampled images are used (without preprocessing), (B2) the samples are preprocessed using the 2-D Wavelet Transform (WT) decomposition with the Haar wavelets at the first level. This task was achieved with the `wavedec2` function from Matlab.

We test the following NMF algorithms: MKL and MUE (standard multiplicative Lee-Seung algorithms for minimizing the Euclidean distance and KL divergence, respectively) [20], standard projected ALS [6], LPG (Lin's Projected Gradient) [23], regularized FCNNLS [35] (improved version of the l_2-norm regularized NMF algorithm proposed in [17]), LNMF [21], DNMF [33], GNMF [5], SPG (Algorithm 2 without the penalty terms), SPG-MD (regularized version with the penalty term determined by (14)), SPG-DNMF (the penalty term modeled by (10)).

Each NMF algorithm was extensively tested for various values of the related parameters. The results presented here are obtained for the parameters that give the highest performance. In the SPG, we set up: $\alpha_{min} = 10^{-8}$, $\alpha_{max} = 10^4$, $\varepsilon = 10^{-8}$. The parameters, such as α_X, γ and δ (DNMF), were tuned up individually to each dataset. For the GNMF, the matrix L_X is determined using the hard connection

[2] `http://people.cs.uchicago.edu/~dinoj/vis/orl/`

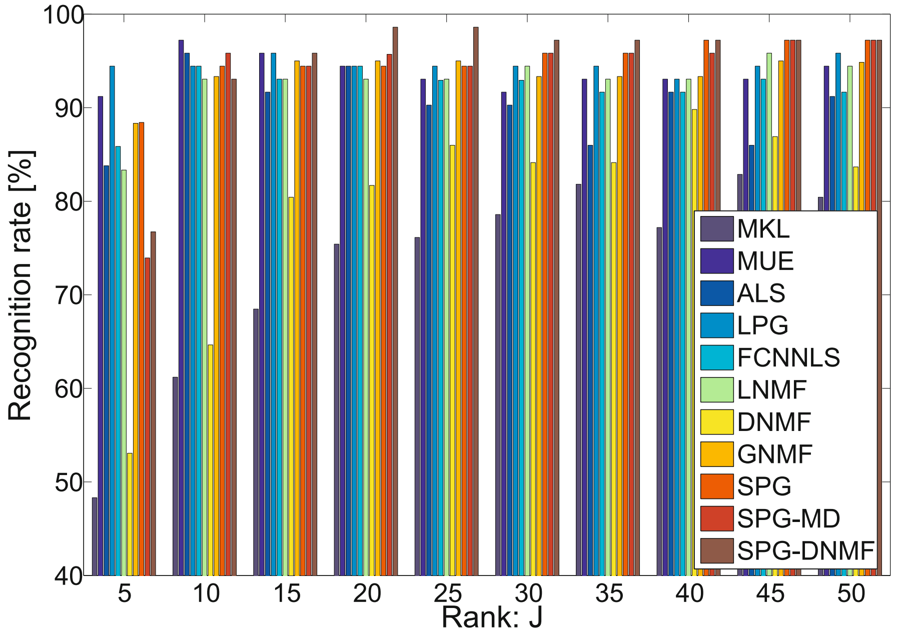


Fig. 1 Recognition rates obtained for dataset A using various NMF algorithms versus the rank J

criterion given by (11). For the SPG-MD, the Heat kernel weighting in (12) was used with $\sigma^2 = 10^7$, $k_1 = 5$ and $k_2 = 20$. For many test cases: $\alpha_X = 10^{-2}$ and $\xi = 10^2$. For the SPG-DNMF: $\alpha_X = 1$, $\gamma \approx 10^{-5}$ and $\delta \leq 10^{-6}$.

All the NMF algorithms were initialized by the SimplexMax algorithm (for $p = 1$) that was proposed in [36]. The stop criterion in all the tested algorithms was the same. The inner iterations (i.e. for updating one factor at the other fixed) were determined on the basis of the projected gradient criterion that was used in the LPG algorithm [23]. The maximum number of inner iterations was set to 10. The alternating steps (outer iterations) were terminated in all the tested algorithms if the normalized residual error drops below 10^{-4}.

The mean recognition rates versus the rank of factorization (J) are illustrated in Figs. 1–4 for the datasets A, B1, B2 and C, respectively. The same results but at the rank fixed to 30 are presented in Table 1. Additionally, the table contains the standard deviations of recognitions rates across CV-folds, and the runtime for processing each algorithm. Note that it is not the runtime of executing a given number of iterations but the elapsed time measured until the stop criterion holds.

The normalized residual errors versus the number of iterations for the selected NMF algorithms are plotted in Fig. 5.

The averaged classification results obtained with the MKL, LNMF and SPG-DNMF algorithms are also illustrated in Fig. 6 in the form of the Hinton graph of

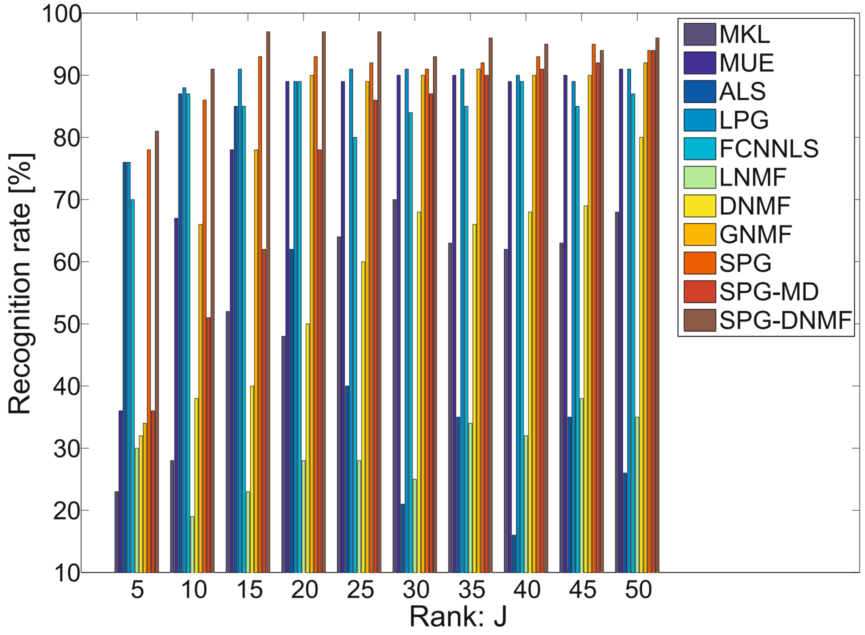


Fig. 2 Recognition rates obtained for dataset B without preprocessing using various NMF algorithms versus the rank *J*

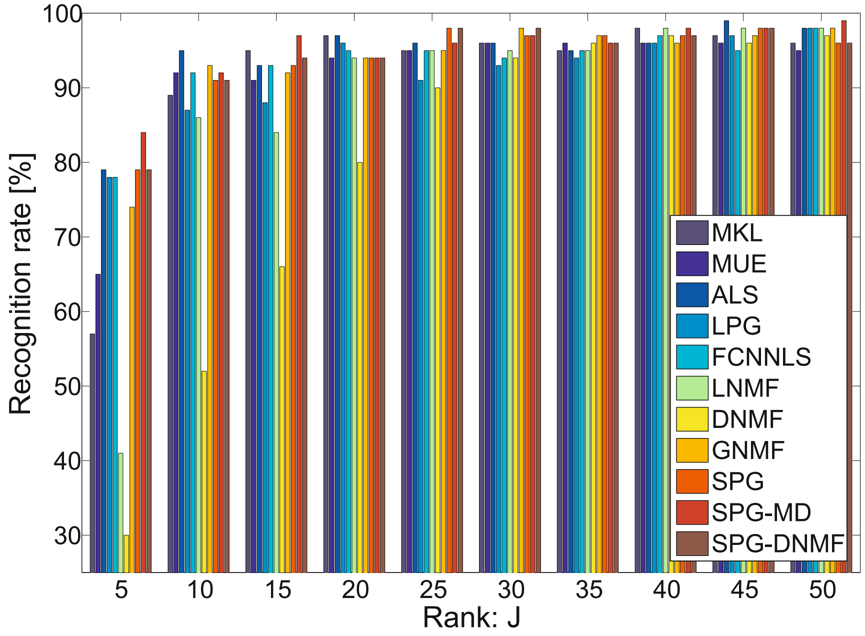


Fig. 3 Recognition rates obtained for dataset B with WT preprocessing using various NMF algorithms versus the rank *J*

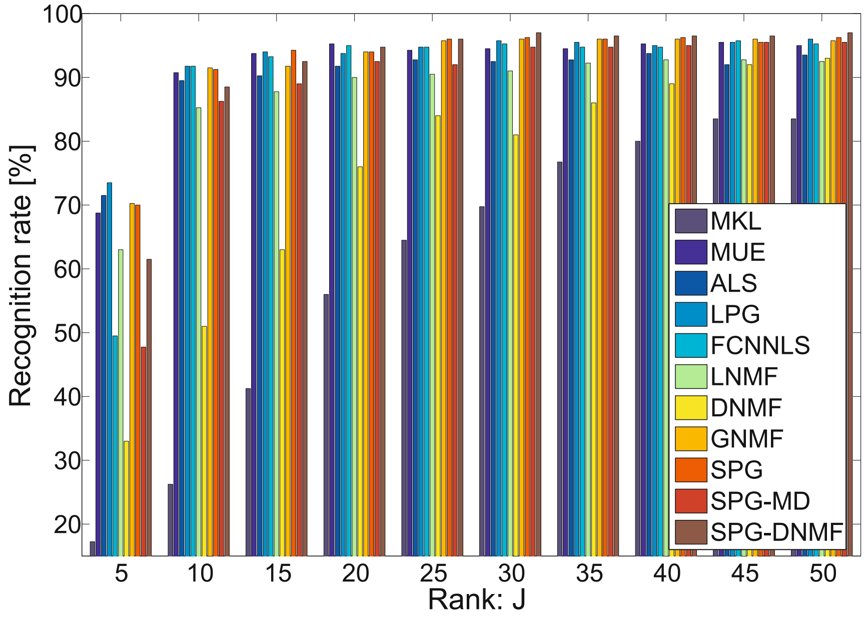


Fig. 4 Recognition rates obtained for dataset C using various NMF algorithms versus the rank *J*

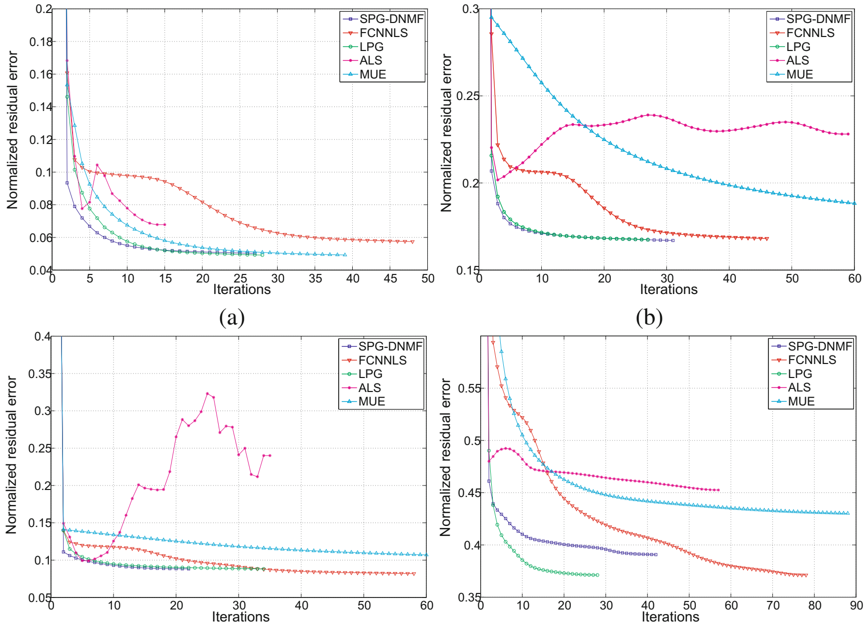


Fig. 5 Normalized residual errors versus alternating iterations: (a) dataset A; (b) dataset B without preprocessing; (c) dataset B with WT preprocessing; (d) dataset C

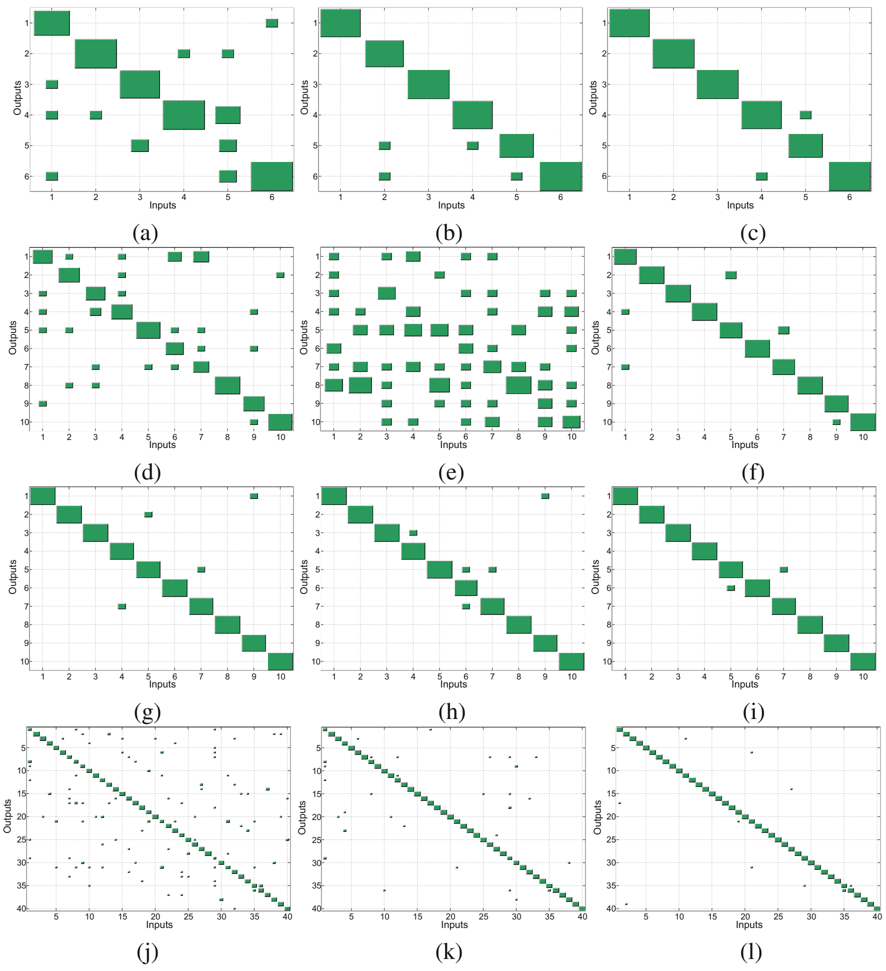


Fig. 6 Confusion matrices: (a) MKL, Dataset A; (b) LNMF: Dataset A; (c) SPG-DNMF, Dataset A; (d) MKL, Dataset B without preprocessing; (e) LNMF, Dataset B without preprocessing; (f) SPG-DNMF, Dataset B without preprocessing; (g) MKL, Dataset B with WT preprocessing; (h) LNMF, Dataset B with WT preprocessing; (i) SPG-DNMF, Dataset B with WT preprocessing; (j) MKL, Dataset C; (k) LNMF: Dataset C; (l) SPG-DNMF, Dataset C

confusion matrices. Both the Hinton graph and the confusion matrix are obtained with the Matlab functions from the *Neural Network Toolbox*.

5 Conclusions

In the chapter, we compared several NMF algorithms for various classification problems. We observed that the classification results are more affected by the dataset or the preprocessing than the NMF algorithm. For the dataset A, the SPG-DNMF outperforms the other algorithms for $J \geq 15$ (see Fig. 1). This algorithm behaves similarly for the dataset B1 (see Fig. 2), even for a lower rank ($J = 5$). After applying the preprocessing, nearly all the tested algorithms give the similar performance (see Fig. 3). Only the DNMF works noticeably worse for smaller ranks. For $J = 30$, the GNMF is slightly better than the SPG-DNMF (see Table 1). For classification of facial images, the SPG-DNMF is ranked first for $J > 20$. Usually an increase in the factorization rank leads to a higher recognition rate.

Comparing the results obtained for the datasets B1 and B2, one can easily notice that the WT preprocessing is crucial for this kind of images. For the other datasets, it does not lead to such a big difference.

The tests show that the Euclidean distance-based algorithms better classify the images in the datasets A, B1 and C than the KL divergence-based ones. Despite, the MKL and MUE have a similar convergence rate, the difference in performance is large. The LNMF and DNMF significantly outperform the MKL but are inferior to the SPG-DNMF.

Fig. 5 proves that the SPG-DNMF algorithm has a good convergence behavior. It converges monotonically and usually faster (especially in initial iterations) than the others. In each alternating step, the updates for A and X are optimal according to the first-order KKT optimality conditions. For example, the monotonic convergence is not observed for the ALS algorithm. Additionally, Fig. 5 shows the number of iterations performed to satisfy the stop criterion. For the threshold of the normalized residual error at the level of 10^{-4}, this number for the SPG-DNMF is usually lower than 50 iterations, while the MUE and FCNNLS require much more iterations. Hence, the runtime for MUE in Table 1 is sometimes longer than for the SPG-based algorithms. In this comparison, the FCNNLS is the slowest.

Summing up, the experiments demonstrate that the SPG-based algorithms work very efficiently in NMF-based classification of various images. It can be also easily extended to other applications, such as multi-linear discriminant analysis or multi-way array decompositions.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Advances in Neural Information Processing Systems 14, pp. 585–591. MIT Press (2001)
2. Benetos, E., Kotti, M., Kotropoulos, C.: Musical instrument classification using non-negative matrix factorization algorithms and subset feature selection. In: Proc. of 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006), Toulouse, France, p. V (2006)
3. Birgin, E.G., Martnez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM Journal on Control and Optimization 10, 1196–1211 (2000)
4. Cai, D., He, X., Han, J., Huang, T.: Graph regularized nonnegative matrix factorization for data representation. IEEE Transactions on Pattern Analysis and Machine Intelligence 33(8), 1548–1560 (2011)

5. Cai, D., He, X., Wu, X., Han, J.: Nonnegative matrix factorization on manifold. In: Proc. 8th IEEE International Conference on Data Mining (ICDM), pp. 63–72 (2008)
6. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.I.: Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation. Wiley and Sons (2009)
7. Cotter, S.F., Rao, B.D., Engan, K., Kreutz-Delgado, K.: Sparse solutions to linear inverse problems with multiple measurement vectors. IEEE Transaction on Signal Processing 53(7), 2477–2488 (2005)
8. Donoho, D., Stodden, V.: When does non-negative matrix factorization give a correct decomposition into parts? In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems (NIPS), vol. 16. MIT Press, Cambridge (2004)
9. Franc, V., Hlaváč, V., Navara, M.: Sequential coordinate-wise algorithm for the nonnegative least squares problem. In: Gagalowicz, A., Philips, W. (eds.) CAIP 2005. LNCS, vol. 3691, pp. 407–414. Springer, Heidelberg (2005)
10. Guan, N., Tao, D., Luo, Z., Yuan, B.: Manifold regularized discriminative nonnegative matrix factorization with fast gradient descent. IEEE Transactions on Image Processing 20(7), 2030–2048 (2011)
11. Guan, N., Tao, D., Luo, Z., Yuan, B.: NeNMF: An optimal gradient method for nonnegative matrix factorization. IEEE Transactions on Signal Processing 60(6), 2882–2898 (2012)
12. Guillamet, D., Vitria, J.: Classifying faces with nonnegative matrix factorization. In: Proc. 5th Catalan Conference for Artificial Intelligence, Castello de la Plana, Spain, pp. 24–31 (2002)
13. He, X., Niyogi, P.: Locality preserving projections. In: Advances in Neural Information Processing Systems 16. MIT Press (2003)
14. Hoyer, P.O.: Non-negative sparse coding. In: Neural Networks for Signal Processing XII (Proc. IEEE Workshop on Neural Networks for Signal Processing), Martigny, Switzerland, vol. 2, pp. 557–565 (2002)
15. Hoyer, P.O.: Non-negative matrix factorization with sparseness constraints. Journal of Machine Learning Research 5, 1457–1469 (2004)
16. Huang, K., Sidiropoulos, N.D., Swami, A.: Nonnegative matrix factorization revised: Uniquness and algorithm for symmetric decomposition, pp. 211–224 (2014)
17. Kim, H., Park, H.: Non-negative matrix factorization based on alternating non-negativity constrained least squares and active set method. SIAM Journal in Matrix Analysis and Applications 30(2), 713–730 (2008)
18. Kotsia, I., Zafeiriou, S., Pitas, I.: Discriminant non-negative matrix factorization and projected gradients for frontal face verification. In: Schouten, B., Juul, N.C., Drygajlo, A., Tistarelli, M. (eds.) BIOID 2008. LNCS, vol. 5372, pp. 82–90. Springer, Heidelberg (2008)
19. Lanteri, H., Theys, C., Richard, C.: Nonnegative matrix factorization with regularization and sparsity-enforcing terms. In: 4th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), pp. 97–100 (2011)
20. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature 401, 788–791 (1999)
21. Li, S.Z., Hou, X.W., Zhang, H.J., Cheng, Q.S.: Learning spatially localized, parts-based representation. In: Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), vol. 1, pp. I–207–I–212 (2001)
22. Lin, C.J.: On the convergence of multiplicative update algorithms for non-negative matrix factorization. IEEE Transactions on Neural Networks 18(6), 1589–1596 (2007)

23. Lin, C.J.: Projected gradient methods for non-negative matrix factorization. Neural Computation 19(10), 2756–2779 (2007)
24. Liu, X., Yan, S., Jin, H.: Projective nonnegative graph embedding. IEEE Transactions on Image Processing 19(5), 1126–1137 (2010)
25. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research. Springer, New York (1999)
26. Pascual-Montano, A., Carazo, J.M., Kochi, K., Lehmean, D., Pacual-Marqui, R.: Nonsmooth nonnegative matrix factorization (nsNMF). IEEE Transaction Pattern Analysis and Machine Intelligence 28(3), 403–415 (2006)
27. Phan, A.H., Cichocki, A.: Tensor decompositions for feature extraction and classification of high dimensional datasets. IEICE Nonlinear Theory and Its Applications 1(1), 37–68 (2010)
28. Phan, A.H., Cichocki, A.: Extended HALS algorithm for nonnegative Tucker decomposition and its applications for multiway analysis and classification. Neurocomputing 74(11), 1956–1969 (2011); Adaptive Incremental Learning in Neural Networks; Learning Algorithm and Mathematic Modelling. Selected paper from the International Conference on Neural Information Processing (ICONIP 2009)
29. Qin, L., Zheng, Q., Jiang, S., Huang, Q., Gao, W.: Unsupervised texture classification: Automatically discover and classify texture patterns. Image and Vision Computing 26(5), 647–656 (2008)
30. Wang, C., Song, Z., Yan, S., Lei, Z., Zhang, H.J.: Multiplicative nonnegative graph embedding. In: CVPR, pp. 389–396. IEEE (2009)
31. Wang, Y., Jia, Y., Hu, C., Turk, M.: Fisher nonnegative matrix factorization for learning local features. In: Proc. 6th Asian Conf. on Computer Vision, Jeju Island, Korea, pp. 27–30 (2004)
32. Yang, J., Yan, S., Fu, Y., Li, X., Huang, T.S.: Non-negative graph embedding. In: CVPR 2008, vol. 4, pp. 1–8 (2008)
33. Zafeiriou, S., Tefas, A., Buciu, I., Pitas, I.: Exploiting discriminant information in nonnegative matrix factorization with application to frontal face verification. IEEE Transactions on Neural Networks 17(3), 683–695 (2006)
34. Zak, M.: Recognition of hand-written digits with nonnegative matrix factorization. Master's thesis, Wroclaw University of Technology, Wroclaw (2011); Supervisor: R. Zdunek
35. Zdunek, R.: Regularized active set least squares algorithm for nonnegative matrix factorization in application to Raman spectra separation. In: Cabestany, J., Rojas, I., Joya, G. (eds.) IWANN 2011, Part II. LNCS, vol. 6692, pp. 492–499. Springer, Heidelberg (2011)
36. Zdunek, R.: Initialization of nonnegative matrix factorization with vertices of convex polytope. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2012, Part I. LNCS, vol. 7267, pp. 448–455. Springer, Heidelberg (2012)
37. Zdunek, R., Cichocki, A.: Nonnegative matrix factorization with constrained second-order optimization. Signal Processing 87, 1904–1916 (2007)
38. Zdunek, R., Cichocki, A.: Sequential coordinate-wise DNMF for face recognition. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part I. LNCS, vol. 6113, pp. 563–570. Springer, Heidelberg (2010)
39. Zdunek, R., Phan, A.-H., Cichocki, A.: GNMF with Newton-based methods. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 90–97. Springer, Heidelberg (2013)

Energy-Time Tradeoff in Recurrent Neural Nets

Jiří Šíma

Abstract. In this chapter, we deal with the energy complexity of perceptron networks which has been inspired by the fact that the activity of neurons in the brain is quite sparse (with only about 1% of neurons firing). This complexity measure has recently been introduced for feedforward architectures (i.e., threshold circuits). We shortly survey the tradeoff results which relate the energy to other complexity measures such as the size and depth of threshold circuits. We generalize the energy complexity for recurrent architectures which counts the number of simultaneously active neurons at any time instant of a computation. We present our energy-time tradeoff result for the recurrent neural nets which are known to be computationally as powerful as the finite automata. In particular, we show the main ideas of simulating any deterministic finite automaton by a low-energy optimal-size neural network. In addition, we present a lower bound on the energy of such a simulation (within a certain range of time overhead) which implies that the energy demands in a fixed-size network increase exponentially with the frequency of presenting the input bits.

1 Energy Complexity—Motivations and Survey

According to biological studies on energy consumption by cortical computation which are based on electrophysiological recordings [6], the energy cost of a single spike is substantially higher than that of the no-spike rest state of a neuron. On the other hand, the energy supply to the brain is known to be limited, which suffices possibly to fewer than 1%, the number of neurons that can be substantially active concurrently, and hence the activity of neurons in the cortex is quite sparse. This is not in contradiction with the fMRI observations [2] that the oxygen consumption by a functional part of the brain (e.g. visual cortex) when engaged in appropriate tasks

Jiří Šíma
Institute of Computer Science, Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic
e-mail: sima@cs.cas.cz

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_3

is not substantially higher as compared to the state when this region is not employed for its purpose. The reason is that the brain exhibit permanent, although sparse, activity, and thus the difference in energy consumption by a used vs. not used area of the brain is relatively small. In fact, this confirms the brain is quite effective in performing its tasks from the energy point of view.

In contrast to their biological counterparts, artificial neural circuits are frequently designed such that they do not take energy constraints into account. In fact, only the size (i.e., the number of gates) or the depth (i.e., the number of layers) of such networks are usually somehow optimized while computations have the property that, on average, approximately a half of units in the circuit fire (i.e. output a "1") during any computation. This fact has recently motivated the definition of a new complexity measure for *feedforward* perceptron networks (threshold circuits), the so-called *energy complexity* [17] which is the maximum number of units in the network which output 1, taken over all the possible inputs to the circuit.

Although the perceptron networks represent only a very rough abstract model of biological neural networks as compared to e.g. networks of spiking neurons, in a preliminary study of energy phenomenon, the stereotypical spikes can be simulated by outputs 1 from threshold gates, which makes the analysis technically more tractable. In addition, the first results have shown that the feedforward perceptron networks which are widely used in engineering applications have a surprisingly large computational power even if their energy complexity is restricted which means these circuits can exhibit only sparse activity. Minimizing the energy complexity requires a different approach to the circuit design which is potentially closer to the brain structures. In particular, different pathways in these circuits are activated for different clusters of inputs.

Furthermore, energy complexity of feedforward neural networks has been shown to be closely related by tradeoff results to other complexity measures such as the network size, the circuit depth, and the fan-in. In particular, a formula binding the energy e and the *size* s of any threshold circuit that computes a symmetric Boolean function f with n variables (i.e., the function value of f depends only on the number of 1's in the input) was derived [21], which gives e.g. the tradeoff $e \log s \geq \log n$ for the parity function f. Or even an exponential lower bound $\exp(\Omega(n^{1-o(n)}))$ on the number of perceptrons in any feedforward network with constant number of layers (i.e., constant depth) computing the Boolean inner product of $2n$ variables (see Eq. 6) was proven when the energy supply is restricted by $e = n^{o(1)}$ [19].

There is also a close relationship between the energy complexity e and the *depth* d (i.e., parallel computational time) of feedforward perceptron network: Any threshold circuit of energy complexity e and size s, computing a Boolean function f can be transformed to another threshold circuit of depth $2e+1$ and size $2es+1$, computing f [18]. This means that, within polynomial size, the energy complexity provides an upper bound on the depth of feedforward networks. Moreover, the tradeoff $e = O(n/\ell)$ between the energy e and the *fan-in* ℓ (i.e., the maximum number of inputs to a single unit) of any threshold circuit computing the modulus function MOD_m of n variables (i.e., MOD_m gives 0 iff the number of 1's in the input is divisible by m) was proven including an almost tight lower bound $e = \Omega((n-m)/\ell))$ [16].

Last but not least, energy complexity has found its important use in circuit complexity as a tool for proving the lower bounds [20].

2 Energy Complexity of Recurrent Networks—Chapter Outline

In this chapter, we study, for the first time, the energy complexity of *recurrent* neural (perceptron) networks which we define to be the maximum number of neurons outputting 1 at any time instant, taken over all possible computations. Clearly, this generalizes the energy complexity of threshold circuits (Sect. 1) to recurrent architectures as the energy of feedforward networks remains the same according to this definition.

It has been known for a long time that the computational power of binary-state recurrent networks corresponds to that of finite automata since the network of size s units can reach only a finite number (at most 2^s) different states [14]. A simple way of simulating a given deterministic finite automaton A with m states by a neural network N of size $O(m)$ is to implement each of the $2m$ transitions of A (having 0 and 1 transitions for each state) by a single unit in N which checks whether the input bit agrees with the respective type of transition [8]. Clearly, this simple linear-size implementation of finite automata requires only a constant energy since the determinism of automaton ensures that only a single acceptance path is traversed through the state transition graph, that is, only one neuron fires on this path at any time instant.

Much effort had been given to reducing the size of neural automata [1, 3, 4, 15] and, indeed, neural networks of size $\Theta(\sqrt{m})$ implementing a given deterministic finite automaton with m states were proposed and proven to be size-optimal [3, 4]. A natural question arises: What is the energy consumption when simulating finite automata by optimal-size neural networks? We answer this question in this chapter by showing the tradeoff between the energy and the time overhead of the simulation. In particular, we prove that an optimal-size neural network of $s = \Theta(\sqrt{m})$ units can be constructed to simulate a deterministic finite automaton with m states using the energy $O(e)$ for any function e such that $e = \Omega(\log s)$ and $e = O(s)$, while the time overhead for processing one input bit is $\tau = O(s/e)$. This means that the frequency of presenting the input bits can increase when more energy is supplied to the simulating network. For this purpose, we adapt the asymptotically optimal method of threshold circuit synthesis [7].

In addition, we also derive lower bounds on the energy consumption e of a neural network of size s simulating a finite automaton within the time overhead τ per one input bit, by using the technique due to Uchizawa and Takimoto [19] which is based on a communication complexity argument [5]. In particular, for less than sublogarithmic time overhead τ satisfying $\tau \log \tau = o(\log s)$, we obtain the lower bound $\log e = \Omega_\infty\left(\frac{1}{\tau}\log s\right)$ which implies $e \geq s^{c/\tau}$ for some constant $c > 0$ and for infinitely many s. Thus, the energy complexity in a fixed-size neural network increases exponentially with the frequency of presenting the input bits. For example, this means that for constant time overhead $\tau = O(1)$, the energy of any simulation meets $e \geq s^\delta$

for some constant δ such that $0 < \delta < 1$, and for infinitely many s, which can be compared to the energy $e = O(s)$ consumed by our simulation. For $\tau = O(\log^{\alpha} s)$ where $0 < \alpha < 1$, any such simulation requires energy $e = \Omega_{\infty}\left(s^{\log\log s/\log^{\delta} s}\right)$ for any $\delta > \alpha$, while $e = O\left(s/\log^{\alpha} s\right)$ is sufficient for our implementation.

This chapter is organized as follows. After a brief review of the basic definitions regarding neural networks as finite automata in Sect. 3, the main result concerning a low-energy simulation of finite automata by neural nets is presented in Sect. 4 including the basic ideas of the proof. The lower bounds on the energy consumption of such neural automata are formulated and compared to the respective upper bounds in Sect. 5. A concluding summary is given in Sect. 6. A preliminary version of this chapter has appeared as an extended abstract [12] which was further expanded to journal paper [13] including complete proofs. This chapter is focused on motivations and a survey, providing a brief exposition of the main ideas of our results on the energy complexity of recurrent neural networks while complicated technical details are only sketched or even omitted.

3 Neural Finite Automata

In order to precisely present our results we first recall the formal definition of an (artificial) *neural network* N and then we will introduce its I/O protocol for implementing a finite automaton. The network consists of s *units (neurons, threshold gates)*, indexed as $V = \{1,\ldots,s\}$, where s is called the network *size*. The units are connected into a directed graph representing the *architecture* of N, in which each edge (i,j) leading from unit i to j is labeled with an integer *weight* $w(i,j)$. The absence of a connection within the architecture corresponds to a zero weight between the respective neurons, and vice versa.

In contrast to general *recurrent* networks, which have cyclic architectures, the architecture of a *feedforward* network (or a so-called *threshold circuit*) is an acyclic graph. Hence, units in a feedforward network can be grouped in a unique minimal way into a sequence of $d+1$ pairwise disjoint *layers* $\alpha_0,\ldots,\alpha_d \subseteq V$ so that neurons in any layer α_t are connected only to neurons in subsequent layers α_u, $u > t$. Usually the zeroth, or *input layer* α_0 consists of external inputs and is not counted in the number of layers and in the network size. The last, or *output layer* α_d is composed of output neurons. The number of layers d excluding the input one is called the *depth* of threshold circuit.

The *computational dynamics* of (not necessarily feedforward) network N determines for each unit $j \in V$ its binary *state (output)* $y_j^{(t)} \in \{0,1\}$ at discrete time instants $t = 0,1,2,\ldots$. We say that neuron j is *active (fires)* at time t if $y_j^{(t)} = 1$, while j is *passive* for $y_j^{(t)} = 0$. This establishes the *network state* $\mathbf{y}^{(t)} = (y_1^{(t)},\ldots,y_s^{(t)}) \in \{0,1\}^s$ at each discrete time instant $t \geq 0$. At the beginning of a computation, the neural network N is placed in an *initial state* $\mathbf{y}^{(0)}$ which may also include an external input. At discrete time instant $t \geq 0$, an *excitation* of any neuron $j \in V$ is defined as

$$\xi_j^{(t)} = \sum_{i=1}^{s} w(i,j) y_i^{(t)} - h(j) \tag{1}$$

including an integer *threshold* $h(j)$ local to unit j. At the next instant $t+1$, the neurons $j \in \alpha_{t+1}$ from a selected subset $\alpha_{t+1} \subseteq V$ update their states $y_j^{(t+1)} = H(\xi_j^{(t)})$ in parallel by applying the *Heaviside function* $H : \Re \longrightarrow \{0,1\}$ which is defined as

$$H(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0\,. \end{cases} \tag{2}$$

The remaining units $j \in V \setminus \alpha_{t+1}$ do not change their outputs, that is $y_j^{(t+1)} = y_j^{(t)}$ for $j \notin \alpha_{t+1}$. In this way, the new network state $\mathbf{y}^{(t+1)}$ at time $t+1$ is determined.

Without loss of efficiency [9], we implicitly assume *synchronous* computations. Thus, the sets α_t which define the computational dynamics of N are predestined deterministically for each time instant t (e.g. $\alpha_t = V$ for any $t \geq 1$ means fully parallel synchronous updates). Note that computations in feedforward networks proceed layer by layer from the input layer up to the output one (i.e. sets α_t naturally coincide with layers), which implement Boolean functions. We define the *energy complexity* of N to be the maximum number of active units

$$\sum_{j=1}^{s} y_j^{(t)}$$

at any time instant $t \geq 0$, taken over all the computations of N.

The computational power of recurrent neural networks has been studied analogously to the traditional models of computations so that the networks are exploited as acceptors of formal languages $L \subseteq \{0,1\}^*$ over the binary alphabet. For the finite networks that are to recognize regular languages, the following I/O protocol has been used [1, 3, 4, 11, 15, 14]. A binary input word (string) $\mathbf{x} = x_1 \ldots x_n \in \{0,1\}^n$ of arbitrary length $n \geq 0$ is sequentially presented to the network bit by bit via an *input neuron* in $\in V$. The state of this unit is externally set (and clamped) to the respective input bits at prescribed time instants, regardless of any influence from the remaining neurons in the network, that is,

$$y_{\text{in}}^{(\tau(i-1))} = x_i \tag{3}$$

for $i = 1, \ldots, n$ where an integer parameter $\tau \geq 1$ is the *period* or *time overhead* for processing a single input bit. Then, an *output neuron* out $\in V$ signals at time τn whether the input word belongs to underlying language L, that is,

$$y_{\text{out}}^{(\tau n)} = \begin{cases} 1 & \text{for } \mathbf{x} \in L \\ 0 & \text{for } \mathbf{x} \notin L\,. \end{cases} \tag{4}$$

As usual, we will describe the limiting behavior (rate of growth) of functions when the argument tends towards infinity in terms of simpler functions by using Landau or big O notation. Recall that for functions $f \geq 1$ and $g \geq 1$ defined for all natural numbers, notations $g = O(f)$ and $g = \Omega(f)$ mean that for some real constant $c > 0$ and for all but finitely many natural numbers n, $g(n) \leq c \cdot f(n)$ and $g(n) \geq c \cdot f(n)$, respectively. In addition, $g = \Theta(f)$ if $g = O(f)$ and $g = \Omega(f)$ simultaneously. Similarly, $g = o(f)$ denotes that for *every* real constant $c > 0$ and for all but finitely many natural numbers n, $g(n) \leq c \cdot f(n)$, while $g = \Omega_\infty(f)$ means that for some real constant $c > 0$ and for infinitely many natural numbers n, $g(n) \geq c \cdot f(n)$. Clearly, $g = o(f)$ iff $\lim_{n\to\infty} g(n)/f(n) = 0$ iff $g \neq \Omega_\infty(f)$.

4 A Low-Energy Implementation of Finite Automata by Optimal-Size Neural Nets

Our main result concerning a low-energy implementation of finite automata by optimal-size neural nets is formulated in the following theorem. We will below provide a short informal proof sketch which explains the main ideas of the low-energy construction while the complicated details are deferred to technical paper [12].

Theorem 1. *A given deterministic finite automaton A with m states can be simulated by a neural network N of optimal size $s = \Theta(\sqrt{m})$ neurons with time overhead $\tau = O(s/e)$ per one input bit, using the energy $O(e)$, where e is any function satisfying $e = \Omega(\log s)$ and $e = O(s)$.*

Proof. (Sketch) For the construction of an optimal-size neural network N implementing a given deterministic finite automaton A we employ the approach due to Horne and Hush [3] which is based on Lupanov's result [7]. Unlike Horne and Hush who used the statement of Lupanov's theorem as a "black box", we need to modify Lupanov's construction in order to optimize the energy consumption.

Thus, a set Q of m states of a given deterministic finite automaton A is enumerated by integers $0, 1, \ldots, m-1$ so that each $q \in Q$ is encoded in binary using $p = \lceil \log_2 m \rceil + 1$ bits including one additional (e.g. the pth) bit which indicates the final states (i.e its value is 1 just for the final states of A). Then, the respective transition function $\delta : Q \times \{0,1\} \longrightarrow Q$ of automaton A, producing its new state $q_{\text{new}} = \delta(q_{\text{old}}, x) \in Q$ from the old state $q_{\text{old}} \in Q$ and current input bit $x \in \{0,1\}$, can be viewed as a vector Boolean function $\mathbf{f}_\delta : \{0,1\}^{p+1} \longrightarrow \{0,1\}^p$ in terms of binary encoding of automaton's states.

Furthermore, "transition" function $\mathbf{f}_\delta$ is implemented by a four-layer neural network C of asymptotically optimal size $\Theta(2^{p/2}) = \Theta(\sqrt{m})$ using the method of threshold circuit synthesis due to Lupanov [7]. Feedforward network C computing the transition function δ of A can then simply be transformed to a recurrent neural network N of the same size $s = \Theta(\sqrt{m})$ simulating A by adding the recurrent connections from the fourth layer to the first one, as it is schematically depicted in Fig. 1. In fact, the fourth output layer of C having p threshold gates is identified with the p units of the zeroth layer in N which store the current state of A while the remaining $(p+1)$th neuron in $\in V$ serves as an input to A (cf. Eq. 3). The recurrent connections ensure that the binary code of automaton's old state is replaced by the new one. In addition, the pth neuron in the zeroth layer of N representing the output

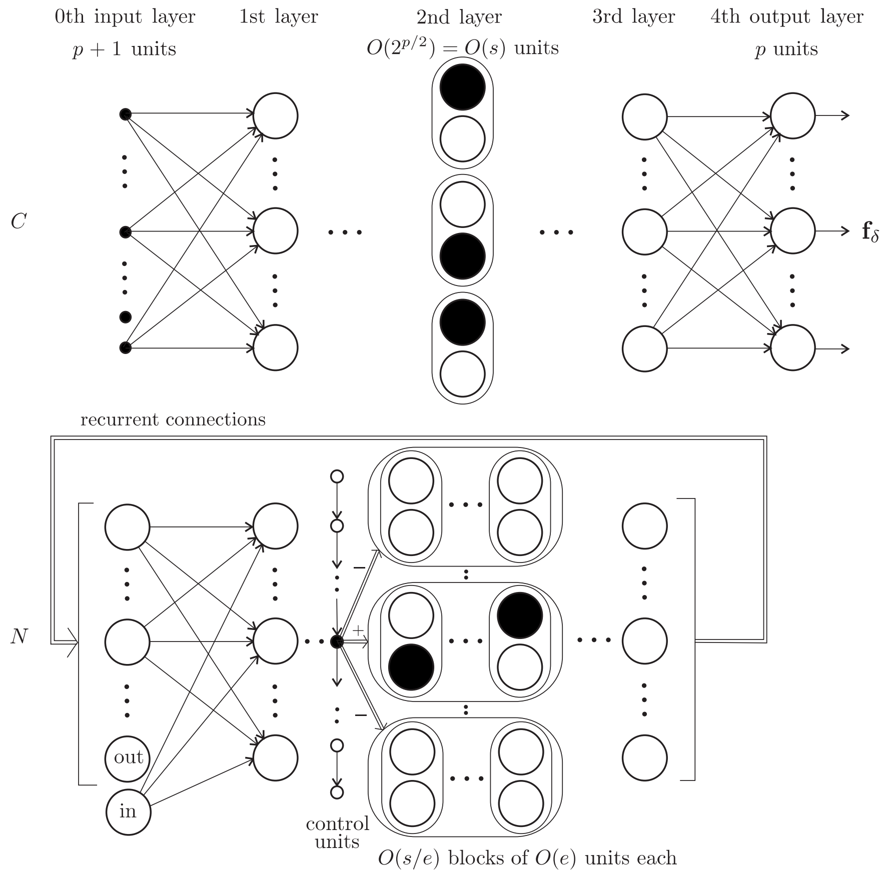


Fig. 1 A schema of the transformation of threshold circuit *C* (on top) implementing the "transition" function $\mathbf{f}_\delta$ of finite automaton *A* into a low-energy neural network *N* (at the bottom) simulating *A*.

neuron out $\in V$ signals whether the automaton is in an accepting state (cf. Eq. 4) because the *p*th bit in the binary code of states indicates the final states. Using this approach, a finite automaton can be implemented by an optimal-size neural net [3].

Unfortunately, the second layer of threshold circuit *C* in Lupanov's construction [7] contains $\Theta(s)$ gates, half of which fire for any input to *C*, which results in an unacceptably high energy consumption $\Omega(s)$ although the energy demands of the remaining three layers are bounded by $O(p) = O(\log s)$. In particular, this second layer is composed of $O(2^{p/2})$ pairs of units such that exactly one neuron of each pair fires, except for just $2p$ pairs of which both their neurons are active simultaneously. In fact, determining these $2p$ pairs is the core of evaluating the function $\mathbf{f}_\delta$ for a given input.

In order to achieve a low-energy implementation of *A*, this layer of $\Theta(s)$ neurons is properly partitioned into $O(s/e)$ blocks of $O(e)$ units each so that no pair of units is split into two blocks (see Fig. 1). Then, so-called *control units*, one for each block, are introduced which ensure that these blocks are updated successively one by one so that the energy consumption (i.e. the maximum number of simultaneously active neurons) is bounded by $O(e)$, while the time overhead for processing a single input bit possibly increases to $O(s/e)$. In particular, the control units create typically a path along which the control signal is propagated so that only one control unit on the path is active at each time instant. Each such a control unit releases an update of one associated block while the remaining blocks are blocked typically at the zero state (consuming no energy) apart from those $2p$ pairs of which both neurons fire simultaneously. In this way, the energy consumption of C is reduced from $\Omega(s)$ to $O(e+p) = O(e+\log s) = O(e)$ as $e = \Omega(\log s)$. □

Theorem 1 provides an energy-time tradeoff for the size-optimal recurrent networks implementing finite automata, which generalizes the tradeoff results for threshold circuits (see Sect. 1). In particular, the time overhead τ which is necessary for processing a single input bit decreases when more energy e is supplied to the network. For the full energy $e = \Omega(s)$ we obtain the constant-time overhead.

5 The Energy Lower Bound

In this section, we will show lower bounds on the energy complexity of neural networks implementing finite automata. For this purpose, we will employ the technique due to Uchizawa and Takimoto [19] which is based on communication complexity [5]. Assume that $f : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$ is a Boolean function whose value $f(\mathbf{x},\mathbf{y})$ has to be computed by two players with unlimited computational power, each receiving only his/her part of the input $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$, respectively, while they wish to exchange with each other the least possible number of bits. In particular, they communicate according to a randomized protocol additionally making use of the same public random bit string. For any error probability ε satisfying $0 \leq \varepsilon < 1/2$, the *communication complexity* $R_\varepsilon(f)$ of function f is defined to be the maximum number of bits needed to be exchanged for the best randomized protocol to make the two players compute the correct value of $f(\mathbf{x},\mathbf{y})$ with probability at least $1-\varepsilon$, for every input assignment $\mathbf{x}$ and $\mathbf{y}$.

It is well known [5] that almost all Boolean functions f of $2n$ variables have large communication complexity

$$R_\varepsilon(f) = \Omega\left(n + \log\left(\frac{1}{2} - \varepsilon\right)\right) \tag{5}$$

for any error probability ε such that $0 \leq \varepsilon < 1/2$. An example of a particular function that meets condition (5) is the Boolean inner product $IP_n : \{0,1\}^{2n} \longrightarrow \{0,1\}$, defined as

$$IP_n(x_1,\ldots,x_n,y_1,\ldots,y_n,) = \bigoplus_{i=1}^{n}(x_i \wedge y_i) \tag{6}$$

where $\oplus$ denotes the parity which gives 1 iff the number of satisfied conjunctions $x_i \wedge y_i$ (i.e. $x_i = y_i = 1$), for $i = 1, \ldots, n$, is odd.

On the other hand, Uchizawa and Takimoto [19] proved the upper bound on the communication complexity of Boolean function f in terms of the size, depth, and energy complexity of a feedforward network computing f:

Theorem 2 ([19]). *If a Boolean function $f : \{0,1\}^{2n} \longrightarrow \{0,1\}$ can be computed by a threshold circuit of size S, depth d, and energy complexity E, then*

$$R_\varepsilon(f) = O\Big((E+d)(\log n + (E+1)^d \log S\Big) \tag{7}$$

for error probability

$$\varepsilon = \frac{1}{2} - \frac{1}{4S^{3(E+1)^d}} \, . \tag{8}$$

The lower and upper bounds on the communication complexity (5) and (7), respectively, are put together in the following lemma:

Lemma 1 ([13]). *Let $f : \{0,1\}^{2n} \longrightarrow \{0,1\}$ be a Boolean function of $2n$ variables whose communication complexity satisfies condition (5), which can be computed by a threshold circuit of size S, depth d, and energy complexity E such that $n = O(S)$ and $d = O(E)$. Then $n = O(E^{d+1} \log S)$.*

Now we will formulate the result providing the lower bound $e \geq s^{c/\tau}$ (for some constant $c > 0$ and for infinitely many s) on the energy complexity e of a recurrent neural network of size s neurons implementing a given finite automaton with time overhead τ such that $\tau^\tau = o(s)$. This means the lower bound is valid for less than sublogarithmic time overheads.

Theorem 3. *Let $\tau \log \tau = o(\log s)$. There exists a neural network of size s neurons simulating a finite automaton with time overhead τ per one input bit which needs energy e such that $\log e = \Omega_\infty\left(\frac{1}{\tau} \log s\right)$.*

Proof. Let N be a neural network of size s neurons simulating a finite automaton A with time overhead τ per one input bit. The states of A are represented by the 2^{s-1} states of N (excluding the input neuron in) and the transition function of A is computed by N within τ time steps. Clearly, network N can be "unwound" into a threshold circuit C of depth $d = \tau$ and size $S = \tau s$ which implements the transition function of A so that each layer is a copy of N [10]. Thus, the states of neurons in the ith layer of C coincide with the network state $\mathbf{y}^{(k\tau+i)}$ for $0 \leq i \leq \tau$, when the new state $\mathbf{y}^{((k+1)\tau)}$ of A is produced from the old one $\mathbf{y}^{(k\tau)}$ including the current input bit. Hence, the energy complexity of C is a τ multiple of the energy consumed by N, that is, $E = \tau e$.

As component $f_k : \{0,1\}^s \longrightarrow \{0,1\}$ (for $1 \leq k \leq s$) of the transition function defining A can be arbitrary, there is a neural network N simulating A such that f_k implemented by C has large communication complexity satisfying condition (5). Moreover, $n = \lceil s/2 \rceil = O(S)$ and $d = \tau = O(E)$, which meets the remaining assumptions of Lemma 1. It follows that

$$\lceil s/2 \rceil = n = O\left(E^{d+1} \log S\right) = O\left((\tau e)^{t+1} \log \tau s\right) \tag{9}$$

according to Lemma 1. On the contrary, suppose that

$$\log e = o\left(\frac{\log s}{\tau}\right). \tag{10}$$

We will prove that $(\tau e)^{\tau+1} \log \tau s = o(s)$ which contradicts equation (9). For this purpose, it suffices to show that $\log((\tau e)^{\tau+1} \log \tau s) = o(\log s)$. This can be rewritten as $(\tau+1)\log\tau + (\tau+1)\log e + \log\log\tau + \log\log s = o(\log s)$ which follows from the assumption of the theorem and equation (10), completing the argument. □

Theorem 3 provides an energy-time tradeoff in recurrent neural networks from the lower-bound perspective as a counterpart to Theorem 1. In particular, the underlying formula relates the energy demands to the time overhead for processing a single input bit. It follows that the energy complexity in a fixed-size neural network increases exponentially with the frequency of presenting the input bits. In the following corollary we will formulate the lower bounds on energy complexity in terms of the network size for selected cases of sublogarithmic time overhead.

Corollary 1.
1. If $\tau = O(1)$, then $e \geq s^{\delta}$ for some δ such that $0 < \delta < 1$ and for infinitely many s.
2. If $\tau = O(\log\log s)$, then $e = \Omega_\infty\left(s^{1/\log^{\delta} s}\right) = \Omega_\infty\left(2^{\log^{1-\delta} s}\right)$ for any δ such that $0 < \delta < 1$.
3. If $\tau = O(\log^{\alpha} s)$ for some $0 < \alpha < 1$, then $e = \Omega_\infty\left(s^{\log\log s/\log^{\delta} s}\right) = \Omega_\infty\left((\log s)^{\log^{1-\delta} s}\right)$ for any δ such that $\delta > \alpha$.

Proof. 1. For $\tau = O(1)$, the assumption $\tau \log \tau = o(\log s)$ trivially holds and the proposition follows straightforwardly from Theorem 3.
2. For $\tau = O(\log\log s)$, there is $c_u > 0$ such that for all but finitely many s we have $\tau\log\tau \leq c_u(\log\log s)\log\log\log s + (c_u \log c_u)\log\log s = o(\log s)$. According to Theorem 3, there is $c_\ell > 0$ such that $\log e \geq \frac{c_\ell \log s}{c_u \log\log s}$ for infinitely many s. On the contrary, suppose that $e = o\left(s^{1/\log^{\delta} s}\right)$ for some δ satisfying $0 < \delta < 1$, which implies $\log^{1-\delta} s \geq \frac{c_\ell \log s}{c_u \log\log s}$ leading to a contradiction $\frac{\log\log s}{\log^{\delta} s} \geq \frac{c_\ell}{c_u} > 0$.
3. If $\tau = O(\log^{\alpha} s)$ for some $0 < \alpha < 1$, then there is $c_u > 0$ such that for all but finitely many s we have $\tau\log\tau \leq c_u(\log^{\alpha} s)\log\log^{\alpha} s + (c_u\log c_u)\log^{\alpha} s = o(\log s)$. According to Theorem 3, there is $c_\ell > 0$ such that $\log e \geq \frac{c_u}{c_\ell}\log^{1-\alpha} s$ for infinitely many s. On the contrary, suppose that $e = o\left(s^{\log\log s/\log^{\delta} s}\right)$ for some δ satisfying $\delta > \alpha$, which implies $(\log\log s)\log^{1-\delta} s \geq \frac{c_u}{c_\ell}\log^{1-\alpha} s$ leading to a contradiction $\frac{\log\log s}{\log^{\delta-\alpha} s} \geq \frac{c_u}{c_\ell} > 0$. □

We can compare the lower bounds on energy complexity of simulating the finite automata by neural nets presented in Corollary 1 to the respective upper bounds provided by Theorem 1. For the constant time overhead $\tau = O(1)$, the construction from Theorem 1 achieves the energy consumption of $e = O(s)$, while any simulation requires energy $e \geq s^{\delta}$ for some constant δ such that $0 < \delta < 1$ and for infinitely many s, according to Corollary 1. Similarly, for the time overhead of $\tau = O(\log^{\alpha} s)$ where $0 < \alpha < 1$, we have the upper bound of $e = O(s/\log^{\alpha} s)$ which compares to the lower bound of $e = \Omega_{\infty}\left(s^{\log\log s/\log^{\delta} s}\right)$. Clearly, there are still gaps between these lower and upper bounds, respectively, which need to be eliminated.

6 Conclusions

We have, for the first time, applied the energy complexity measure to recurrent neural nets. This measure has recently been introduced and studied for feedforward perceptron networks. The binary-state recurrent neural networks recognize exactly the regular languages so we have investigated their energy consumption of simulating the finite automata with the asymptotically optimal number of neurons. We have presented a low-energy implementation of finite automata by optimal-size neural nets with the tradeoff between the time overhead for processing one input bit and the energy varying from the logarithm to the full network size. It appears that the frequency of presenting the input bits can only increase when more energy is supplied to the network. We have also achieved lower bounds for the energy consumption of neural finite automata which are valid for less than sublogarithmic time overheads and are still not tight. It follows that the energy demands in a fixed-size network increase exponentially with the frequency of presenting the input bits. An open problem remains for further research whether these bounds can be improved. In addition, we have so far assumed the worst case energy consumption while the average case analysis would be another challenge.

Acknowledgements. Author's research was done with institutional support RVO: 67985807 and partially supported by the grant of the Czech Science Foundation No. P202/12/G061.

References

1. Alon, N., Dewdney, A.K., Ott, T.J.: Efficient simulation of finite automata by neural nets. J. ACM 14(2), 495–514 (1991)
2. Gafaniz, R., Sanches, J.M.: Neuronal electrical activity, energy consumption and mitochondrial ATP restoration dynamics: A physiological based model for FMRI. In: Proceedings of the ISBI 2011 Eighth IEEE International Symposium on Biomedical Imaging: From Nano to Macro, pp. 341–344. IEEE (2011)
3. Horne, B.G., Hush, D.R.: Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Netw. 9(2), 243–252 (1996)
4. Indyk, P.: Optimal simulation of automata by neural nets. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 337–348. Springer, Heidelberg (1995)
5. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)

6. Lennie, P.: The cost of cortical computation. Curr. Biol. 13(6), 493–497 (2003)
7. Lupanov, O.: On the synthesis of threshold circuits. Probl. Kibern. 26, 109–140 (1973)
8. Minsky, M.: Computations: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
9. Orponen, P.: Computing with truly asynchronous threshold logic networks. Theor. Comput. Sci. 174(1-2), 123–136 (1997)
10. Savage, J.E.: Computational work and time on finite machines. J. ACM 19(4), 660–674 (1972)
11. Siegelmann, H.T., Sontag, E.D.: Computational power of neural networks. J. Comput. Syst. Sci. 50(1), 132–150 (1995)
12. Šíma, J.: A low-energy implementation of finite automata by optimal-size neural nets. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 114–121. Springer, Heidelberg (2013)
13. Šíma, J.: Energy complexity of recurrent neural networks. Neural Comput. 26(5) (2014)
14. Šíma, J., Orponen, P.: General-purpose computation with neural networks: A survey of complexity theoretic results. Neural Comput. 15(12), 2727–2778 (2003)
15. Šíma, J., Wiedermann, J.: Theory of neuromata. J. ACM 45(1), 155–178 (1998)
16. Suzuki, A., Uchizawa, K., Zhou, X.: Energy and fan-in of threshold circuits computing Mod functions. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 154–163. Springer, Heidelberg (2011)
17. Uchizawa, K., Douglas, R., Maass, W.: On the computational power of threshold circuits with sparse activity. Neural Comput. 18(12), 2994–3008 (2006)
18. Uchizawa, K., Nishizeki, T., Takimoto, E.: Energy and depth of threshold circuits. Theor. Comput. Sci. 411(44-46), 3938–3946 (2010)
19. Uchizawa, K., Takimoto, E.: Exponential lower bounds on the size of constant-depth threshold circuits with small energy complexity. Theor. Comput. Sci. 407(1-3), 474–487 (2008)
20. Uchizawa, K., Takimoto, E.: Lower bounds for linear decision trees via an energy complexity argument. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 568–579. Springer, Heidelberg (2011)
21. Uchizawa, K., Takimoto, E., Nishizeki, T.: Size-energy tradeoffs for unate circuits computing symmetric Boolean functions. Theor. Comput. Sci. 412(8-10), 773–782 (2011)

An Introduction to Delay-Coupled Reservoir Computing

Johannes Schumacher, Hazem Toutounji, and Gordon Pipa

Abstract. Reservoir computing has been successfully applied in difficult time series prediction tasks by injecting an input signal into a spatially extended reservoir of nonlinear subunits to perform history-dependent nonlinear computation. Recently, the network was replaced by a single nonlinear node, delay-coupled to itself. Instead of a spatial topology, subunits are arrayed in time along one delay span of the system. As a result, the reservoir exists only implicitly in a single delay differential equation, the numerical solving of which is costly. We give here a brief introduction to the general topic of delay-coupled reservoir computing and derive approximate analytical equations for the reservoir by solving the underlying system explicitly. The analytical approximation represents the system accurately and yields comparable performance in reservoir benchmark tasks, while reducing computational costs practically by several orders of magnitude. This has important implications with respect to electronic realizations of the reservoir and opens up new possibilities for optimization and theoretical investigation.

1 Introduction to Reservoir Computation

Predicting future behavior and learning temporal dependencies in time series of complex natural systems remains a major goal in many disciplines. In Reservoir Computing, the issue is tackled by projecting input time series into a recurrent network of nonlinear subunits [13, 19]: Recurrency provides memory of past inputs, while the large number of nonlinear subunits expand their informational features. History-dependent nonlinear computations are then achieved by simple linear readouts of the network activity.

In a recent advancement, the recurrent network was replaced by a single nonlinear node, delay-coupled to itself [2]. Such a setup is formalized by a delay differential equation which can be interpreted as an *infinite-dimensional* dynamical system.

Johannes Schumacher · Hazem Toutounji · Gordon Pipa
Institute of Cognitive Science, University of Osnabrück, Albrechtstr. 28,
49069 Osnabrück, Germany
e-mail: {joschuma,htoutounji,gpipa}@uos.de

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_4

Whereas classical reservoirs have an explicit spatial representation, a delay-coupled reservoir (DCR) uses temporally extended sampling points across the span of its delayed feedback, termed *virtual nodes*. The main advantage of such a setup is that it allows for easy realization in optical and electronic hardware [26] which has great potential for industrial application.

A drawback of this approach is the fact that the actual reservoir is always only implicit in a single delay differential equation. Consequently, in many implementations the underlying system has to be solved numerically. This leads to a computational bottleneck and creates practical limitations for reservoir size and utility. The lack of reservoir equations also presents problems for applying optimization procedures.

To overcome this, we present a recursive analytical solution used to derive approximate virtual node equations. The solution is assessed in its computational capabilities as a DCR, and compared against numerical solvers in nonlinear benchmark tasks. While computational performance is comparable, the analytical approximation leads to considerable savings in computation time, allowing the exploration of exceedingly large setups. The latter allows us to stir the system away from toy examples to the realm of application.

Moreover, we provide in this chapter a general introduction to the topic of delay-coupled reservoir computing to familiarize the reader with the concept, and to give some practical guidelines for setting up a DCR. To this end, we first discuss some theory regarding solvability and dynamics of the delay differential equation underlying a DCR. In a next step, we use this insight to derive the approximate analytical equations for the reservoir and explore their accuracy. A computationally efficient numerical solution scheme arises that allows the investigation of large reservoirs, the performance of which is evaluated on classical benchmark tasks. These will also serve to illustrate a practical implementation. Finally, we present an application to an experimental recording of a far-infrared laser operating in a chaotic regime and show how a DCR can be embedded into Gaussian process regression to deal with uncertainty and to be able to optimize hyperparameters.

2 Single Node Delay-Coupled Reservoirs

In this section, we discuss the theory of simple retarded functional differential equations, of which the delay differential equations underlying a DCR are a particular instance, and derive approximate expressions for the virtual nodes. These will serve as the basis for a practical implementation in later parts of this chapter.

2.1 Computation via Delayed Feedback

In a DCR, past and present information of a covariate time series undergo nonlinear mixing via injection into a dynamically evolving “node” with delayed feedback. Formally, these dynamics can be modeled by a delay differential equation of the type

$$\frac{dx(t)}{dt} = -x(t) + f(x(t-\tau), J(t)) \in \mathbb{R}, \tag{1}$$

where τ is the delay time, $J(t)$ is a weighted and temporally multiplexed transformation of some input signal $u(t)$ driving the system, and f is a sufficiently smooth real-valued nonlinear function. The nonlinearity is necessary to provide a rich feature expansion and separability of the information present in the input time series. Although a solution to system (1) will in general not be obtainable analytically, it can often be fully optically or electronically realized, for example in an all-optical laser system with nonlinear interference given by its own delayed feedback [16].

The Mackey-Glass system represents a possible choice of nonlinearity which also admits a hardware implementation of the system. The corresponding differential equation is given by

$$\frac{dx(t)}{dt} = -x(t) + \frac{\eta(x(t-\tau) + \gamma J(t))}{1 + (x(t-\tau) + \gamma J(t))^p} \in \mathbb{R}. \tag{2}$$

The parameters $\gamma, \eta, p \in \mathbb{R}$ determine in which dynamical regime the system operates. Although the Mackey-Glass system can exhibit even chaotic dynamics for $p > 9$, a fixed point regime appears to have the most suitable properties with respect to memory capacity of the resulting reservoir. In a chaotic regime, the system would have an excellent separability of input features, due to its sensitive dependence on initial conditions. However, presumably as a result of the intrinsic entropy production and exponential decay of auto-correlation in strange attractors, the chaotic system essentially lacks memory capacity with respect to the input signal. Furthermore, the required precision for computing the trajectories in a numerical solver would result in prohibitive computational costs with increasing chaoticity of the system. Other choices of nonlinearities are possible and have been investigated [2]. For purposes of illustration, we will use the Mackey-Glass system throughout this chapter.

Injecting a signal $u(t)$ into the reservoir is achieved by multiplexing it in time: The DCR receives a single constant input $u(\bar{t}) \in \mathbb{R}$ in each reservoir time step $\bar{t} = \lceil \frac{t}{\tau} \rceil$, corresponding to one τ-cycle of the system. For example, $(i-1)\tau \leq t \leq i\tau$ denotes the ith τ-cycle and is considered to be a single reservoir time step during which $u(t) = u_i = const.$ This scheme is easily extendable to vector-valued input signals. The dynamics of x during a single delay span τ are to be seen as the temporally extended analogon of an artificial neural network, where the nonlinear subunits are arrayed not in space, but in time. Since one is interested in operating $x(t)$ in a fixed point regime, the system would at this point simply saturate and converge in the course of a τ-cycle during which u_i is constant, e.g. $\lim_{t\to\infty} x(t) = 0$ for $u_i = 0$ and suitable initial conditions (see section 2.2). To add perturbation and create a rich *feature expansion* of the input signal in time (analogous to neural network activity in space), the delay line τ is shattered into N subintervals of length θ_j, for $j = 1, ..., N$. On these subintervals an additional mask function reweights the otherwise constant input value u_i, such that the saturating system x is frequently perturbed and prevented from converging. That is, the mask is a real-valued function on an interval of length τ, e.g. $[-\tau, 0]$, which is piecewise constant:

$$m(t) = m_j \in \mathbb{R} \qquad \text{for} \quad \theta_{j-1} < t \leq \theta_j, \quad \sum_{j=1}^{N} \theta_j = \tau. \tag{3}$$

The input to the reservoir $x(t)$ during the ith τ-cycle is thus given by $J(t) = m(t)u_i$ (compare eq. (1)).

In the original approach, the m_j were simply random samples from $\{-1, 1\}$, meant to perturb the system and to create transient trajectories. The impact of certain binary sequences on computation, as well as multi-valued masks, have also been studied and may lead to a task-specific improvement of performance. General principles for the choice of optimal $m(t)$ are, however, not yet well understood, mainly due to problems in solving the system efficiently and due to an inconvenient dependence on the θ_j. For optimal information processing, θ_j has to be short enough to prevent convergence of the system trajectory (so as to retain the influence of past states), but long enough for the system to act upon the masked input signal and expand information. In the case of equidistant $\theta_j = \theta$, it can be determined experimentally that choosing θ to be one fifth of the system's intrinsic time scale yields

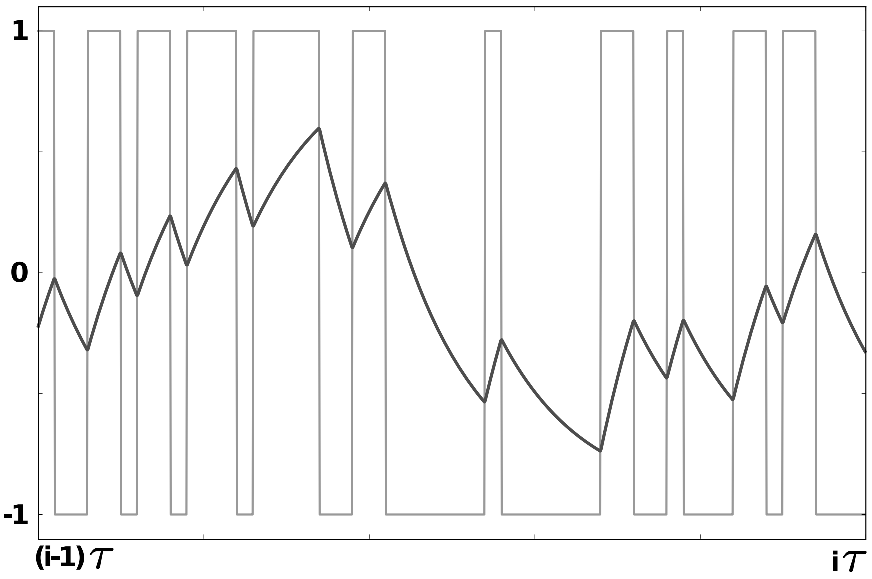


Fig. 1 Exemplary trajectory (dark gray) of system (2) during one τ-cycle of length 10, with equidistant $\theta_j = 0.2$ for $j = 1, ..., 50$, $\eta = 0.04$, $\gamma = 0.005$, $p = 7$, while $u_i = 0.385$ constant during the entire τ-cycle. In light gray, the piecewise constant mask (eq. (3)) with $m_j \in \{-1, 1\}$ for $j = 1, ..., 50$ is shown. The system is in a fixed point regime (parametrized by $J(t)$) and starts converging on intervals where $J(t) = u_i m(t)$ is constant.

good computational performance in benchmark tasks [2]. In system (1) the intrinsic timescale is the multiplicative factor of the derivative on the left-hand side, which is 1, so that $\theta = 0.2$ accordingly. Figure (1) illustrates how the resulting dynamics of $x(t)$ may look like during a single τ-cycle, using a short delay span $\tau = 10$, shattered into 50 subintervals.

To obtain a statistical model, a sample is read out at the end of each θ_j, thus yielding N predictor variables at each τ-cycle (i.e. reservoir time step $\bar{t} = \lceil \frac{t}{\tau} \rceil$). These are termed "*virtual nodes*" in analogy to the spatially extended nodes of a neural network. During the ith τ-cycle, virtual node j is sampled as

$$x_j(u_i) := x((i-1)\tau + \sum_{k=1}^{j} \theta_k)$$

and used in a linear functional mapping

$$\hat{y}_i = \sum_{j=1}^{N} \alpha_j x_j(u_i) \approx g(u_i, ..., u_{i-M}) \tag{4}$$

to predict some scalar target signal y by the estimator $\hat{y}$. The latter can be seen as a function g of covariate time series u, where the finite fading memory of the underlying system x causes g to be a function of at most $M+1$ predictor variables $u_i, ..., u_{i-M}$, where $\mathbb{N} \ni M \ll \infty$. The memory capacity of the system, indicated by M, represents the availability for computation of past input values u_i across τ-cycles. For details, the reader is referred to [2]. A schematic illustration of such a DCR is given in figure (2) and compared to a classical neural network with spatially extended nodes. The α_j are free parameters of the statistical model. The simplest way to determine the coefficients is by linear regression, i.e. using the *least squares solution* minimizing the sum of squared errors, $\sum_i (y_i - \hat{y}_i)^2$. However, in general, this approach will only be feasible in case of a noise-free target y and on large data sets. For a more rigorous statistical treatment of uncertainty, the problem can be formalized, for example, within the framework of Bayesian statistics or Gaussian process regression (see 3.5 and appendix). The delay span τ has, due to the recursive nature of (1), no impact on the memory capacity and can simply be chosen to harbor an adequate number N of virtual nodes along the delay line. Typically, $N = 400$ strikes a good balance between computational cost and reservoir performance in benchmark tasks.

In order to study the DCR model (4), system (1) has to be solved and virtual nodes sampled accordingly. However, (1) can neither be solved directly, due to the recursive terms, nor exists, to the best of our knowledge, a series expansion of the solution (for example of *Peano-Baker* type), due to the nonlinearity of f. Typically, (1) is therefore solved numerically using a lower order Runge-Kutta method, such as Heun's method. Although for the simple fixed point regime, the system operating in a numerical step size of $\theta/2$ is sufficient, the computational cost for large numbers of virtual nodes $N \gg 500$ can still be quite excessive if the hyperparameters are unknown and have to be determined.

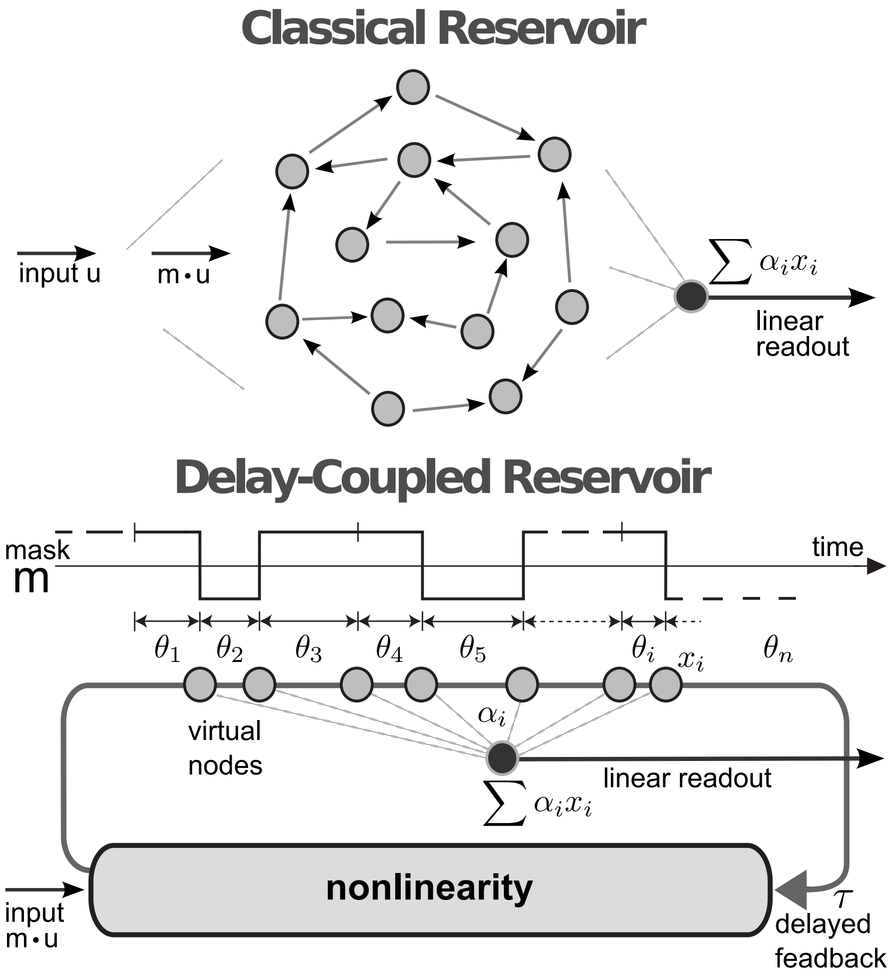


Fig. 2 Schematic illustration of a DCR (bottom), where nonlinear units (*virtual nodes*) are arrayed in time along the delay span τ, with temporal distance θ_j between virtual nodes $j-1$ and j. In contrast, a classical reservoir (top) has nonlinear units extended in space according to a certain network topology.

From a modeling perspective, the hyperparameters θ_j, τ, N, $m(t)$, γ, η are shrouded in the nonlinear non-solvable retarded functional differential equation (1), hardly accessible to optimization. Furthermore, due to the piecewise sampling procedure of the virtual nodes, the shape of m is inconveniently restricted and entangled with the sampling points θ_j, as well as the numerical simulation grid. If the latter is chosen too fine-grained, the computational costs become prohibitive in the scheme described above. Task-specific optimization of the hyperparameters is, however, crucial prior to any hardware implementation. In an optical implementation, for example, τ may directly determine the length of the glass-fibre cable that projects the delayed feedback back into the system. Determining optimal hyperparameters therefore has to be done in advance by means of numerical simulation of the system and with respect to rigorous statistical optimality criteria that will allow for proper confidence in the resulting permanent hardware setup.

To address these issues it is necessary to study system (1) in some detail. The goal is to gain insight into the dynamics of the system underlying the DCR, and to understand its theoretical solution as a semi-flow in an infinite-dimensional state space. These insights will form the basis for an approximate analytical solution scheme, alleviating access to the system as a functional statistical model and its hyperparameters. In addition, the approximate solution gives rise to a fast simulation algorithm, which will be key to the analysis and optimization of the DCR.

2.2 Retarded Functional Differential Equations

Following [7], let $C_\tau := C([-\tau,0],\mathbb{R})$ denote the Banach space of continuous mappings from $[-\tau,0]$ into $\mathbb{R}$, equipped with the supremum norm. If $t_0 \in \mathbb{R}$, $A \geq 0$ and $x : [t_0-\tau, t_0+A] \to \mathbb{R}$ a continuous mapping, then $\forall t \in [t_0, t_0+A]$, one can define $C_\tau \ni x_t : C_\tau \to C_\tau$ by $x_t(\sigma) = x(t+\sigma)$ for $\sigma \in [-\tau,0]$. Furthermore, let $H : C_\tau \to \mathbb{R}$ be a mapping such that

$$\frac{dx(t)}{dt} = H(x_t), \tag{5}$$

then (5) is called a *retarded functional differential equation*. A solution to (5) is a differentiable function x satisfying (5) on $[t_0, t_0+A]$ such that $x \in C([t_0-\tau, t_0+A),\mathbb{R})$. If H is locally Lipschitz continuous then x is unique, given an initial condition $(t_0,\phi) \in \mathbb{R}\times C_\tau$.

To illustrate this, consider a solution $x(t)$ of system (2) for $t \geq 0$, where

$$h : \mathbb{R}\times\mathbb{R} \to \mathbb{R} \tag{6}$$

such that

$$h : (x(t), x(t-\tau)) \mapsto \frac{\eta(x(t-\tau)+\gamma m(t)u(\bar{t})}{1+(x(t-\tau)+\gamma m(t)u(\bar{t}))^p} - x(t)$$

as given in (2), where mask m and input u are known (recall $m(t)u(\bar{t}) = J(t)$) and $\bar{t} = \lceil \frac{t}{\tau} \rceil$. Assume $p = 1$ for illustration in this section. The system will depend on its history during $[-\tau,0]$ as specified by

$$\phi : [-\tau,0] \to \mathbb{R}.$$

If ϕ is continuous, then h is continuous and locally Lipschitz in $x(t)$, since f is differentiable and $\sup|\frac{d}{dx}h(x,\phi)| = 1$. As a result, for $t \in [0,\tau]$

$$\frac{dx(t)}{dt} = h(x(t),\phi(t-\tau)), \quad t \in [0,\tau], \quad x(0) = \phi_0(0)$$

specifies an initial value problem that is solvable. Denote this solution on $t \in [0,\tau]$ by ϕ_1. Then

$$\frac{dx(t)}{dt} = h(x(t), \phi_1(t-\tau)), \quad t \in [\tau, 2\tau]$$

becomes solvable, too, since $x(\tau) = \phi_1(\tau)$ is already given. One can iterate this procedure to yield solutions to (2) on all intervals $[(i-1)\tau, i\tau]$, subject to some initial condition $\phi_0 = x|_{[-\tau,0]}$. This procedure is know as the *method of steps* [7].

The function $x_t : C_\tau \to C_\tau$, defined above as

$$x_t(\sigma) = x(t+\sigma), \quad \sigma \in [-\tau, 0],$$

specifies a translation of the segment of x on $[t-\tau, t]$ back to the initial interval $[-\tau, 0]$. Together with $x_0 = \phi_0$, the solution to system (16) (and, more generally, to (1)) can be shown to yield a semiflow $[0,\infty] \ni t \mapsto x_t \in C_\tau$.

It is now apparent that $\sigma \in [-\tau, 0]$ corresponds to a parameterization of *coordinates* for an "*infinite vector*" $x(\sigma)$ in state space C_τ, which is why (1) really constitutes an infinite-dimensional dynamical system. Delay-coupled reservoir computation can thus be thought of as expanding an input time series nonlinearly into an infinite-dimensional feature state space. However, given the sampling scheme of *virtual nodes*, and the piecewise constant mask, these properties will be effectively reduced to the number of samples N that constitute the covariates of the statistical model (4).

To study the stability properties of system (16), consider the autonomous case with constant input $J(t) = const$, and recall the temporal evolution h of system x as given by (6). For a fixed point x^* it holds that

$$\frac{dx}{dt} = h(x^*, x^*) = 0$$

Setting $p = \gamma = 1$ for illustration, solving for x^* evaluates to

$$\begin{aligned} 0 &= \frac{\eta(x^* + J)}{1 + (x^* + J)} - x^* \\ x^* &= \frac{\eta - 1 - J}{2} \pm \sqrt{\left(\frac{1 + J - \eta}{2}\right)^2 + \eta J}. \end{aligned} \tag{7}$$

To simplify the expression, let $J = 0$, in which case one obtains

$$x^* = \frac{\eta - 1}{2} \pm \frac{1 - \eta}{2}.$$

Of interest is now the central solution $x^* = 0$, around which the reservoir will be operated later on by suitable choice of η and γ. To determine its stability, one linearizes the system in a small neighborhood of x^* by dropping all higher order terms in the Taylor series expansion of h, which gives

$$\frac{dx}{dt} = D_x[h](x^*,x^*)x(t) + D_y[h](x^*,x^*)x(t-\tau) \tag{8}$$

where

$$\begin{aligned} D_x[h](x^*,x^*) &= \frac{\partial}{\partial x}h(x,y)|_{x=y=x^*} = -1 \\ D_y[h](x^*,x^*) &= \frac{\partial}{\partial y}h(x,y)|_{x=y=x^*} = \frac{\eta}{1+x^*}. \end{aligned} \tag{9}$$

If $D_x[h](x^*,x^*) + D_y[h](x^*,x^*) < 0$ and $D_y[h](x^*,x^*) \geq D_x[h](x^*,x^*)$, $x^* = 0$ is *asymptotically stable* (Theorem 4.7, [25]), which is the case for $\eta \in [-1,1)$. In general, the analysis of eq. (8) may be quite involved and can result in an infinite number of linearly independent solutions $x(t) = Ce^{\lambda t}$, since the corresponding characteristic equation $\lambda = D_x[h](x^*,x^*) + D_y[h](x^*,x^*)e^{-\lambda\tau}$ gives rise to an analytic function that may define roots on the entire complex plane.

Equations (7) and (9) already indicate that, given suitable η and γ, for $J(t) = m_j u_i$ a fixed point regime may still exist between two virtual node sampling points $\theta_{j-1} < t \leq \theta_j$, in which the reservoir computer can be operated while driven by input time series u. However, for $p > 1$ bifurcations can occur (even a period doubling route to chaos), if the feedback term is weighted too strongly. In this case, the virtual nodes would not yield a consistent covariate feature expansion and, in all likelihood, lead to a bad performance of the statistical model (4). The above considerations also show that the nonlinear feature expansion desired in reservoir computation will depend completely on the mask $m(t)$, since the input u_i is constant most of the time and $x(t)$ practically converges in a fraction of τ. The changes in m are in fact the only source of perturbation that prevents the semiflow $x(t)$ from converging to its fixed point. It is not at all clear in this situation that a piecewise constant binary (or even n-valued) mask is the most interesting choice to be made. More research is needed to determine optimal mask functions.

2.3 *Approximate Virtual Node Equations*

In the following, we discuss a recursive analytical solution to equation (1), employing the method of steps. The resulting formulas are used to derive a piecewise solution scheme for sampling points across τ that correspond to the reservoir's virtual nodes. Finally, we use the *trapezoidal rule* for further simplification, hereby deriving approximate virtual node equations, the temporal dependencies of which only consist of other virtual nodes. As will be shown in the remainder of this article, the resulting closed-form solutions allow reservoir computation without significant loss of performance as compared to a system obtained by explicit numerical solutions, e.g. *Heun's method* ((1,2) Runge-Kutta).

First, we discuss a simple application of the method of steps. System (1) is to be evaluated for the span of one τ during the ith τ-cycle, so $(i-1)\tau \leq t \leq i\tau$. Let a continuous function $\phi_{i-1}(\sigma) \in C_{[(i-2)\tau,(i-1)\tau]}$ be the solution for $x(t)$ on the previous τ-interval. We can now replace the unknown $x(t-\tau)$ by the known $\phi_{i-1}(t-\tau)$ in equation (1). Consequently, (1) can be solved as an ODE where the *variation of* *constants* [9] is directly applicable. The variation of constants theorem states that a real valued differential equation of type

$$\frac{dy}{dt} = a(t)y + b(t)$$

with initial value $y(t_0) = c \in \mathbb{R}$ has exactly one solution, given by

$$y(t) = y_h(t)\left(c + \int_{t_0}^{t} \frac{b(s)}{y_h(s)} ds\right), \tag{10}$$

where

$$y_h(t) = exp\left(\int_{t_0}^{t} a(s)ds\right)$$

is a solution of the corresponding homogeneous differential equation $\frac{dy}{dt} = a(t)y$. In system (1), we identify $a(t) = -1$ and $b(t) = f(\phi_{i-1}(t-\tau), J(t))$. Applying (10) now yields immediately the solution to the initial value problem (1) on the interval $t_{i-1} = (i-1)\tau \leq t \leq i\tau$, with initial value $x(t_{i-1}) = \phi_{i-1}((i-1)\tau)$, given by

$$x(t) = \phi_{i-1}(t_{i-1})e^{t_{i-1}-t} + e^{t_{i-1}-t}\int_{(i-1)\tau}^{t} f(\phi_i(s-\tau), J(s))e^{s-t_{i-1}}ds. \tag{11}$$

Recall that the semi-flow corresponding to $x(t)$ is determined by the mapping $x_t : C_\tau \rightarrow C_\tau$, defined above as $x_t(\sigma) = x(t+\sigma)$ with $\sigma \in [-\tau, 0]$. This specifies a translation of the segment of x on $[t-\tau, t]$ back to the initial interval $[-\tau, 0]$. Accordingly, we can reparametrize the solution for $x(t)$ in terms of $\sigma \in [-\tau, 0]$. Let x_i denote the solution on the ith τ-interval, then

$$x_i(\sigma) = x_{i-1}(0)e^{-(\tau+\sigma)} + e^{-(\tau+\sigma)}\int_{-\tau}^{\sigma} f(x_{i-1}(s), m(s)u_i)e^{s+\tau}ds, \tag{12}$$

where u_i denotes the constant input in the i^{th} reservoir time step, and we assume $m(\sigma)$ only has a finite number of discontinuities (see (3)). Accordingly, $x_{i-1} \in C_{[-\tau,0]}$.

Due to the recursion in the nonlinear $x_{i-1} = \phi_{i-1}$, the integral in (12) cannot be solved analytically. To approximate the integral, the recursion requires repeated evaluation at the same sampling points in each τ-cycle. The family of *Newton-Cotes formulas* for numerical integration is appropriate in this situation. We use the *cumulative trapezoidal rule* [20], which is 2^{nd} order accurate and the simplest in that family. It is given by

$$\int_a^b g(x)dx \approx \frac{1}{2}\sum_{j=1}^{N}(\chi_j - \chi_{j-1})(g(\chi_j) + g(\chi_{j-1})), \quad \text{with} \quad \chi_0 = a, \chi_N = b. \tag{13}$$

To approximate the integral in (12), consider a non-uniform grid $-\tau = \chi_0 < ... < \chi_N = 0$, where $\chi_j - \chi_{j-1} = \theta_j$. This yields the approximation

$$\begin{aligned} x_i(\chi_k) &\approx x_{i-1}(\chi_N)\exp(-\sum_{i=1}^{k}\theta_i) \\ &+\exp(-\sum_{i=1}^{k}\theta_i)[\frac{1}{2}\sum_{j=1}^{k}\theta_j\exp(\sum_{i=1}^{j-1}\theta_i)(f[x_{i-1}(\chi_j),m(\chi_j)u_i]e^{\theta_j} \\ &+f[x_{i-1}(\chi_{j-1}),m(\chi_{j-1})u_i]], \end{aligned} \tag{14}$$

which can be computed as cumulative sum in one shot per τ-cycle for all approximation steps χ_j simultaneously.

We are now interested in equations for $1 \leq k \leq N$ single virtual nodes x_{ik}, during reservoir time step i. At this point we will choose an equidistant sampling grid of virtual nodes. This is not crucial but simplifies the notation considerably for the purpose of illustration. Assuming equidistant virtual nodes, it holds that $\tau = N\theta$ where N is the number of virtual nodes. For application of formula (13), the numerical sampling grid will be uniform and chosen directly as the sampling points of virtual nodes, such that $\chi_j = -\tau + j\theta$ with $j = 0,...,N$. To get an expression for x_{ik}, we now have to evaluate equation (12) at the sampling point $t = -\tau + k\theta$, which results in

$$\begin{aligned} x_{ik} &= x_i(k\theta) = x((i-1)\tau + k\theta) \\ &\approx e^{-k\theta}x_{(i-1)N} + \frac{\theta}{2}e^{-k\theta}f[x_{(i-2)N}, J_N(i-1)] \\ &+\frac{\theta}{2}f[x_{(i-1)k}, J_k(i)] + \sum_{j=1}^{k-1}\underbrace{\theta e^{(j-k)\theta}}_{c_{kj}}f[x_{(i-1)j}, J_j(i)]. \end{aligned} \tag{15}$$

Here $J_k(i) = m_k u_i$ denotes the masked input to node k at reservoir time step i, given a mask that is piecewise constant on each θ-interval (see eq. 3).

Note that equation (15) only has dependencies on sampling points corresponding to other virtual nodes. An exemplary coupling coefficient is indicated by c_{kj}, weighting a nonlinear coupling from node j to node k. In addition, each node receives exponentially weighted input from virtual node N (first term eq. (15)). In analogy to a classical reservoir with a spatially extended network topology, we can derive a corresponding weight matrix. Figure (3) shows an exemplary DCR weight matrix for a temporal network of 20 virtual nodes, equidistantly spaced along a delay span $\tau = 4$ with distance $\theta = 0.2$. The lower triangular shape highlights the fact that a DCR corresponds to a classical reservoir with ring topology. A ring topology features the longest possible feed-forward structure in a given network. Since the memory capacity of a recurrent network is mainly dependent on the length of its internal feed-forward structures, the ring topology is most advantageous with respect to the network's finite fading memory [6].

Formula (15) allows simultaneous computation of all nodes in one reservoir time step (τ-cycle) by a single vector operation, hereby dramatically reducing the computation time of simulating the DCR by several orders of magnitude in practice, as compared to an explicit second order numerical ODE solver.

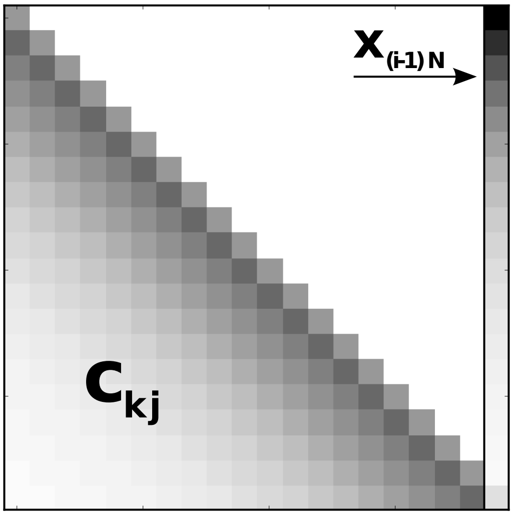


Fig. 3 Illustration of a *temporal* weight matrix for a DCR comprised of 20 virtual nodes, arrayed on a delay line of length $\tau = 4$ with equidistant spacing $\theta = 0.2$. The lower triangular part of the matrix corresponds to the c_{kj} of formula (15), where darker colour indicates stronger coupling. The last column indicates the dependence on node $x_{(i-1)N}$, as given by the first term in expression (15).

3 Implementation and Performance of the DCR

We compare the analytical approximation of the Mackey-Glass DCR, derived in the previous section, to a numerical solution obtained using Heun's method with a stepsize of 0.1. The latter is chosen due to the relatively low computational cost and provides sufficient accuracy in the context of DCR computing. As a reference for absolute accuracy, we use numerical solutions obtained with *dde23* [24], an adaptive (2,3) Runge-Kutta based method for delay differential equations. The nonlinearity f is chosen according to the Mackey-Glass equation with $p = 1$ for the remainder of this chapter, such that the system is given by

$$\frac{dx}{dt} = -x(t) + \frac{\eta(x(t-\tau) + \gamma J(t))}{1 + x(t-\tau) + \gamma J(t)}, \tag{16}$$

where η, γ and p are metaparameters, τ the delay length, and $J(t)$ is the temporally stretched input $u(\bar{t})$, $\bar{t} = \lceil \frac{t}{\tau} \rceil$, multiplexed with a binary mask m (see eq. 3).

Note that the trapezoidal rule used in the analytical approximation, as well as Heun's method, are both second order numerical methods that should yield a global truncation error of the same complexity class. As a result, discrepancies originating from different step sizes employed in the two approaches (e.g. 0.2 in the analytical approximation and 0.1 in the numerical solution) may be remedied by simply decreasing θ in the analytical approximation, for example by increasing N while keeping a fixed τ (see sec. 3.4).

3.1 Trajectory Comparison

In a first step, we wish to establish the general accuracy of the analytical approximation in a DCR relevant setup. Figure 4 shows a comparison of reservoir trajectories computed with equation (15) (red) against trajectories computed numerically using *dde23* (blue) with relative error tolerance 10^{-3} and absolute error tolerance 10^{-6}. The system received uniformly distributed input $u(\bar{t}) \sim \mathscr{U}_{[0,0.5]}$. The sample points correspond to the activities of $N = 400$ virtual nodes with a temporal distance of $\theta = 0.2$, and $\tau = 80$ accordingly. Given 4000 samples (corresponding to 10 reservoir time steps $\bar{t}$), the mean squared error between the trajectories is *MSE* $= 5.4 \times 10^{-10}$. As can be seen in the figure, the trajectories agree very well in the fixed point regime of the system (autonomous case). Although it is expected that the *MSE* would increase in more complex dynamic regimes (e.g. chaos), the latter are usually not very suitable for a DCR for various reasons. The following results also show a high task performance of the analytical approximation when used for DCR computing.

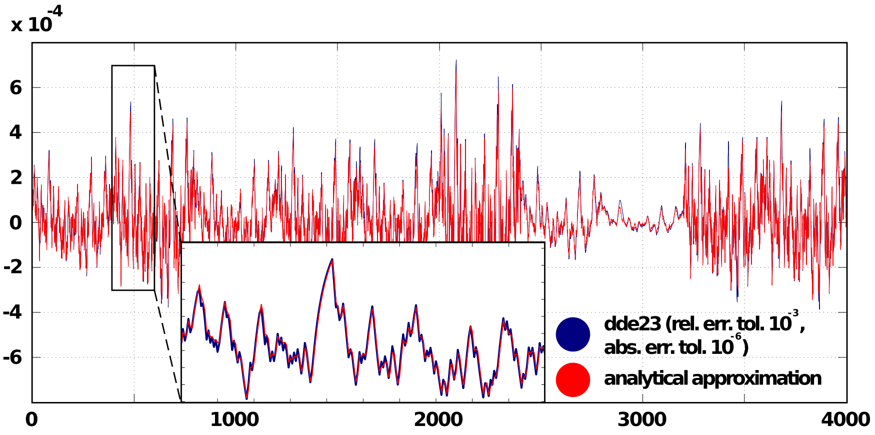


Fig. 4 Comparison between analytical approximation and numerical solution for an input-driven Mackey-Glass system with parameters $\eta = 0.4$, $\gamma = 0.005$ and $p = 1$, sampled at the temporal positions of virtual nodes, with a distance $\theta = 0.2$.

3.2 NARMA-10

A widely used benchmark in reservoir computing is the capacity of the DCR to model a nonlinear autoregressive moving average system y in response to uniformly distributed scalar input $u(k) \sim \mathscr{U}_{[0,0.5]}$. The NARMA-10 task requires the DCR to compute at each time step k a response

$$y(k+1) = 0.3y(k) + 0.05y(k)\sum_{i=0}^{9} y(k-i) + 1.5u(k)u(k-9) + 0.1.$$

Thus, NARMA-10 requires modeling of quadratic nonlinearities and shows a strong history dependence that challenges the DCR's memory capacity. We measure performance in this task using the correlation coefficient $r(y,\hat{y}) \in [-1,1]$ between the target time series y and the DCR output $\hat{y}$ in response to u. Here, the DCR is trained (see sec. 2.1) on 3000 data samples, while $r(y,\hat{y})$ is computed on an independent validation data set of size 1000. Figure 5A summarizes the performance of 50 different trials for a DCR computed using the analytical approximation (see eq. 15), shown in red, as compared to a DCR simulated with Heun's method, shown in blue. Both reservoirs consist of $N = 400$ virtual nodes, evenly spaced with a distance $\theta = 0.2$ along a delay line $\tau = 80$. Both systems show a comparable performance across the 50 trials, with a median correlation coefficient between $r(y,\hat{y}) = 0.96$ and 0.97, respectively.

3.3 5-Bit Parity

As a second benchmark, we chose the delayed 5-bit parity task [23], requiring the DCR to handle binary input sequences on which strong nonlinear computations have to be performed with arbitrary history dependence. Given a random input sequence u with $u(k) \in \{-1,1\}$, the DCR has to compute at each time step k the parity $p_m^\delta(k) = \prod_{i=0}^{m} u(k-i-\delta) \in \{-1,1\}$, for $\delta = 0,\ldots,\infty$. The *performance* ϕ_m is then calculated on n data points as $\phi_m = \sum_{\delta=0}^{\infty} \kappa_m^\delta$, where *Cohen's Kappa*

$$\kappa_m^\delta = \frac{\frac{1}{n}\sum_{k=1}^{n} \max(0, p_m^\delta(k)\hat{y}(k)) - p_c}{1-p_c} \in \{0,1\}$$

normalizes the average number of correct DCR output parities $\hat{y}$ by the chance level $p_c = 0.5$. We used 3000/1000 data points in training and validation set respectively. To compare performance between analytical approximation and numerical solution of the DCR, we chose $m = 5$ and truncated ϕ_m at $\delta = 7$, so that $\phi_5 \in [0,7]$. For parameters $\eta = 0.24$, $\gamma = 0.032$ and $p = 1$, and a DCR comprised of 400 neurons ($\tau = 80$), figure 5B shows that performance ϕ_5 is comparable for both versions of the DCR, with median performances between 4.3 and 4.5. across 50 different trials of this task. As the performance is far from the ideal value of 7 and the model suffers slightly from overfitting (not shown), it is clear that the delayed 5-bit parity task is a hard problem which leaves much space for improvement.

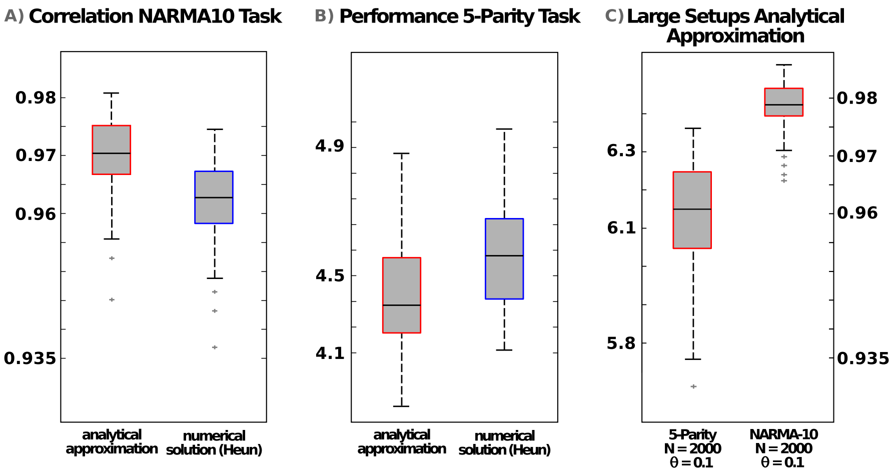


Fig. 5 Comparison on nonlinear tasks between analytical approximation and numerical solution for an input-driven Mackey-Glass system, sampled at the temporal positions of virtual nodes with a distance $\theta = 0.2$. Mackey-Glass parameters are $\eta = 0.4$, $\gamma = 0.005$ and $p = 1$ (NARMA-10) and $\eta = 0.24$, $\gamma = 0.032$ and $p = 1$ (5-bit parity), respectively. Results are reported for 400 neurons ($\tau = 80$) on data sets of size 3000/1000 (training/validation) in figures 5A and 5B, size 3000/1000 in 5C (right plot), as well as for data sets of size 10000/10000 in figure 5C (left plot). Each plot is generated from 50 different trials. The plots show median (black horizontal bar), $25^{th}/75^{th}$ percentiles (boxes), and most extreme data points not considered outliers (whiskers).

3.4 *Large Setups*

We repeated the tasks in larger network setups where the computational cost of the numerical solver becomes prohibitive. In addition to increasing the number of virtual nodes N one can also decrease the node distance θ, thus fitting more nodes into the same delay span τ. Although too small θ may affect a virtual node's computation negatively, decreasing θ increases the accuracy of the analytical approximation.

3.4.1 NARMA-10

We illustrate this by repeating the NARMA-10 task with $N = 2000$ virtual nodes and $\tau = 200$. This results in $\theta = 0.1$, corresponding to the step size used in the numerical solution before. Note that this hardly increases the computational cost of the analytical approximation since the main simulation loop along reservoir time steps $\bar{t}$ (τ-cycles) remains unchanged. The results are summarized for 50 trials in figure 5C (right boxplot). The median correlation coefficient increased significantly to nearly 0.98 while the variance across trials is notably decreased (compare fig. 5A).

3.4.2 5-Bit Parity

For the 5-bit parity task, we addressed the task complexity by increasing both, training and validation sets, to a size of 10000. Second, we increased once more the virtual network size to $N = 2000$ virtual nodes and $\tau = 200$. The performance of the resulting DCR setup, computed across 50 trials using the analytical approximation, is summarized in figure 5C (left boxplot). The model no longer suffers as much from overfitting and the performance on the validation set increased dramatically to a median value of 6.15, which is now close to the theoretical limit of 7.

3.5 Application to Experimental Data

When analyzing experimental data that is subject to measurement noise and uncertainty, a more sophisticated statistical model than presented so far is required. In this section, we embed the output of a DCR into a proper Bayesian statistical framework and discuss practical difficulties that may arise. A part of this discussion and further motivation for the statistical model can be found in the appendix. As exemplary task we chose the one-step ahead prediction of the Santa Fe laser time series, which is an experimental recording of a far-infrared laser operating in a chaotic regime [12]. These data consist of 9000 samples as supplied in the Santa Fe time-series competition [29]. Employing Bayesian model selection strategies with a Volterra series model [22] (which can be thought of as a Taylor series expansion of the functional (4)), we found that the prediction of a time series sample $y(t+1)$ required at least 8 covariates $y(t), y(t-1), ..., y(t-7)$ to capture the auto-structure, and a model with 4^{th} order nonlinearities, which makes the one-step ahead prediction of the chaotic Santa Fe laser data an interesting and challenging task for a DCR.

When predicting experimental data, in addition to computing a point estimator it is also important to quantify the model confidence in some way, as a result of the potential variability of the estimator (across data sets) and one's ignorance about certain aspects of the target quantity. In a Bayesian approach, this uncertainty is summarized in a probability distribution. If the task is to predict a target time series y in terms of a covariate time series u, the uncertainty in the prediction of unseen data y_* given covariate time series u_* is summarized in a predictive distribution $P(y_*|u_*, u, y, H)$. The predictive distribution incorporates knowledge of the given data $\mathscr{D} = (y, u)$ to infer y_* given u_*. We denote by H the particular modeling assumption that has to be made and the corresponding hyperparameters. The covariance structure of this distribution represents the model uncertainty and supplies confidence intervals.

To derive a predictive distribution, first recall the DCR functional model (4),

$$\hat{y}_i := g(\bar{u}_i) = g(u_i, u_{i-1}, ..., u_{i-M}) = \sum_{j=1}^{N} \alpha_j x_{ij},$$

where x_{ij} denotes the j^{th} virtual node (15) in reservoir time step $\bar{t} = i$, and the dependence on covariates $\bar{u}_i = \{u_{i-1}, ..., u_{i-M}\}$ is given implicitly in each x_{ij} via the temporal evolution of $x(t)$ (11). Let

$$X = \begin{pmatrix} x_{11} & \dots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nN} \end{pmatrix} \in \mathbb{R}^{n \times N}, \quad \alpha \in \mathbb{R}^N, \tag{17}$$

then

$$\hat{y} = X\alpha \in \mathbb{R}^n \tag{18}$$

shall denote the estimator of target time series y, given covariate time series u.

We can now state a statistical model. First, define a multivariate isotropic normal prior distribution for α as

$$\alpha \sim \mathcal{N}(0, \lambda^2 I), \tag{19}$$

where $I \in \mathbb{R}^{N \times N}$ is the identity matrix. We denote the corresponding density by $p(\alpha|\lambda^2)$. This choice of isotropic prior results effectively in an L_2-regularized model fit [11].

Given data $y, u \in \mathbb{R}^n$, the standard modeling assumption about the target time series y is that of noisy measurements of an underlying process $g(\bar{u}_i)$, stated as

$$y_i = g(u_i, ..., u_{i-M}) + \varepsilon_i. \tag{20}$$

Here, $\forall i = 1, ..., n : \varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$, and it is usually assumed that ε_i is independent of ε_j for $j \neq i$. Note that the assumption of normality is also implicitly present in any model fit given by the *least squares* solution as the minimizer of the *sum-of-squared-errors* objective (compare section 2). It follows that the sampling distribution for $y \in \mathbb{R}^n$ is given by

$$y \sim \mathcal{N}(\mathbf{g}(u), \sigma_\varepsilon^2 I), \tag{21}$$

where $\mathbf{g}(u) = (g(\bar{u}_1), ..., g(\bar{u}_n))^T \in \mathbb{R}^n$ and $I \in \mathbb{R}^{n \times n}$. We denote the density of this distribution by $p(y|u, \alpha, \lambda^2, \sigma_\varepsilon^2)$.

One can now derive all distributions necessary to formalize the uncertainty associated with data (y, u). The details of these derivations can be found, for example, in [4]. First, Bayes formula supplies an expression for the posterior distribution that summarizes our uncertainty about α. The corresponding density is given as

$$p(\alpha|y, u, \lambda^2, \sigma_\varepsilon^2) = \frac{p(y|u, \alpha, \lambda^2, \sigma_\varepsilon^2)p(\alpha|\lambda^2)}{p(y|u, \lambda^2, \sigma_\varepsilon^2)}, \tag{22}$$

where the normalizing constant $p(y|u, \lambda^2, \sigma_\varepsilon^2)$ will be referred to as *marginal likelihood*. The posterior distribution of the model coefficients α can be explicitly computed as a normal distribution in this case. Given this posterior, one can compute the predictive distribution as a convolution of the sampling distribution (21) with the posterior of model coefficients, which will again be normal with density

$$p(y_*|u_*,u,y,\lambda^2,\sigma_\varepsilon^2) = \int_{\mathbb{R}^N} p(y_*|u_*,\alpha,\sigma_\varepsilon^2)p(\alpha|y,u,\lambda^2,\sigma_\varepsilon^2)d\alpha. \tag{23}$$

The full distribution can be derived as

$$P(y_*|u_*,u,y,\lambda^2,\sigma_\varepsilon^2) = \mathcal{N}(m_*,S_*), \tag{24}$$

with

$$\begin{aligned} m_* &:= \mathbb{E}[y_*] = X_*(X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X^Ty \\ S_* &:= \mathrm{Cov}[y_*] = \sigma_\varepsilon^2 I + X_*(X^TX\frac{1}{\sigma_\varepsilon^2} + \frac{1}{\lambda^2}I)^{-1}X_*^T. \end{aligned} \tag{25}$$

These terms are numerically computable. The predictive distribution (24) summarizes all uncertainty related to our forecasting of y_* and provides m_* as point estimator $\hat{y} = X_*\hat{\alpha}$, where $\hat{\alpha} = (X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X^Ty$ denotes the expected value of the coefficient posterior (22). From a *decision theoretic* point of view, it can also be shown that m_* minimizes the *expected* squared error loss $(y_* - \hat{y})^2$ [3]. For an explanation of how to derive estimators $\hat{\sigma}_\varepsilon^2, \hat{\lambda}^2$ and a short discussion on how to deal with hyperparameters in a proper Bayesian model, the interested reader is referred to the appendix.

To evaluate the model, one can compute $\hat{y}_v = m_v(\hat{\sigma}_\varepsilon^2, \hat{\lambda}^2)$ from (25) on a separate validation set y_v, conditional on training data (u_t, y_t). However, not only would one always want to condition the prediction on all available data, and not just on a fixed training data set y_t, but a lower number of conditional data samples in a prediction may also lead to a reduced performance and a higher susceptibility to overfitting. It may therefore seem more natural to compute the *leave-one-out* (LOO) predictive distribution $P(y_i|u_i,u_{\setminus(i)},y_{\setminus(i)},\hat{\sigma}_\varepsilon^2,\hat{\lambda}^2)$, where $(u_{\setminus(i)},y_{\setminus(i)})$ denotes the full data set with the i^{th} data point removed. The joint LOO predictive distribution for all data points i supplies its mean vector $\hat{y}_{cv} \in \mathbb{R}^n$ as optimal predictor for the given data y. Accordingly, this yields a cross-validated performance measure by e.g. the correlation coefficient $r(y, \hat{y}_{cv})$.

To yield a confidence interval for the correlation coefficient and somehow quantify our uncertainty in this statistic, note that all uncertainty pertaining to r is governed by the predictive distribution (24). Using the LOO predictive distribution accordingly, one can perform a parametric resampling of the data, yielding new data sets $y^* \in \mathbb{R}^n$ with mean vector $\hat{y}_{cv}$. With these, it is possible to recompute $r(\mathbb{E}[y] = \hat{y}_{cv}, y^*)$ for each resampled time series y^* to get an empirical distribution for r and quantify its expected spread on data sets of similar size. The resulting empirical distribution over r can be used to determine confidence bounds as indicators for the variability of the goodness of fit. Alternatively, although time-consuming, non-parametric bootstrapping could be employed by resampling design matrix rows (17) and used with the *BCa* method [5] to yield confidence intervals that are corrected for deviation from normality in several ways.

Applying the above scheme to the Santa Fe laser time series, we set up an auto-regressive model with $u_i = y(i-1)$ to predict $y(i)$. We will first use only

$n = 600$ samples in the training data set to illustrate the importance of formalizing uncertainty and test the DCR model with a varying number of virtual nodes. As goodness-of-fit measure the squared correlation coefficient $r^2 \in [0,1]$ will be computed either on the leave-one-out cross-validated prediction of the training data set y_t ($r^2(y_t, \hat{y}_{cv})$), or as actual prediction of the 600 consecutive data points y_v given y_t ($r^2(y_v, \hat{y}_v)$). The data will be mean corrected and rescaled using the (5,95) percentiles. As such, the DCR parameters can be chosen again as $\gamma = 0.005, \eta = 0.4$ to maintain a proper dynamical range of the underlying Mackey-Glass system given the normalized input. Furthermore, $p = 1$ and $\tau = 100$ are set and operated with a randomly sampled binary mask m. Results will be reported for $N = 100, 250, 500, 1000$ virtual nodes respectively, leading to different uniform sampling positions $\theta \in \mathbb{R}^N$ in the τ-cycle with fixed length 100. In this section, we use equation (14) as an approximate solution scheme with step-size $k := \chi_j - \chi_{j-1} = 0.01$ on an equidistant approximation grid $\chi_0 = 0, ..., \chi_{10000} = \tau = 100$. For example, the DCR with 500 virtual nodes will have an equidistant sampling grid according to $\theta_j = \tau/500 = 0.2$. In this case, the activity of a virtual node during θ_j is computed using $\theta_j/k = 20$ samples from the underlying approximation grid.

The results on the Santa Fe data are summarized in table 1. It can be seen that all predictions conditional on merely $n = 600$ data points suffer from overfitting, as there is a noteworthy difference between predictions on the training set data and the two types of cross-validation. The models with $N = 500$ and $N = 1000$ virtual nodes appear to have no noteworthy difference in performance. In figure (6), 500 points of the validation set are shown for the DCR model with 500 virtual nodes. The predictive distribution is computed conditional on the first 600 data points of the time series and confidence intervals for the individual data points are derived as two standard deviations ($\sqrt{S_*}$) above and below the estimated expected value (m_*, see eq. (25)). The two main characteristics of the Santa Fe time series are its irregular oscillatory behavior, as well as several sudden breakdowns of the amplitude, occurring infrequently in intervals of several hundred data points. As highlighted in the magnification of figure (6), around these rare events the confidence intervals

Table 1 Results on the Santa Fe data set: N denotes the number of virtual nodes used in the DCR model and n is the number of data points used in the training set (that is, the number of data on which the predictive distribution is conditioned). The goodness-of-fit is measured by the squared correlation coefficient $r^2 \in [0,1]$ and evaluated for an estimate $\hat{y} = m_*$ from equation (25) and the corresponding actual data y. First, the training set estimate $\hat{y}_t \in \mathbb{R}^n$ conditional on the whole training set y_t is considered, then the *leave-one-out* cross-validated estimator $\hat{y}_{cv} \in \mathbb{R}^n$ (see text above), and finally an estimate $\hat{y}_v \in \mathbb{R}^{600}$ for a validation set of size 600, conditional on the preceding training set y_t.

	n=600, N=100	n=600, N=250	n=600, N=500	n=600, N=1000	n=5000, N=250
$r^2(y_t, \hat{y}_t)$	0.952	0.953	0.975	0.975	0.997
$r^2(y_t, \hat{y}_{cv})$	0.932	0.934	0.947	0.943	0.995
$r^2(y_v, \hat{y}_v)$	0.922	0.928	0.945	0.946	0.997

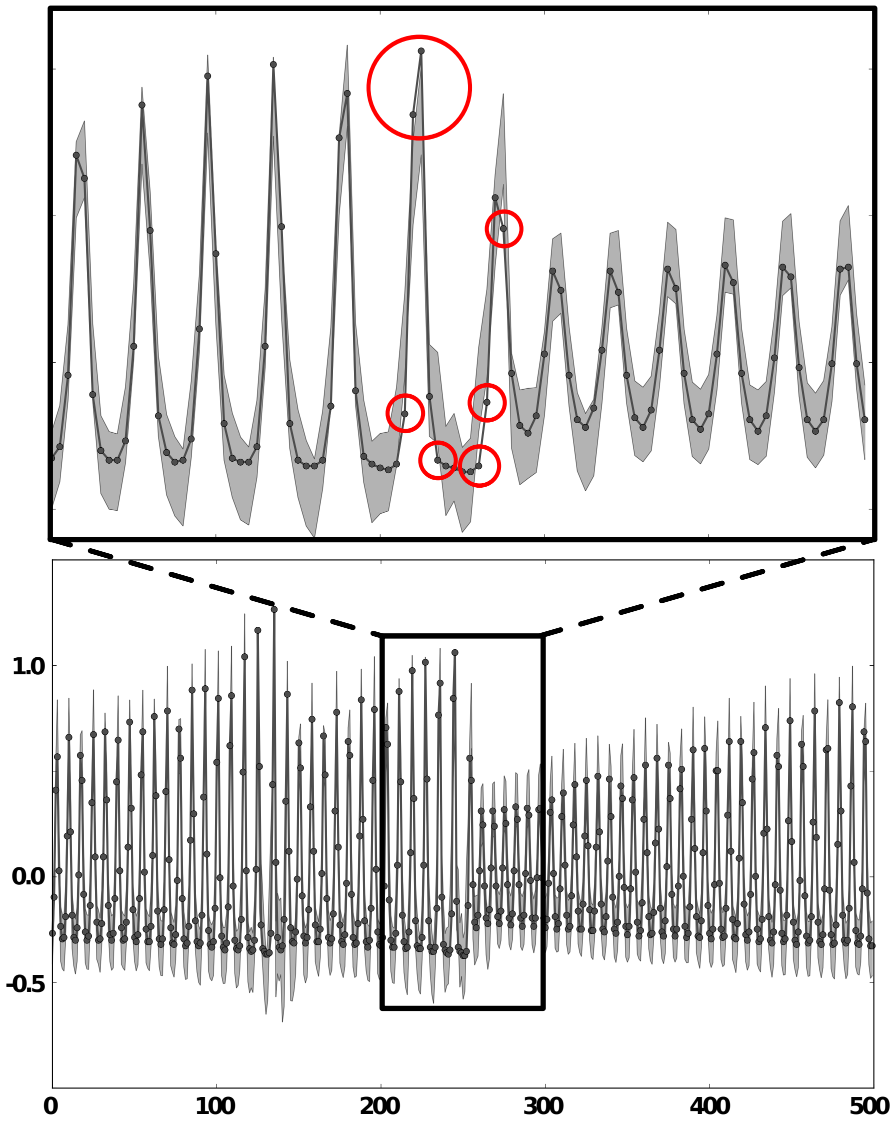


Fig. 6 Normalized data points 600 to 1100 of the Santa Fe data, corresponding to a validation set. In gray, a confidence interval of 2 standard deviations of the predictive distribution is shown, as computed using a model with 500 virtual nodes, trained on the first 600 samples of the time series. It can be seen that the confidence intervals lose accuracy in a neighborhood around the sudden amplitude change after the 250^{th} data point.

are less accurate and don't always contain the sample. Presumably, the predictions conditional on merely $n = 600$ data points do not contain enough information to account for the rare events. In contrast, using $n = 5000$ training data may increase the

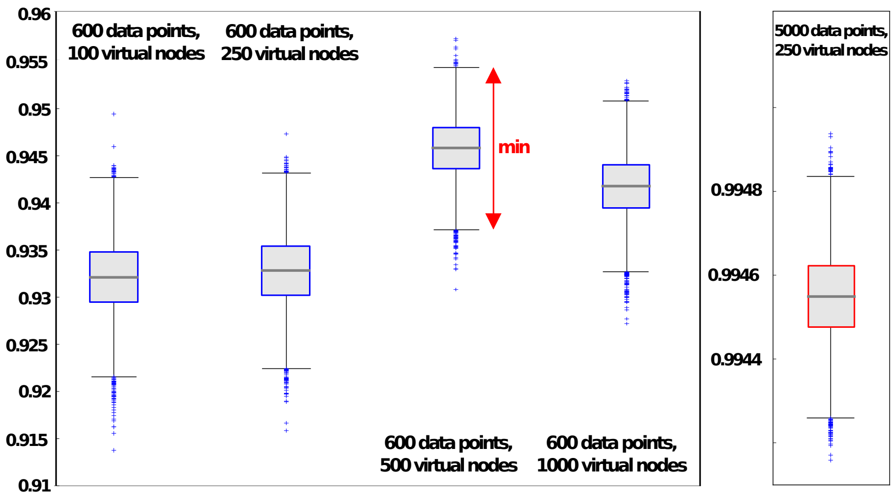


Fig. 7 Squared correlation coefficient r^2 of *leave-one-out* cross-validated prediction $\hat{y}_{cv}$ with parametrically resampled training data sets y^*_t (see text). The boxes denote quartiles, whiskers 95% confidence intervals, as given by the empirical distribution upon resampling 10000 times $r^2(\hat{y}_{cv} = \mathbb{E}[y_t]_{cv}, y^*_t)$. The model with $N = 500$ virtual nodes shows the lowest variance among predictions conditional on $n = 600$ training data points.

accuracy of the prediction around these rare events, as more information is available to the model. As can be seen in the last column of table 1, the performance is much improved and shows no longer signs of overfitting.

Figure 7 shows in addition for each model the variability associated with the r^2 goodness-of-fit measure, as computed using the parametric resampling strategy described earlier. The $N = 500$ model shows the smallest estimated variance and it becomes obvious from this figure that an increase of virtual nodes to $N = 1000$ leads to overfitting. In general, all predictions conditional on merely $n = 600$ data points show a substantial variability in accuracy, which highlights the necessity for a proper quantification of this variability in any report of performance. In contrast, the predictions conditional on $n = 5000$ data points have a very low estimated variance and high accuracy, which suggests that the data set is in fact not very noisy. The goodness-of-fit is elevated to state-of-the-art levels with a squared correlation coefficient of $r^2(y_t, \hat{y}_{cv}) = 0.99$, as can be seen in the last figure on the right.

4 Discussion

In summary, we provided in this chapter a general introduction to the emerging field of delay-coupled reservoir-computing. To this end, a brief introduction to the theory of delay differential equations was discussed. Based on these insights, we have developed analytical approaches to evaluate and approximate solutions of delay differential equations that can be used for delay-coupled reservoir computing.

In particular, we derived approximate closed-form equations for the virtual nodes of a DCR. It has been shown that the resulting update equations in principle lose neither accuracy with respect to the system dynamics nor computational power in DCR benchmark tasks. Using the analytical approximation reduces computational costs considerably. This enabled us to study larger networks of delay-coupled nodes, yielding a dramatic increase in nonlinear benchmark performance that was not accessible before. These results can lead to serious improvement regarding the implementation of DCRs on electronic boards.

Moreover, the approach yields an explicit handle on the DCR components which are otherwise implicit in equation (1). This creates new possibilities to investigate delay-coupled reservoirs and provides the basis for optimization schemes, a crucial necessity prior to any hardware implementation. Together with the reduction in computation time, this makes the use of supervised batch-update algorithms feasible to directly optimize model hyperparameters (see eq. (16) and appendix). A few of these possibilities were illustrated in a practical application to an experimental recording of a far-infrared laser operating in a chaotic regime, where the DCR model was embedded in a fully Bayesian statistical model. The relation to a potential application in Gaussian process regression is discussed in the appendix, in light of numerical difficulties that may arise with a DCR.

Future research will be focused on optimal hyperparameters of the DCR and improved adaptability to input signals. Foremost, optimal mask functions have to be identified systematically. To this end, the inconvenient coupling to the sampling grid of virtual nodes has to be overcome, so as to make an independent evaluation of mask functions possible. Accounting for and optimizing non-uniform virtual node sampling grids could be an interesting next step (compare [28]). In addition, optimization procedures may include unsupervised gradient descent schemes on DCR parameters (e.g. θ, τ, N) with respect to other information theoretic objectives. Continuing this line of thought, one may even try to modify the update equations directly according to self-organizing homeostatic principles, inspired, for example, by neuronal plasticity mechanisms (e.g. [17]). We intend to explore these possibilities further in future work to maximize the system's computational power and render it adaptive to information content in task-specific setups.

Acknowledgements. The authors acknowledge the financial support of the State Lower Saxony, Germany via the University of Osnabrück, and the European project PHOCUS in the Framework 'Information and Communication Technologies' (FP7-ICT-2009-C/Propopsal Nr. 240763). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Appendix

In this section, we expand and motivate the statistical model employed in section 3.5. A proper statistical model allows one to treat uncertainty associated with the data in a formally optimal way. The authors believe that the greatest *theoretical* rigor in this regard is achieved with Bayesian statistics (see for example [18]). In a first step, we therefore choose to formalize the model accordingly. In a second step, we try to implement the theory as far as possible while dealing with practical issues such as numerical accuracy, data size and computability.

Recall the DCR functional model (4),

$$\hat{y}_i := g(\bar{u}_i) = g(u_i, u_{i-1}, ..., u_{i-M}) = \sum_{j=1}^{N} \alpha_j x_{ij},$$

where x_{ij} denotes the j^{th} virtual node (15) in reservoir time step $\bar{t} = i$, and the dependence on covariates $u_{i-1}, ..., u_{i-M}$ is given implicitly in each x_{ij} via the temporal evolution of $x(t)$ (11). We chose an isotropic prior $\alpha \sim \mathcal{N}(0, \lambda^2 I)$, which corresponds effectively to an L_2 regularization. The regularization term plays an important part in safeguarding the model from overfitting. Further arguments (see [14],[3]) suggest this prior represents our state of ignorance optimally, since (19) maximizes entropy of α given mean and variance $(0, \lambda^2)$, while being invariant under a certain set of relevant transformations. It is thus the *least informative* choice of a prior distribution, in addition to the assumptions following from the role of α in model (18).

The likelihood function (21) is also normal, as a result of the normal choice for the distribution of residuals ε_i. Although we may have no reason to believe that ε_i is actually normally distributed, one can again make strong points that the normal distribution nonetheless represents our state of knowledge optimally, unless information about higher moments of the sampling distribution is available [14]. In practice, the latter will often not be the case, since there is no explicit access to the residuals ε_i.

From these considerations, the predictive distribution (25) can be derived analytically, as was suggested in an earlier section. Note that, equivalently, we could have derived these formulas in a framework of *Gaussian Process Regression*. The resulting expressions, though equivalent, would look slightly different. The Gaussian Process framework has many interesting advantages and allows for elegant derivations of the necessary distributions in terms of Bayesian statistics, in particular with regard to the hyperparameters $(\sigma_\varepsilon^2, \lambda^2)$. It has thus become an important theoretical and practical tool in modern machine learning approaches and would have been our first choice for a statistical model. In the following, we will therefore discuss the predictive distribution in light of Gaussian processes, allude to difficulties in applying this framework to DCR models, and present a practical alternative for deriving estimates for the hyperparameters $(\sigma_\varepsilon^2, \lambda^2)$.

A Gaussian Process is a system of random variables indexed by a linearly ordered set, such that any finite number of samples are jointly normally distributed [10]. A Gaussian Process thus defines a distribution over functions and is completely specified by a *mean function* and a *covariance function*. In terms of a Gaussian process, we can define the DCR functional model $g \sim \mathcal{GP}$ as

$$\begin{aligned} \mathbb{E}[g] &= 0, \\ \mathrm{Cov}[g(\bar{u})] &= \mathbb{E}[g(\bar{u}_i)g(\bar{u}_j)] \\ &= X\mathbb{E}[\alpha\alpha^T]X^T = \lambda^2 XX^T =: [K(u,u)]_{ij}, \end{aligned} \tag{26}$$

where $[K]_{ij}$ denotes coordinate i,j of matrix K.

One can now derive all distributions necessary to formalize the uncertainty associated with data (y,u). The details of these derivations can be found, for example, in [21]. The covariance matrix K_y corresponding to y is given by

$$K_y := \mathrm{Cov}[g(\bar{u}) + \varepsilon] = \mathrm{Cov}[g(\bar{u})] + \mathrm{Cov}[\varepsilon] = K(u,u) + \sigma_\varepsilon^2 I.$$

Accordingly, for data (y_*,u_*) (which may be the same as (y,u)),

$$\begin{pmatrix} y \\ y_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(u,u)+\sigma_\varepsilon^2 I & K(u,u_*) \\ K(u_*,u) & K(u_*,u_*) \end{bmatrix}\right). \tag{27}$$

If one is interested in predicting the noisy measurement of y_*, adding σ_ε^2 to $K(u_*,u_*)$ is appropriate and was presumed in the earlier derivations of section 3.5. From (27), the marginal likelihood can be derived as $y|u \sim \mathcal{N}(0,K_y)$ with log-density

$$\log[p(y|u)] = -\frac{1}{2}y^T K_y^{-1} y - \frac{1}{2}\log[|K_y|] - \frac{n}{2}\log[2\pi]. \tag{28}$$

Furthermore, the predictive distribution can be derived as

$$P(y_*|u_*,u,y) = \mathcal{N}(m_*,S_*), \tag{29}$$

with

$$\begin{aligned} m_* &:= \mathbb{E}[y_*] = K(u_*,u)K_y^{-1}y \\ S_* &:= \mathrm{Cov}[y_*] = K(u_*,u_*) - K(u_*,u)K_y^{-1}K(u,u_*). \end{aligned} \tag{30}$$

To see that the predictive distribution (25) derived earlier and (30) are in fact equivalent, consider the pseudo-inverse Φ^+ of a matrix Φ,

$$\Phi^+ = \lim_{\delta\downarrow 0}(\Phi^T\Phi + \delta I)^{-1}\Phi^T = \lim_{\delta\downarrow 0}\Phi^T(\Phi\Phi^T + \delta I)^{-1}.$$

These limits exists even if $\Phi^T\Phi$ or $\Phi\Phi^T$ are not invertible. In addition, the equality also holds for $\delta > 0$ [1]. This allows us to rewrite (30) as

$$\begin{aligned} m_* := \mathbb{E}[y_*] &= K(u_*,u)K_y^{-1}y \\ &= X_*X^T(XX^T + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}y \\ &= X_*(X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X^Ty \end{aligned}$$

$$\begin{aligned} S_* := \mathrm{Cov}[y_*] &= K(u_*,u_*) - K(u_*,u)K_y^{-1}K(u,u_*) \\ &= K(u_*,u_*) - X_*(X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X^TXX_*^T\lambda^2 \\ &= \lambda^2 X_*(I - (X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X^TX)X_*^T \\ &= \lambda^2 X_*(\frac{\sigma_\varepsilon^2}{\lambda^2}(X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1})X_*^T \\ &= \sigma_\varepsilon^2 X_*(X^TX + \frac{\sigma_\varepsilon^2}{\lambda^2}I)^{-1}X_*^T. \end{aligned} \tag{31}$$

Unfortunately, K_y has deplorable numerical properties regarding inversion, as necessary in (30). This is most likely owed to the fact that the individual virtual nodes have a very high pairwise correlation across time, as can be expected of samples from a smooth system. Recall that $K(u,u) = \lambda^2 X^TX \in \mathbb{R}^{n\times n}$. Since typically there will be much less virtual nodes than data points, $N < n$, so that $K(u,u)$ is usually rank deficient. Although in theory K_y should always be invertible, numerically this fact tends to hold only if $\sigma_\varepsilon^2/\lambda^2 \geq 10^{-12}$. Note that this ratio corresponds to the L_2-regularization parameter. While this is a common problem in Gaussian process regression and poses for many functional models no severe difficulties, the covariance matrix built from the reservoir samples is very fickle in this regard: Setting $\sigma_\varepsilon^2/\lambda^2 \geq 10^{-12}$ can already lead to severe loss of performance in non-noisy reservoir benchmark tasks such as NARMA-10. Using the standard formulation of the predictive distribution (25) allows one to sidestep these numerical issues. In both frameworks, however, the marginal likelihood in (22) is given by $y|u \sim \mathcal{N}(0,K_y)$ with log-density

$$\log[p(y|u)] = -\frac{1}{2}y^TK_y^{-1}y - \frac{1}{2}\log[|K_y|] - \frac{n}{2}\log[2\pi].$$

The marginal likelihood is important for Bayesian model selection, e.g. in computing a *Bayes Factor* or *Posterior Odds*. Given the numerical troubles discussed above, a straight-forward application of Bayesian model selection therefore seems unavailable to DCR models.

In a fully Bayesian treatment, the hyperparameters $\sigma_\varepsilon^2, \lambda^2$ would have to be assigned (non-informative) priors and be integrated out of the distributions relevant for inference or model selection. However, in general it is not possible to get rid of dependence on both, hyperparameters and α. Instead, explicit values for the hyperparameters may be estimated by maximizing, for example, the marginal likelihood (28), or the predictive distribution (24) in a leave-one-out cross-validation scheme [27]. With respect to computability of the involved terms, given the particular numerical difficulties arising in the DCR setup, a compromise between theory and realizability is needed. We found a good practical performance to be achieved by the following method. Looking at the marginal likelihood (28), one notes that it contains essentially a term that reflects how well the model fits the data, and another term that measures the complexity of the model as a function of K_y.

A similar approach is given by the information criterion AIC_M [15], which can be stated as

$$\mathrm{AIC}_M := n(\log(2\pi)+1)+n\log(\hat{\sigma}_\varepsilon^2)+2[\mathrm{tr}(G)+1]. \tag{32}$$

Here, G denotes a *smoother matrix*, i.e. a matrix that is multiplied with the data y to arrive at a prediction. From the first equation in (25), which represents our optimal predictor, we can infer

$$G(\lambda^2) = X_*(X^TX+\frac{\hat{\sigma}_\varepsilon^2}{\lambda^2}I)^{-1}X^T. \tag{33}$$

The term $\mathrm{tr}(G)$ is called *effective number of parameters* and was proposed by Hastie et al. [8] to control the complexity of a model. The model fit to the data in (32) is given by $\log(\hat{\sigma}_\varepsilon^2)$ as a function of an estimator of the residual variance. Although one could estimate $\hat{\sigma}_\varepsilon^2$ along with λ^2, it is usually possible to express one as a function of the other, thus simplifying the optimization problem. To account for generalization of the model with respect to prediction performance, we choose

$$\hat{\sigma}_\varepsilon^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i-\mu^{(i)})^2. \tag{34}$$

The term $\mu^{(i)}$ denotes the predictive estimator of y_i, obtained in a *leave-one-out cross-validation* scheme by computing $m_* = m_i$ in equation (25) with the ith data point removed from X, y. This can be efficiently done in one shot for all i [15], yielding

$$\mu^{(i)} = \frac{[Gy]_i - [G]_{ii}y_i}{1-[G]_{ii}}. \tag{35}$$

One can now determine

$$\hat{\lambda}^2 = \underset{\lambda^2}{\mathrm{argmax}}\ \mathrm{AIC}_M(\lambda^2)$$

and compute $\hat{\sigma}_\varepsilon^2$ as a function of $\hat{\lambda}^2$ accordingly. In addition, the AIC_M score can be used to perform a selection between models of varying complexity, for example to compare DCRs employing different numbers of virtual nodes.

References

1. Albert, A.: The Penrose-Moore Pseudo Inverse with Diverse Statistical Applications. Part I. The General Theory and Computational Methods. Defense Technical Information Center (1971)
2. Appeltant, L., Soriano, M.C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I.: Information processing using a single dynamical node as complex system. Nature Communications 2, 468 (2011)
3. Berger, J.O.: Statistical Decision Theory and Bayesian Analysis. Springer Series in Statistics. Springer (1985)
4. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus (2006)

5. Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. Chapman & Hall, New York (1993)
6. Ganguli, S., Huh, D., Sompolinsky, H.: Memory traces in dynamical systems. Proceedings of the National Academy of Sciences 105(48), 18970–18975 (2008)
7. Guo, S., Wu, J.: Bifurcation Theory of Functional Differential Equations. Applied Mathematical Sciences. Springer London, Limited (2013)
8. Hastie, T.J., Tibshirani, R.J.: Generalized Additive Models. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis (1990)
9. Heuser, H.: Lehrbuch der Analysis. Number pt. 1 in Mathematische Leitfäden. Teubner Verlag (2009)
10. Hida, T., Hitsuda, M.: Gaussian Processes. Translations of Mathematical Monographs. American Mathematical Society (2007)
11. Hoerl, A.E., Kennard, R.W.: Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12(1), 55–67 (1970)
12. Huebner, U., Abraham, N.B., Weiss, C.O.: Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared nh3 laser. Phys. Rev. A 40(11), 6354–6365 (1989)
13. Jäger, H.: The echo state approach to analysing and training recurrent neural networks. Technical report (2001)
14. Jaynes, E.T., Bretthorst, G.L.: Probability Theory: The Logic of Science. Cambridge University Press (2003)
15. Konishi, S., Kitagawa, G.: Information Criteria and Statistical Modeling. Springer Series in Statistics. Springer (2008)
16. Larger, L., Soriano, M.C., Brunner, D., Appeltant, L., Gutierrez, J.M., Pesquera, L., Mirasso, C.R., Fischer, I.: Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. Opt. Express 20(3), 3241–3249 (2012)
17. Lazar, A., Pipa, G., Triesch, J.: SORN: a self-organizing recurrent neural network. Frontiers in Computational Neuroscience 3 (2009)
18. Lindley, D.V.: The 1988 wald memorial lectures: The present position in bayesian statistics. Statistical Science 5(1), 44–65 (1990)
19. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Computation 14(11), 2531–2560 (2002)
20. Quarteroni, A., Sacco, R., Saleri, F.: Numerical Mathematics, 2nd edn. Texts in Applied Mathematics, vol. 37. Springer, Berlin (2006)
21. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptative computation and machine learning series. University Press Group Limited (2006)
22. Rugh, W.J.: Nonlinear system theory: the Volterra/Wiener approach. Johns Hopkins series in information sciences and systems. Johns Hopkins University Press (1981)
23. Schrauwen, B., Buesing, L., Legenstein, R.A.: On computational power and the order-chaos phase transition in reservoir computing. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) NIPS, pp. 1425–1432. Curran Associates, Inc. (2008)
24. Shampine, L.F., Thompson, S.: Solving ddes in matlab. Applied Numerical Mathematics 37, 441–458 (2001)
25. Smith, H.: An Introduction to Delay Differential Equations with Applications to the Life Sciences. Texts in Applied Mathematics. Springer (2010)
26. Soriano, M.C., Ortín, S., Brunner, D., Larger, L., Mirasso, C.R., Fischer, I., Pesquera, L.: Optoelectronic reservoir computing: tackling noise-induced performance degradation. Optics Express 21(1), 12–20 (2013)

27. Sundararajan, S., Sathiya Keerthi, S.: Predictive approaches for choosing hyperparameters in gaussian processes. Neural Computation 13(5), 1103–1118 (2001)
28. Toutounji, H., Schumacher, J., Pipa, G.: Optimized Temporal Multiplexing for Reservoir Computing with a Single Delay-Coupled Node. In: The 2012 International Symposium on Nonlinear Theory and its Applications (NOLTA 2012) (2012)
29. Weigend, A., Gershenfeld, N. (eds.): Time series prediction: forecasting the future and understanding the past. SFI studies in the sciences of complexity. Addison-Wesley (1993)

Double-Layer Vector Perceptron for Binary Patterns Recognition

Vladimir Kryzhanovskiy and Irina Zhelavskaya

Abstract. A new model – Double-Layer Vector Perceptron (DLVP) – is proposed. Compared with a single-layer perceptron, its operation requires slightly more computations (by 5%) and more effective computer memory, but it excels at a much lower error rate (four orders of magnitude lower). The estimate of DLVP storage capacity is obtained.

Keywords: vector neural networks, Potts model.

1 Introduction

The Potts model [1-2] is the first and the most well known vector neural network. The model still draws much attention from researchers in such fields as physics, medicine, image segmentation and neural networks. Later, the parametric neural network [3] was offered and thoroughly studied by a small group of the Institute of Optical Neural Technologies of Russian Academy of Sciences (the Center of Optical Neural Technologies of the System Research Institute of RAS today). A similar model (CMM) was developed independently and is still investigated at York University [4]. V. Kryzhanovsky's thesis introduces a vector neural network model with a proximity measure between neuron states. This kind of neural networks generalizes all above-mentioned models. Researchers studied both fully connected and perceptron-like architectures. Various vector-net learning rules were studied [6]. The results proved the high efficiency of vector networks.

Perceptron is most suitable for associative memory-based applications (in our case it is a vector perceptron). However, it has a major drawback: even one output neuron taking a wrong state results in an input vector not being recognized.

Vladimir Kryzhanovskiy · Irina Zhelavskaya
Scientific-Research Institute for System Analysis of Russian Academy of Sciences (SRISA RAS), 36/1 Nahimovskiy Ave., Moscow, Russia, 117218
e-mail: vladimir.krizhanovsky@gmail.com

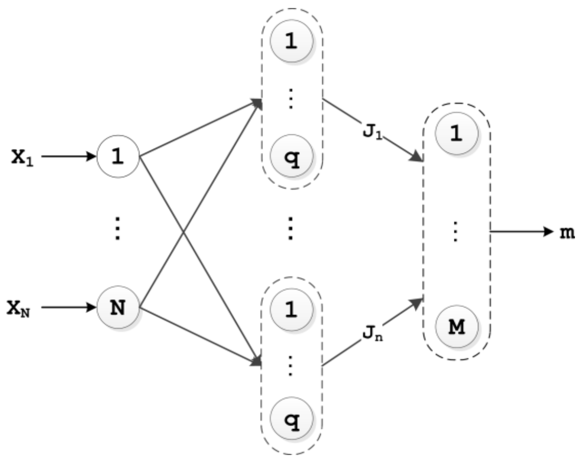


Fig. 1 The general arrangement of the double-layer vector perceptron

To overcome this, one has to raise the reliability of each neuron by increasing the net redundancy or decreasing the load of the net. In other words, the vector perceptron consists of "reliable" neurons that cannot make mistakes, which contradicts the whole philosophy of neural networks.

The alternative approach is to use weak neurons. With similar requirements for RAM, a collection of weak neurons proves to be more effective than a small number of reliable neurons. The trick is to supply a vector perceptron with an additional layer consisting of only one neuron that has a number of states equal to the number of stored patterns. Its aim is to accumulate the information from the preceding layer and to identify an input pattern. The approach is close to the idea offered in papers [7, 8].

The paper consists of three parts: formal description of the model, qualitative description with a simple example that helps to understand the point of the approach, and experimental results.

2 Setting Up the Problem

In this paper we are solving the nearest neighbor search problem, which consists in the following. Let us have a set of M N-dimensional bipolar patterns:

$$\mathbf{X}_\mu \in R^N,\ x_{\mu i} \in \{\pm 1\},\ \mu \in \overline{1,M}\,. \tag{2.1}$$

A bipolar vector **X** is applied to the inputs of the network. The goal is to find reference pattern $\mathbf{X}_m$ with the smallest Hamming distance to input pattern **X**.

3 Formal Description of the Model

3.1 Model Description

Let us consider double-layer architecture (Fig. 1). The input layer has N scalar neurons, each of which takes one of two states $x_i = \pm 1$, $i = 1, 2, \ldots, N$. The first (inner) layer consists of n vector neurons. Each of these neurons has $2q$ fictive states during the training, and is described by basis vectors of q-dimensional space $\mathbf{y}_i \in \{\pm\mathbf{e}_1, \pm\mathbf{e}_2, \ldots, \pm\mathbf{e}_q\}$, where $\mathbf{e}_k = (0,\ldots,0,1,0,\ldots,0)$ is the unit vector with k-th component equal to 1. These fictive states are applicable only during the training, and can be considered as responses of the inner layer of the network. That is done since we use Hebb rule for training, so the responses of the network should be known in advance. At the recognition stage, these neurons are simple summators (thus, there is no activation function in the inner layer). This is done to simplify the description of the model. The second (output) layer has one vector neuron that can take M states, and is described by basis vectors of M-dimensional space (where M is the number of patterns in the training set) $\mathbf{O} \in \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_M\}$.

The state of the perceptron is described by three vectors:

1) Input layer is described by N-dimensional bipolar vector $\mathbf{X} = (x_1, x_2, \ldots, x_N)$, where $x_i = \pm 1$;
2) The first (inner) layer is described by n-dimensional $2q$-nary vector $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$, where $\mathbf{y}_i \in \{\pm\mathbf{e}_1, \pm\mathbf{e}_2, \ldots, \pm\mathbf{e}_q\}$, and $\mathbf{e}_k = (0,\ldots,0,1,0,\ldots,0)$ is the q-dimensional unit vector with k-th component equal to 1;
3) The second (output) layer is described by M-nary vector $\mathbf{O} \in \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_M\}$, where $\mathbf{o}_r = (0,\ldots,0,1,0,\ldots,0)$ is the M-dimensional unit vector holding unit in the r-th digit.

Each reference pattern $\mathbf{X}_m$ is uniquely associated with vector $\mathbf{Y}_m$. In its turn, each vector $\mathbf{Y}_m$ is uniquely associated with vector $\mathbf{o}_m$. Each component of vector $\mathbf{Y}_m$ is generated in a way that on the one hand, $\mathbf{Y}_m$ is a unique vector, and on the other hand, possible states $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_q\}$ are distributed evenly among reference vectors, i.e. $\sum_\mu \mathbf{y}_{\mu i} \equiv \frac{M}{q}(1,1,\ldots,1)$. If the last condition is not satisfied, the error rate grows by several orders of magnitude, which was proved experimentally. So, we build a neural network that stores association:

$$\mathbf{X}_m \Leftrightarrow \mathbf{Y}_m \Leftrightarrow \mathbf{o}_m \tag{3.1}$$

3.2 Learning Procedure

The synaptic connections of the vector perceptron are computed using generalized Hebb's rule:

$$\mathbf{W}_{ji} = \sum_{m=1}^{M} \mathbf{y}_j^m x_i^m \quad \text{and} \quad \mathbf{J}_j = \sum_{m=1}^{M} \mathbf{o}_m^T \mathbf{y}_j^m , \qquad (3.2)$$

where $\mathbf{W}_{ji}$ is the q-dimensional vector describing the connection between the i-th neuron of the input layer and the j-th neuron of the inner layer; $\mathbf{J}_j$ is the $M\times q$ matrix responsible for connection between j-th neuron of the inner layer and the sole output neuron, $i = \overline{1,N}$, $j = \overline{1,n}$.

3.3 Identification Process

Let us apply vector **X** to the network inputs. Let us compute the response of the net **O**. For that purpose, let us first calculate local fields of the inner layer:

$$\mathbf{h}_j = \sum_{i=1}^{N} \mathbf{W}_{ji} x_i \qquad (3.3)$$

.

Since the inner-layer neurons act as simple summators during recognition, signal $\mathbf{h}_j$ arrives to the output neuron without any changes. That is why local field of the output layer has the form:

$$\mathbf{H} = \sum_{j=1}^{n} \mathbf{J}_j \mathbf{h}_j^T . \qquad (3.4)$$

The final output **O** is calculated in the following way. We identify the largest component of local field **H**. Let it be component r. Then, the output of the perceptron is $\mathbf{O} = \mathbf{o}_r$. In other words, the input of the perceptron receives a distorted variant of the r-th reference pattern. And the larger the product $(\mathbf{H}, \mathbf{o}_r)$ is, the more statistically reliable the response of the network is. Moreover, if we arrange the numbers of components in increasing order the resulting list will tell us how close to corresponding vectors input vector **X** is in terms of Hamming vicinity.

4 Qualitative Description of the Model

4.1 The General Idea

Each vector neuron corresponds to a unique partition of the whole set of reference patterns into q subsets. For instance, Fig. 2 shows us two partitions of the set of M=12 patterns into q=4 subsets. For any partition we can calculate q "probabilities" (components of the vector of local fields, h_j^k) of the input pattern belonging to one of the q subsets. Each vector neuron is basically a solver that selects a subset with the highest “probability” (in Fig. 3 it is subset No.1 in the first partition and subset No.1 in the second partition). The intersection of the subsets that were selected by all solvers determines the output of a single-layer perceptron. Calculations of the “probabilities” may contain errors due to the statistical nature of calculations. So, a solution found by selecting the “highest-probability” subsets might be wrong. Mistake in selecting a “winning” subset in at least one partition is enough to get a wrong solution (Fig. 3).

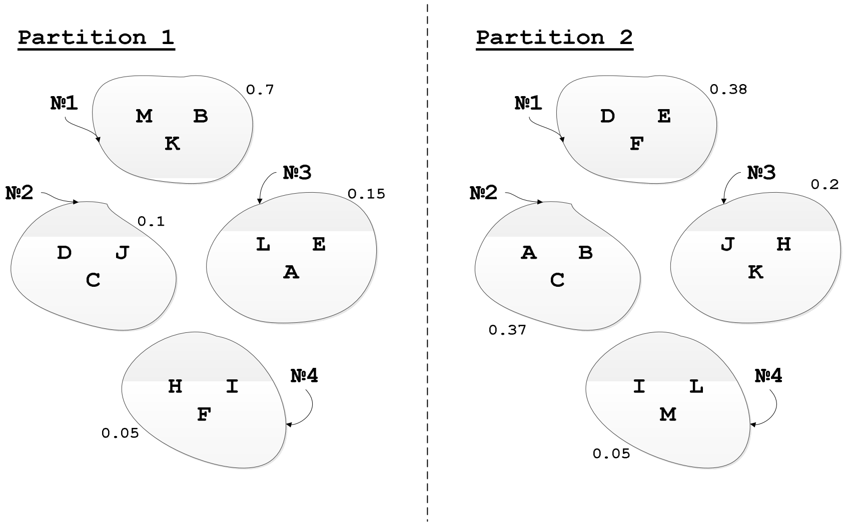


Fig. 2 Partition of a set of objects in two different ways

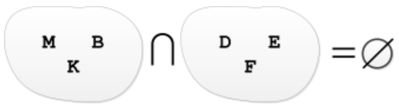


Fig. 3 Intersection of winning subsets from partition 1 and 2 results in a null subset

The goal of the proposed method is to overcome this drawback. The idea is to make decisions by accumulating “probabilities” over all partitions rather than using “probabilities” of partitions separately (and cutting off possible solutions by doing so). To do that, we need to interpret the probabilities $h_j^1, h_j^2, ..., h_j^q$ for j-th partition differently from what we did before. If previously we treated h_j^k as an indicator of k-th subset in j-th partition, now we will say that each element (pattern) of k-th subset in j-th partition is associated with the same indicator h_j^k . Thus, each pattern has n corresponding probabilities (where n is a number of different partitions of the total set), and their sum represents a cumulative indicator of this pattern. Using these cumulative indicators allows us to decide which of the patterns is the winner based on the information from all subsets of all partitions.

(*It should be noted once again that the "probability" here is understood as a certain statistical quantity – a component* h_j^k *of local field, to be exact. The higher the probability of an input pattern being a pattern from a subset corresponding to this local field, the larger this component is.*)

4.2 Example

Let us exemplify the idea. Fig. 2 shows two different partitions (n = 2) of a set of 12 letter-denoted patterns into 4 subsets. Let us apply distorted pattern B to the inputs. In the figure each subset has a corresponding number, which is the calculated "probability" that an input pattern is a pattern from this particular subset.

Table 1 Probability that the input pattern belongs to a particular subset

Partition 1			Partition 2		
Subset number	Objects	Probability*	Subset number	Objects	Probability*
1	**M, K, B**	**0.70**	**1**	**D, E, F**	**0.38**
2	**D, J, C**	0.10	**2**	**A, B, C**	0.37
3	**L, E, A**	0.15	**3**	**J, H, K**	0.20
4	**H, I, F**	0.05	**4**	**I, L, M**	0.05

*Probability - chances that the input pattern belongs to the subset.

When a single-layer perceptron is used for recognition, subset No.1 is the "winner" subset in the first partition, and it really contains the input pattern. In the second partition the "winner" is also subset No.1, yet it does not have the input pattern in it. The intersection of the two subsets gives us a null subset (Fig. 3), which means that the net cannot identify the input pattern. It is clear that the failure of one neuron causes the failure of the whole system. At the same time, we can see that the probabilities of the input pattern belonging to subset 1 or subset 2 for the second partition are almost equal – the difference is just 0.01 (1%) (Table 1). That is to say, it is almost equiprobable for the input pattern to be either in the first or the second subset. Our model takes this fact into account, and for each pattern the decision is made by using probabilities from both partitions (Table 2). The pattern that corresponds to the greatest total "probability" is selected as the response of the system. The result is a correct identification of the input pattern by the network.

Table 2 Recognition probabilities computed for two partitions and their sum for each pattern

Pattern	Probability for partition 1	Probability for partition 2	Summary probability
A	0.15	0.37	0.52
B	0.70	0.37	**1.07**
C	0.10	0.37	0.47
D	0.10	0.38	0.48
E	0.15	0.38	0.53
F	0.05	0.38	0.43

5 Details of the Algorithm

We can see from the table that the proposed model requires just 4-5% more computational resources (CPU, RAM) than the single-layer perceptron.

Table 3 Details of the algorithm

	Single layer	Two layers	Ratio*
Computational burden (number of operations)	$2Nnq$	$2Nnq+(n+1)M$	1.025
Necessary amount of RAM, bytes	$4Nnq$	$4Nnq+4nM$	1.033

* - the ratio is taken for $M = 100$; $N = 100$; $q = 300$; $n = 2$.

6 Storage Capacity

A new model of neural networks that is basically a product of adding one more layer to a single-layer perceptron was presented above. The value of this additional layer was illustrated via example in section 4.2. Now, we need to examine the properties of the model and to compare characteristics of a single- and double-layer perceptrons. This can be done in several ways:

1) We can take a range of datasets containing real data from different domains, and investigate how the proposed model and the single-layer perceptron do perform on them. As a result, we will identify types of data (types of problems), for which the networks described above work well and for which they do not. These results would be very important since through them we would be able to understand what place our model take among existing ones. The disadvantage of this approach is that a very deep analysis of the used data is required in order to understand the reasons why models work well or not, and this task is a very nontrivial task itself.

2) Another approach is to generate a number of synthetic datasets that will be considered as reference vectors (reference patterns), and test the models on them. In this case, it becomes possible to create the situations when the targeted models properties are most pronounced. The significant advantage of that is the possibility to calculate statistical characteristics such as expected mean, variance, correlations, etc., and different events probabilities analytically. Such estimates allow us to understand the endogenous processes in neural networks better.

It is obvious that for thorough investigation of the model it is necessary to go both ways. In the present work authors follow the second one: as reference patterns we use vectors, which components are generated independently with equal probabilities and take either +1 or -1.

So, with what purpose do we consider such kind of vectors? Our choice is based on several reasons. First, this case is the simplest one for analytical calculations. Second, the estimate of storage capacity would be an upper bound estimate in this case, i.e. we are estimating the maximum possible storage capacity of a neural network. So, for instance, it is well known that neural networks work worse when recognizing similar patterns, i.e. patterns that have correlations between them rather than patterns without correlations. That means that they are able to "remember" a fewer number of patterns a priori. Moreover, the probability of correct recognition highly depends on correlation values in each particular case. For that reason, we can make comparison between different models of associative memory only by using the upper bound estimate of the storage capacity for the simplest case. For example, the well-known result $0.14N$, which is the storage capacity estimate of Hopfield associative memory model, was obtained under the same assumptions.

Let us give a definition of the storage capacity. The storage capacity of associative memory is a number of reference patterns $M_{\max}$ that can be remembered by a neural network so that it can recognize all of them without error. By that it is understood that adding just one reference pattern to the training set will lead to the fact that one of the patterns is not being recognized correctly. In that event, the probability of error recognition equals to $1/(M_{\max}+1)$.

We may formulate this classical definition in a different way. The storage capacity of associative memory $M_{\max}$ is such number of reference patterns, recognizing which the probability of recognition error P is equal to $1/M_{\max}$[1]. At that, it is a common practice that neural networks are tested at reference patterns without any distortions. Authors think that it is necessary to generalize this definition, and to define $M_{\max}$ at condition that reference vectors being applied to the inputs of the network are distorted at some noise level a, and the probability of recognition error P is not greater than a predefined threshold $P_{\max}$ (value $P_{\max}$ could be any, including $1/M_{\max}$).

Authors managed to estimate storage capacity $M_{\max}$ for both models at the abovementioned conditions. Resulting estimates are in a good agreement with the experiment differing just 1-1.3 times in magnitude from experimental results. Detailed derivations of the following estimates are presented in Appendix 1:

1) Storage capacity of double-layer vector perceptron (DLVP):

$$M < C\frac{nqN(1-2a)^2}{8\ln\left(\frac{nqN}{4\sqrt{2\pi}P_{\max}}\right)},\ C = 2.5\,. \tag{6.1}$$

2) Storage capacity of single-layer perceptron:

$$M = \frac{qN(1-2a)^2}{2\ln\left(\frac{nqN}{\sqrt{2\pi}P_{\max}}\right)} \tag{6.2}$$

Let us analyze the storage capacity of both models. We may draw the following conclusions from (6.1) and (6.2):

1) Storage capacity of both models increases linearly with N, q;
2) Storage capacity of both models decreases quadratically with a rise of the distortion level of reference patterns a;
3) Strengthening the requirements for recognition reliability, i.e. reduction of accepted probability error $P_{\max}$, leads to log decrease of storage capacity for both models;
4) And most importantly, comparing these two estimates we can see that the storage capacity of a double-layer perceptron is n times greater than the storage capacity of a single-layer perceptron!

7 Experimental Results

In this section we explore the properties of the proposed model in the following experiments:

1. First, we show that adding the second layer to the network enhances the probability of the correct recognition of input vectors. For this purpose, we experimentally compare double- and single-layer perceptrons. In these experiments we will vary external parameters N, M, a.
2. Then, we investigate the model behavior depending on internal parameters n and q. Both parameters increment enhances the probability of correct patterns recognition. However, these parameters take different effect on the model. Increment of q results in decrease of the amount of information corresponding to one synaptic connection, and increment of n allows accumulating more statistical information.

Fig. 4 Probability *P* versus the number of stored patterns *M*. Parameters *N*=100, *q*=100, *n*=2

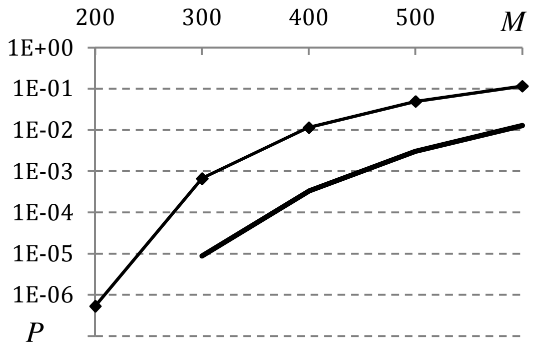
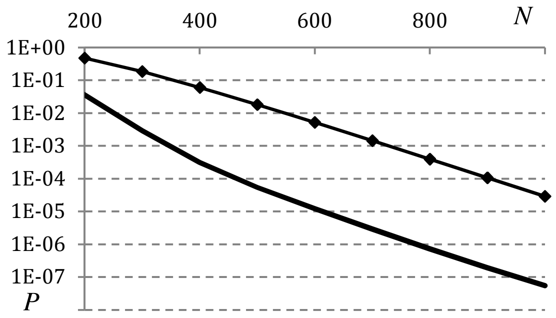


Fig. 5 Probability P versus dimensionality *N*. Parameters *M*= 1000, *q*=50, *n*=3



3. We conduct experiments on storage capacity of the proposed model, and verify the agreement between theoretical and practical results.
4. We also consider another useful option that is provided by the proposed model. That is a possibility of solving the K nearest neighbors task.

7.1 Comparison with a Single-Layer Perceptron

In this section we compare results of operation of a single- and double-layer perceptron.

In Figures 4-6 the Y-axis of the plots is the recognition error probability *P* (when the perceptron fails to recognize a distorted reference vector). In both figures the curves corresponding to the single-layer perceptron are represented by a thin line with rhombic marks (the curves are above the others). Other curves correspond to the double-layer perceptron. The plots are drawn for different *n* and *q*.

If the number of patterns *M*, their dimensionality *N,* and the noise level *a* (the probability of a component of an input binary vector being distorted) are determined by the conditions of a problem to be solved, the number of *q*-digit neurons of the inner layer and the number of their states can be varied to get satisfactory reliability.

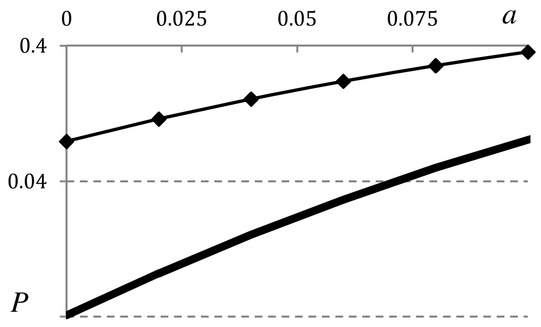


Fig. 6 Recognition failure probability P versus noise level a. M=1000, N=100, q= 200, n=2

Let us first consider how the recognition error probability varies with M and N given constant n and q (Fig. 4 and 5). As expected, the growth of dimensionality of stored patterns N or a decrease of their number M result in an exponential decrease of probability P. It is also seen that the introduction of another layer allows a more than an order of magnitude (two orders and more) decrease of P. The lower the probability P for the original single-layer net, the more significantly P decreases for the double-layer system.

The noise-resistance of the double-layer net is also higher – the rhomb-marked curve lies noticeably higher than the other curve (Fig. 6).

7.2 Model Properties Analysis

Fig. 7 shows us a few dependences of the double-layer network error probability P on the noise level a for different combinations of n and q (given $n*q$ = const). The upper dashed curve corresponds to n=40, q=10, the curve below – to n=8 and q=50. Even lower is the curve for n=4 and q=100. The combination of n=2 and q=200 (thick solid line) demonstrates the lowest P. So we see that from the reliability viewpoint it is better to use a small number of reliable (redundant) neurons for the double-layer system. However, such kind of networks cannot boast of high resistance to a failure of the net itself. The data (dashed line) shown in Fig. 7 proves that reliable and failure-resistant neural systems can be made up of unreliable elements having a considerable parameter spread.

The net with n=40 and q=10 differs from the net with n=2 and q=200 by the principles securing correct recognition. In the first case the second layer that accumulates information from a large number of unreliable elements plays a key role (for a single-layer perceptron with given parameters the recognition probability is zero). In the latter case, the second layer corrects the errors of the first layer only occasionally (thin marked line in Fig. 7).

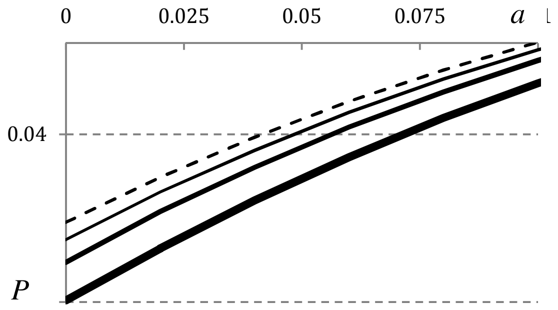


Fig. 7 Recognition failure probability P versus noise level a. M=1000, N=100

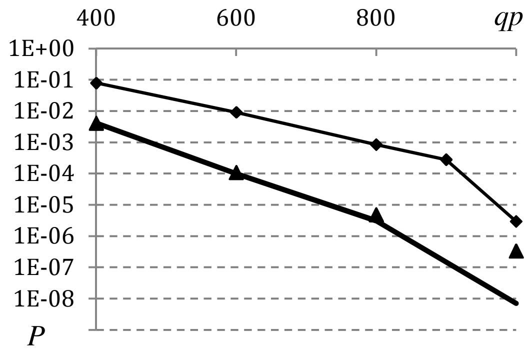


Fig. 8 Recognition failure probability P versus nq. M=1000, N=100, a=0

Fig. 8 shows how the error probability P depends on inner-layer parameters n and q. The thick line corresponds to the probability P of a double-layer network with n=2 and q=200÷500, and triangular marks correspond to n =2÷5 and q=200. Both networks have the same computational burden and requirements for RAM. The simulation shows that

1) The growth of both parameters leads to an exponential decrease of P;

2) Both nets has the same probability P for nq < 800 (an unexpected enough result), which once again says for the conclusion drawn above.

7.3 Storage Capacity

In this subsection we will present the experimental results of DLVP maximum storage capacity measurements and will check how well it corresponds to the theoretical estimate (6.1).

The solid line in figures 9-13 corresponds to theoretical estimate (6.1) calibrated on 2.5, markers are experimental points. The experiment was the following: we were looking for such number of reference vectors M, at which the probability of error recognition P would be equal $1/M$ at fixed parameters N, n, q and a, i.e. solving the following equation numerically:

$$P(M, N, n, q, a) = \frac{1}{M}. \tag{7.1}$$

From the plots represented in Fig. 9-13 we can see that the estimate (6.1) is consistent with the experiment quite well. Resulting curves verify the correctness of conclusions drawed at the end of section 6. It is especially worth noting that the storage capacity of DLVP increases linearly with *n*, while as the storage capacity of a single-layer perceptron decreases with ln(*n*) (see (6.2)).

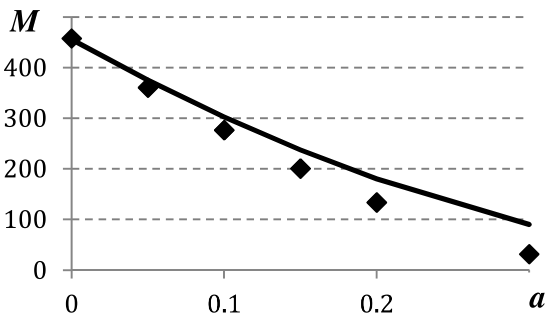


Fig. 9 DLP storage capacity *M* as a function of distortion level *a*. *N=100, q=50, n=4, P_{max}=1/M*

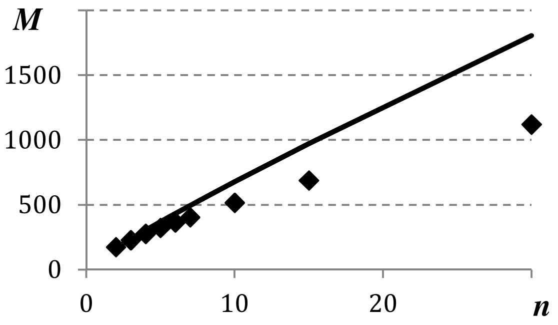


Fig. 10 DLP storage capacity *M* as a function of the number of vector neurons of an inner layer *n*. *N=100, q=50, a=0.1, P_{max}= 1/M*

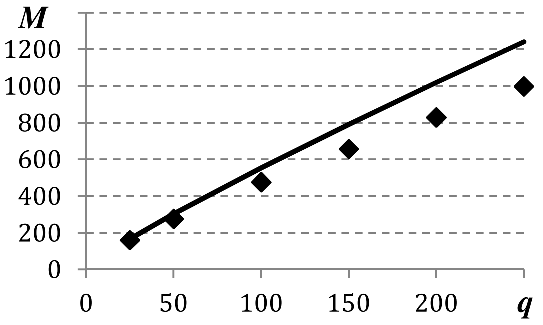


Fig. 11 DLP storage capacity *M* as a function of *q. N=100, a=0.1, n=4, P_{max}=1/M*

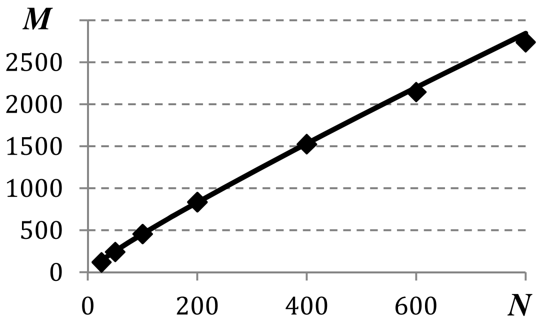


Fig. 12 DLP storage capacity *M* as a function of problem size *N*. *q=50, a=0.1, n=4,* $P_{max}=1/M$

7.4 K-Nearest Neighbors Search Task

The algorithm has yet another useful property, which a single-layer perceptron does not have. If we arrange patterns in decreasing order according to the components of their local field **H** (table 2, column “sum”), the order will tell us how close a pattern is to an input vector, while a pattern in the first place being regarded as the response of the system.

Let us demonstrate this by experiment. We will independently generate *M* random uncorrelated patterns, and additionaly another 5 patterns that are similar to each other to different extents (so they are correlated). The algorithm to generate these patterns is the following:

1) Generate random vector X_1;
2) Obtain vector X_2 by random distortion of 10% components of vector X_1;
3) Obtain vector X_3 by random distortion of 20% components of vector X_1;
4) Obtain vector X_4 by random distortion of 30% components of vector X_1;
5) Obtain vector X_5 by random distortion of 40% components of vector X_1;

Then, we will apply vector X_1 to the network inputs, and will monitor the values of the components of the local field **H**. Components of the local field corresponding to these 5 patterns will be greater than those corresponding to other components. At that, the maximum value of the local field will correspond to vector X_1 (since this vector was applied to the inputs). The second largest value will be the component corresponding to X_2, etc.

And indeed, the results of the experiment that are presented in Fig. 14 demonstrate it perfectly. Fig. 14 shows us distributions of the first six components of the local field **H** after applying vector X_1 to the inputs of the network. From this plot we can see that the spikes of the distributions are put in ascending order of patterns proximity to vector X_1.

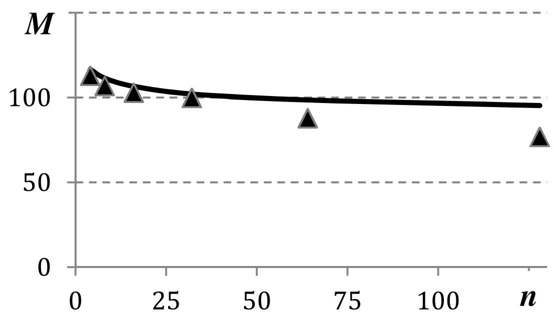


Fig. 13 Single-layer perceptron storage capacity M as a function of n. $N=100$, $q=50$, $a=0.1$, $P_{max}=1/M$. Solid line corresponds to the estimate (6.2), triangular markers corresponds to experimental points.

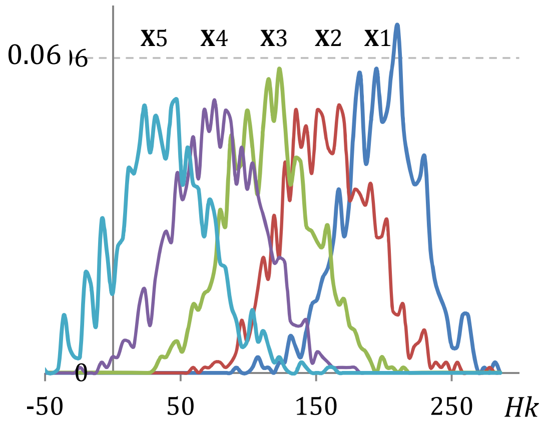


Fig. 14 Distributions of the first six components of the local field **H** after applying vector X_1 to the inputs of the network. $N=100$, $q=200$, $a=0$, $n=2$.

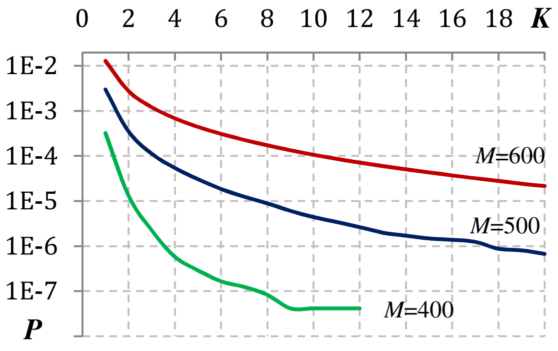


Fig. 15 Recognition error probability P as a function of the number of scalar products K. $N=100$, $n=2$, $q=100$, $a=0$, $M=400, 500, 600$

Such property allows us to solve the problem of K nearest neighbors search, which involves finding K reference patterns that are most similar to the input vector using Hamming distance. Alternatively, we can use this property to enhance the reliability of recognition for the problem of finding the first closest neighbor, i.e. for our task. To do this, we need to choose K reference patterns with the largest corresponding components of the local field **H**. Then, we need to calculate scalar products of an input vector with these reference patterns, and choose the winner (it has the maximum scalar product). The result is that it becomes possible to significantly reduce the probability of recognition error at the cost of a couple of additional scalar products.

Fig. 15 shows us a very high efficiency of this improvement. We see that calculation of two additional scalar products (K=2), for example, results in decrease of recognition error P by about an order of magnitude, and at K=20 – by three orders. The gain is more, the smaller the error probabilty is in the first place.

8 Conclusion

The paper shows that it is possible to raise the efficiency of the single-layer vector perceptron by adding an extra layer. The remarkable efficiency of the algorithm is demonstrated. It is clearly shown that in contrast to a straight increase of network redundancy, purposeful construction of neural nets can give nice results.

The research is supported by projects ONIT RAN 1.8 and 2.1.

References

1. Wu, F.Y.: The Potts model. Review of Modern Physics 54, 235–268 (1982)
2. Kanter, I.: Potts-glass models of neural networks. Physical Review A 37(7), 2739–2742 (1988)
3. Kryzhanovsky, B., Mikaelyan, A.: On the Recognition Ability of a Neural Network on Neurons with Parametric Transformation of Frequencies. Doklady Mathematics 65(2), 286–288 (2002)
4. Austin, J., Turner, A., Lees, K.: Chemical Structure Matching Using Correlation Matrix Memories. In: International Conference on Artificial Neural Networks, IEE Conference Publication 470, Edinburgh, UK, September 7-10. IEE, London (1999)
5. Kryzhanovsky, V.M.: Ph.D. Thesis, Research into Binary-Synaptic-Coefficient Vector Neural Nets for Data Processing and Decision Making Problems, System Research Institute of the Russian Academy of Sci-ences (2010)
6. Kryzhanovskiy, V., Zhelavskaya, I., Fonarev, A.: Vector Perceptron Learning Algorithm Using Linear Programming. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part II. LNCS, vol. 7553, pp. 197–204. Springer, Heidelberg (2012)
7. Podolak, I.T., Biel, S., Bobrowski, M.: Hierarchical classifier. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 591–598. Springer, Heidelberg (2006)
8. Podolak, I.T.: Hierarchical classifier with overlapping class groups. Expert Systems with Applications 34(1), 673–682 (2008)

Appendix 1

Now, we will present a nonstrict analytical derivation of DLVP storage capacity in the engineer style, in which we will neglect different correlations and make some simplifications. Only in this case, it becomes possible to deduce the resulting expression but not another unsolvable theorem. As it was shown above (in subsection 7.3), the final estimate (6.1) is in a good agreement with experiments regardless of introduced simplifications.

If we take a particular set of reference patterns and a DLVP trained on this set, the result of the recognition of particular pattern **X** will be deterministic and nonprobabilistic (note that the process of training DLVP is also deterministic). At such approach one cannot speak of the error recognition probability and moreover, reason about storage capacity. However, let us try another approach.

Assume we have 1000 DLVPs trained on different sets of reference patterns. Let us apply first patterns $\mathbf{X}_1$ from each set to the inputs of DLVPs accordingly. Let us consider their local fields **H**. We will denote k-th component of these fields as H_k. In the case of correct recognition, first components of the local fields H_1 should be greater than other components H_i, $i = 2, 3, \ldots, M$. Then, 1000 of the first and the second components of the local fields **H** can be considered as realizations of two random variables H_1 and H_2 (as you understand, the number of DLVPs can be any, and 1000 is just an example). So, since H_k are random variables, (M-1) inequalities

$$\begin{cases} H_1 > H_2 \\ H_1 > H_3 \\ \vdots \\ H_1 > H_M \end{cases} \tag{A.1}$$

will hold with some probability 1–P, where

$$P = 1 - \Pr\left[\bigcap_{k=2}^{M} H_1 > H_k\right] \tag{A.2}$$

is a probability of error recognition. Thus, if we can analytically estimate a functional relationship between P and all the model parameters, which are reference patterns size N, number of reference patterns in the training set M, internal parameters n and q, and the noise level a, then we will be able to find model reliability at specific parameters values. This will be our first goal.

1. Error Probability

Let us neglect some aspects, which will help us to obtain an expression for P:

1. Random values H_i are dependent and correlate with each other, but we will assume that they are independent.
2. Events (H_1>H_2), (H_1>H_3), …, (H_1>H_M) are also dependent (since they all depend on H_1), but we will assume they are not.

It is obvious that each such approximation results in discrepancies between theory and experiment. But this is a price we have to pay. Eventually, we can make an approximate estimate of (A.2) in the following way:

$$P = 1 - \prod_{k=2}^{M} \Pr\left[H_1 > H_k\right]. \qquad \text{(A.3)}$$

Now, we can focus on each random variable H_k separately, to evaluate its distribution function and its statistical characteristics.

2. Distribution Function of H_k

To evaluate distribution of H_k one need to substitute (3.2) and (3.3) in (3.4):

$$\mathbf{H} = \sum_{m}^{M} \mathbf{o}_m \sum_{\mu}^{M} \sum_{i}^{N} \sum_{j}^{n} (\mathbf{y}_j^m, \mathbf{y}_j^\mu) x_i^\mu \tilde{x}_i^1 , \qquad \text{(A.4)}$$

where index "1" and tilde in the last multiplier $\tilde{x}_i^1$ emphasizes that the first reference pattern ($\mathbf{X}_1$) was applied to DLVP inputs (see (3.3), where x_i denotes i-th component of an input vector, which is $\mathbf{X}_1$ in our case), and this pattern had aN of its components distorted, $0<a<0.5$. Recall that $\mathbf{o}_m$ is an M-dimensional unit vector containing 1 at m-th position. Subject to the last note, m-th component of vector $\mathbf{H}$ will be

$$H_{\text{m}} = \sum_{\mu}^{M} \sum_{i}^{N} \sum_{j}^{n} (\mathbf{y}_j^{\text{m}}, \mathbf{y}_j^\mu) x_i^\mu \tilde{x}_i^1 \qquad \text{(A.5)}$$

We can see from (A.5) that H_m is basically a sum of a great number of «+1» и «-1», which can take only integer values, and therefore, its distribution is discrete. We can approximate it with the normal distribution with a reasonable accuracy. Next, we need to evaluate expected means and variances of random variables H_m.

It is worth going through the approach we conduct our analysis at one more time. We consider the multipliers in expression (A.5) as random variables. Their realizations correspond to particular sets of reference patterns and a particular input vector. So, for instance, we consider x_i^μ, which is i-th component of reference pattern $\mathbf{X}_\mu$, as a random variable, which can take either +1 or -1 equiprobably (expected mean of this value is 0, and standard deviation is 1).

3. Statistical Properties of H_k

Looking ahead, it is worth considering random variable H_1 separately from the rest of the components H_2, H_3, …, H_M, since their statistical properties are different.

To estimate the expected mean and the variance of H_1, we need to extract additives having μ=1 from (A.5):

$$H_1=\sum_i^N\sum_j^n(\mathbf{y}_j^1,\mathbf{y}_j^1)x_i^1\tilde{x}_i^1+\sum_{\mu\neq1}^M\sum_i^N\sum_j^n(\mathbf{y}_j^1,\mathbf{y}_j^\mu)x_i^\mu\tilde{x}_i^1 \qquad \text{(A.6)}$$

Taking into account the way variables $\mathbf{y}_j^\mu$ were defined in 3.1, we get the following:

$$(\mathbf{y}_j^1,\mathbf{y}_j^1)\equiv1\,. \qquad \text{(A.7)}$$

Considering that input vector $\mathbf{X}$ is basically vector $\mathbf{X}_1$ with aN distorted components, we get the following as well:

$$\sum_i^N x_i^1\tilde{x}_i^1\equiv(1-2a)N \qquad \text{(A.8)}$$

The first sum in (A.6) is equal strictly to (1-2*a*)*nN*, so (A.6) becomes:

$$H_1=(1-2a)nN+\sum_{\mu\neq1}^M\sum_i^N\sum_j^n(\mathbf{y}_j^1,\mathbf{y}_j^\mu)x_i^\mu\tilde{x}_i^1 \qquad \text{(A.9)}$$

By signal we shall call the first part of (A.9), and by noise – the second part. Let us consider multipliers of the second sum: $(\mathbf{y}_j^1,\mathbf{y}_j^\mu)$, x_i^μ and $\tilde{x}_i^1$. They are independent, so

$$E\left[(\mathbf{y}_j^1,\mathbf{y}_j^\mu)x_i^\mu\tilde{x}_i^1\right]=E\left[(\mathbf{y}_j^1,\mathbf{y}_j^\mu)\right]E\left[x_i^\mu\right]E\left[\tilde{x}_i^1\right]=0\,, \qquad \text{(A.10)}$$

where *E*[] is expectation operator. Thus, the noise term has zero mean, but has a very large variance. The noise term may turn out to be larger than the signal $(1-2a)nN$ due to statistical outliers, which will cause the error in recognition.

Ultimately, we can estimate the mean and the variance of H_1 as:

$$\begin{aligned}E\left[H_1\right]&=(1-2a)nN\\ D\left[H_1\right]&=\frac{nNM}{q}\end{aligned}\,. \qquad \text{(A.11)}$$

Random variables H_2, H_3, …, H_M variables cannot be divided into the signal and the noise terms, since they have only noise terms. These variables have same statistical characteristics:

$$\begin{aligned}E\left[H_k\right]&=0\\ D\left[H_k\right]&=\frac{nNM}{q}\left(1+\frac{(n-1)q}{M}+\frac{N-1}{M}\right)\end{aligned}\cdot \qquad \text{(A.12)}$$

Estimates of variances in (A.11) and (A.12) were obtained similarly to the mean estimate in (A.10). It is necessary to take into account distribution functions of random variables $(\mathbf{y}_j^1, \mathbf{y}_j^\mu)$ and x_i^μ when deriving the estimates of mathematical expectations and variances:

$$(\mathbf{y}_j^1, \mathbf{y}_j^\mu) = \begin{cases} -1, \textit{ with a probability of } \dfrac{1}{2q} \\ +1, \textit{ with a probability of } \dfrac{1}{2q} \\ 0, \textit{ otherwise} \end{cases} \tag{A.13}$$

and

$$x_i^\mu = \begin{cases} -1, \textit{ with a probability of } \dfrac{1}{2} \\ +1, \textit{ with a probability of } \dfrac{1}{2} \end{cases} \tag{A.14}$$

4. Error Probability Estimate *P*

Let us introduce a new random variable combining both random variables H_1 and H_k:

$$\Delta_k = H_1 - H_k, \; k = \overline{2, M} \tag{A.15}$$

The assumption made above that variables H_k are independent allows us to easily calculate statistical characteristics of Δ_k:

$$\begin{aligned} E[\Delta_k] &= (1-2a)nN \\ D[\Delta_k] &\approx \frac{nNM}{q}\left(2 + \frac{nq}{M} + \frac{N}{M}\right) \end{aligned} \tag{A.16}$$

Variable Δ_k is normally distributed, so the probability of event $\Delta_k > 0$ (see multipliers in (A.3)) is defined by the following expression:

$$1 - P_k = \Pr[\Delta_k > 0] = \frac{1}{\sqrt{2\pi D[\Delta_k]}} \int_0^\infty \exp\left(-\frac{(\xi - E[\Delta_k])^2}{2D[\Delta_k]}\right) d\xi \tag{A.17}$$

We can write the expression of error recognition probability as

$$P = 1 - (1 - P_k)^{M-1}. \tag{A.18}$$

Since we are interested in the case when error probability is $P \to 0$ (so the neural network works very reliably), we can make the following estimate of (A.17):

$$P \approx MP_k . \tag{A.19}$$

Thus, we need to estimate probability P_k. There is a range of expansions for the error function

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt \tag{A.20}$$

These expansions allow us to calculate probability P_k approximately. So, let us express (A.16) in terms of error function:

$$P_k = \frac{1}{2} erfc(\gamma), \tag{A.21}$$

where

$$\gamma^2 = \frac{E^2[\Delta_k]}{2D[\Delta_k]} = \frac{nqN(1-2a)^2}{2M}\Theta, \qquad \Theta = \left(2 + \frac{nq}{M} + \frac{N}{M}\right)^{-1}. \tag{A.22}$$

The greater the value of γ^2 is, the smaller the error probability P is. Therefore, the approximation of (A.17) by (A.18) is made when $\gamma^2 >> 1$.

Let us take the first additive from the well-known error function expansion

$$erfc(x) = \frac{e^{-x^2}}{x\sqrt{\pi}}\left[1 + \sum_{r=1}^{\infty}(-1)^r \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{\left(2x^2\right)^r}\right] \tag{A.23}$$

By doing so we get the resulting error probability expression:

$$P = \frac{M}{2\gamma\sqrt{\pi}} e^{-\gamma^2} \tag{A.24}$$

The estimate (A.23) describes the model in a qualitative manner. However, it is inconsistent with the experiment by several orders. Such significant difference is coming from the point that error probability is in exponential relationship with the parameters of the model:

$$P \sim e^{-\gamma^2} \tag{A.25}$$

Even minor errors in calculations of this exponential factor lead to significant deviations from the experiment, since $\gamma^2 >> 1$. Authors realized that introduced approximations and assumptions would lead to directly that. However, though expression (A.23) is of interest, but it is not the final goal of our derivations. Based on it, we will get a consistent estimate of storage capacity.

5. Storage Capacity

According to the definition of the storage capacity of associative memory $M_{\max}$ given in section 6, it is such number of reference patterns, recognizing which the probability of error P is not greater than a predefined threshold $P_{\max,}$ and input vectors are distorted at the noise level $a \geq 0$. Therefore, we need to solve the following equation for M:

$$P\left(M, N, n, q, a\right) \leq P_{\max}. \tag{A.26}$$

Let us take a logarithm of the right and the left-hand side of (A.25):

$$\gamma^2 \geq \ln \frac{M}{2\gamma P_{\max}\sqrt{\pi}} \tag{A.27}$$

Then, let us substitute γ^2 in this expression:

$$M \leq \frac{nqN(1-2a)^2}{\ln\left(\frac{M^3}{2\pi\left(nqN(1-2a)^2\Theta\right)P_{\max}^2}\right)}\Theta \tag{A.28}$$

Variable M in (A.27) is both in the right and the left-hand sides of the formula. Therefore, let us use the following trick: we will recurrently insert (A.27) into itself:

$$M \leq \frac{nqN(1-2a)^2}{2\ln\left(\frac{nqN(1-2a)^2\Theta}{P_{\max}\sqrt{2\pi}\ln^{3/2}(\ldots)}\right)}\Theta \tag{A.29}$$

There still left multipliers Θ depending on M in the right-hand side of (A.28). Let us try to eliminate this dependency. If we lower the estimate of storage capacity M we will only strengthen inequality (A.25). Therefore, let us give a raw lower estimate of Θ:

$$\Theta \approx \frac{1}{4} \tag{A.30}$$

Eventually, we get the final expression for the estimate of DLVP storage capacity:

$$M < C\frac{nqN(1-2a)^2}{8\ln\left(\frac{nqN}{4\sqrt{2\pi}P_{\max}}\right)}, \; C = 2.5\,. \tag{A.31}$$

According to multiple experiments, (A.30) gives a good qualitative description of the model, however, in order to be in a good agreement with the experiment a normalization constant C=2.5 is required.

Local Detection of Communities by Attractor Neural-Network Dynamics

Hiroshi Okamoto

Abstract. Community structure is a hallmark of a variety of real-world networks. Development of effective and efficient methods for detecting communities in networks has been a central issue of network science. Here we propose a method for detecting communities in networks. We have devised this method inspired by the cell assembly hypothesis, which has been one of the prevailing hypotheses in neuroscience. The cell assembly hypothesis states that neurons coding the same item tend to be mutually connected, thus forming a 'cell assembly'; memory recall of this item is associated with sustained activation of neurons belonging to the cell assembly. Here we compare communities to cell assemblies and examine community detection by use of the neural-network dynamics describing memory recall in the brain. To demonstrate the effectiveness of the proposed method, local detection of communities in synthetic benchmark networks and real social networks is examined. The community structure detected by our method is highly consistent with the correct community structure of these networks.

Keywords: Complex network, Community detection, Cell assembly, Memory recall, Neural-network dynamics, Attractor.

1 Introduction

In literature of network science, 'community' refers to a group of nodes that are more densely connected within this group than with nodes outside this group. Community structure is a fundamental property of a variety of real-world networks. Development of effective and efficient algorithms to detect communities in networks has been a big challenge of network science [1, 2].

Hiroshi Okamoto
Research & Development Group, Fuji Xerox Co., Ltd.,
6-1 Minatomirai, Nishi-ku, Yokohama-shi, Kanagawa 220-8668, Japan, and
RIKEN Brain Science Institute,
2-1 Hirosawa, Wako, Saitama 351-0198, Japan
e-mail: hiroshi.okamoto@fujixerox.co.jp

Community detection algorithms proposed up to now mainly aim to exhaustively detect all the communities in a given network [1, 2]. However, this type of community detection is computationally infeasible for extremely large or dynamically evolving networks such as the World Wide Web, the Twitter network or the Facebook network.

Another, more economic approach is to detect only a community to which a given source node belongs [3-6]. Starting from a given source node, one explores the network crawling links; exploration will continue until certain criteria are met; the explored region, or a part of it, is then judged as the community to which the source node belongs. This type of community detection is described as 'local' because it is intended to find the community to which a given source node belongs without knowing all other communities in the network. The purpose of this study is to propose and examine a method for local detection of communities, which we devised inspired by the longstanding and prevailing hypothesis in neuroscience, the 'cell assembly hypothesis'.

Neurophysiological studies have revealed that short-term memory recall of a certain item in the brain is associated with sustained activation of neurons coding this item [7, 8]. The cell assembly hypothesis states that neurons coding the same item tend to be mutually connected [9]. Thus, neurons coding the same item organize a densely connected group, which is referred to as a 'cell assembly' [9]. Reverberatory propagation of neuronal activities localized in the cell assembly generates a sustained activation of neurons belonging to this cell assembly [10].

Moreover, a lot of memory items are stored in the brain. Different items are associated with different cell assemblies. Therefore, the neural-network dynamics describing memory recall in the brain should have a multitude of stable states [11, 12]. Initial activation of a fraction of neurons, which serves as a cue, determines which stable state (attractor) will be selected. These describe neural mechanisms underlying cue-triggered memory recall of an item.

Cell assemblies can be compared to communities in real-world networks. This will inspire the idea of local detection of communities by use of the neural-network dynamics describing cue-triggered memory recall in the brain [13]. To demonstrate the effectiveness of this idea, we examine local detection of communities in synthetic benchmark networks and real social networks for which correct community structure is known. This paper is an extended version of a previously published article [14] and a part of data presented here is identical to that presented there.

2 Methods

2.1 Neural-Network Dynamics

Consider a network of N nodes with undirected links. Let $\mathbf{A} = (A_{nm})$ $(n, m = 1, \cdots, N)$ be the adjacency matrix of this network. If nodes n and m are connected, $A_{nm} = A_{mn} = 1$; otherwise $A_{nm} = A_{mn} = 0$.

Now we compare individual nodes to neurons and individual links to synaptic connections between neurons. Let $p_n(t)$ and $f_n(t)$ be the "potential" and the "activity" of neuron n at time t, respectively. We assume that the relationship between $p_n(t)$ and $f_n(t)$ is given by a threshold-linear function (Fig. 1), which models the relationship between the membrane potential and the firing rate of pyramidal cells [15]:

$$f_n(t) = \sigma\left(p_n(t) - \theta\right) p_n(t), \tag{1}$$

where $\sigma(x) = 1\Big/\left\{1 + \exp\left[-\beta\left(p_n(t) - \theta\right)\right]\right\}$ with $\beta \geq 0$ and $\theta \geq 0$. Note that as x increases, $y = f(x)$ sigmoidally rises and then asymptotically approaches $y = x$ (Fig. 1).

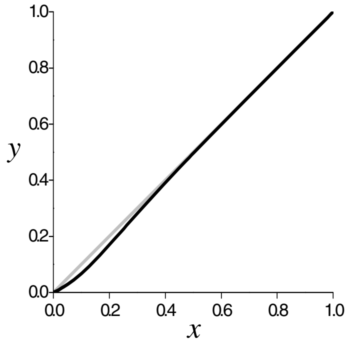


Fig. 1 As x increases, $y = f(x) = \sigma(x)x$ (black line) sigmoidally rises and then asymptotically approaches $y = x$ (grey line). We have chosen $\beta = 10$ and $\theta = 1/34$ for this illustration

Time evolution of potentials of individual neurons is described by the equation

$$p_n(t) = \sum\nolimits_{m=1}^{N} T_{nm} f_m(t-1) + \frac{f_n(t-1)}{F(t-1)}\left[F_0 - F(t-1)\right] \qquad \left(F_0 > 0\right) \tag{2}$$

where $T_{nm} \equiv A_{nm}\Big/\sum\nolimits_{n'=1}^{N} A_{n'm}$ and $F(t) \equiv \sum\nolimits_{n=1}^{N} f_n(t)$. The first term on the right-hand side describes propagation of activities to neuron n from neurons making synaptic connections onto neuron n. The second term models competition between neurons for a finite resource $F_0 - F$; such competition, which is generally considered to occur owing to activation of inhibitory interneurons, is common in cortical network architecture [16].

We can easily verify that the sum of the potential of all neurons is kept constant with time

$$\sum_{n=1}^{N} p_n(t) = F_0 . \tag{3}$$

This property is important as it stabilizes the neural-network dynamics (2). By virtue of this property, the network is kept from falling into pathological states such as flare-up or extinction of the activities of all neurons. Without loss of generality we set $F_0 = 1$.

One can give another interpretation to equation (3). In the limit $\theta \to 0$ and $\beta \to \infty$, equation (3) becomes

$$p_n(t) = \sum_{m=1}^{N} T_{nm} p_m(t-1) . \tag{4}$$

This is formally equivalent to the Markov-chain equation [17, 18]. One can suppose a 'random walker' who is walking around the network along links; $p_n(t)$ is then interpreted as the probability that the random walker stays at node n at time t and T_{nm} is the transition probability from node m to node n (note that by its definition T_{nm} satisfies $\sum_n T_{nm} = 1$). For $\theta > 0$ and finite β, however, $p_n(t)$ can no longer be interpreted as the probability for a single random walker. Instead one can suppose a large number of random walkers; $p_n(t)$ can now be interpreted as the population density of node n at time t. A random walker staying at node m at time $t-1$ monitors the population density of this node. With probability $r = 1/\left(1+\exp\left(-\beta\left(p_m(t-1)-\theta\right)\right)\right)$ he/she selects one of the nodes linked from node m and then moves to the selected node at time t; with probability $1-r$ the random walker jumps to any node. For the latter, to which node he/she jumps is determined by the relative amplitude of the activity; namely, he/she jumps to node n with probability $f_n(t-1)/F(t-1)$.

2.2 *Local Detection of Communities*

For a given source node (say, node l), local community detection is done as follows. First the initial condition is set as follows

$$p_l(0) = 1, \quad p_n(0) = 0 \quad (n \neq l) . \tag{5}$$

This means that neuronal activities are initially concentrated at the source node. As time passes, activities spread in the network according to equation (3). Activities preferentially propagate within the community to be detected, where nodes are more densely connected within this community than with nodes outside. Potentials of nodes located around the frontier region of the community are considered small because these nodes have less links than those centrally located in the community. Owing to the sigmoidal rise of $f(x)$ at small x, activities of nodes around the frontier regions rapidly decay. Thus activities no more spread far beyond the frontier regions. Thus, activities are confined and localized within the community.

Iterative calculation of equation (3) with initial condition (5) eventually leads to the steady-state distribution of potentials $\left\{p_n^{(\mathrm{stead})}\right\}$ $(n=1,\cdots,N)$. We consider that this steady-state distribution represents the community to which the source node belongs. The $p_n^{(\mathrm{stead})}$ has a graded value ranging from 0 to 1, which expresses the level of belongingness of node n to the detected community.

Our algorithm detects communities as attractors of the neural-network dynamic. Arenas et al. [19, 20] have proposed to detect communities in networks by coupled-oscillator dynamics. In their model communities emerge as temporally evolving synchronization patterns, whereas communities detected by the proposed method are represented by static patterns (point attractors).

2.3 Parameters

The neural-network dynamics defined by equation (4) includes two parameters, β and θ. We found that the value of β controls the resolution of the community structure to be detected (see **Results**). We have empirically found that community detection by the proposed method is highly performed when the value of θ is chosen as

$$\theta = 1/N_{\mathrm{explored}}\ , \tag{6}$$

where N_{explored} is the number of neurons whose potential has ever been elevated above zero during course of neuronal propagation in the network. Note therefore that N_{explored} is a function of time t and never decreases with time; that is, $N_{\mathrm{explored}}(t) \geq N_{\mathrm{explored}}(t-1)$.

3 Results

3.1 Local Detection of Communities from Synthesized Networks

First we evaluate the effectiveness of the proposed method using synthetic benchmark networks. Lancichinetti et al. have proposed a method for synthesizing networks with community structure that captures essential features of that of real-world networks [21, 22]. The number of communities and their sizes can be controlled by adjusting the parameter values. In this study we have synthesized a network of $N=200$ nodes with six communities. The parameter values used for synthesizing this network and the statistics of the communities are given in Appendix.

One of the 200 nodes was chosen to define the initial condition by (5) and then the steady state distribution of potentials $\left\{p_n^{(\mathrm{stead})}\right\}$ $(n=1,\cdots,N)$ was obtained by iterative calculation of equation (2). The same calculation was repeated for all nodes. For each node taken as a source, the steady state distribution falls into any of K patterns, with K varying with β as shown below. We consider that each pattern represents the community to which the source node belongs.

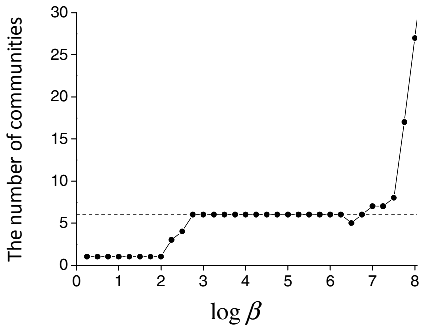


Fig. 2 The number of communities detected in the synthesized benchmark network by the proposed method depends on β. Note that for a wide range of β the correct number of communities (six, indicated by broken line) is stably obtained.

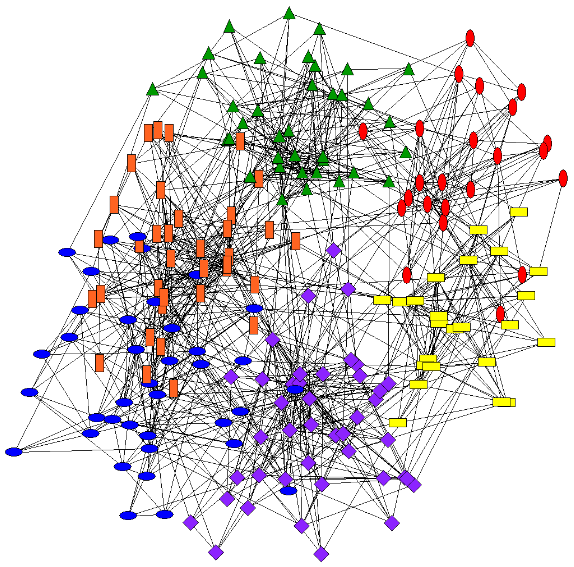


Fig. 3 Local detection of communities in the synthetic benchmark network. The color (red, blue, green, yellow, orange or purple) of each icon indicates the community detected by the proposed method for this node taken as a source. The shape (vertically long ellipse, horizontally long ellipse, vertically long box, horizontally long box, triangle or diamond) of each node indicates the correct community to which this node belongs. Note that nodes that have the same shape also have the same color. Here, we have chosen $\log\beta = 5$.

The number of communities κ depends on β (Fig. 2). For a wide range of β the correct number (six) of communities is obtained. The detected community for each node taken as a source and the correct community to which this node belongs are compared in Fig. 3, which shows that local detection of communities in the synthesized benchmark network by the proposed method is perfect.

3.2 Local Detection of Communities from Real Social Networks

Zachary's karate-club network [23] is a famous benchmark network that has been used for development of algorithms detecting social sub-groups. This is a real social network of 34 members of karate club at a US university observed by a social scientist, Wayne Zachary. Each link represents social tie between two members. During the period of observation by Zachary, a dispute concerning management of membership dues had developed between the head teacher and the administrator. This finally resulted in factional separation of the club into two groups with one led by the head teacher and the other led by the administrator. The goal of Zachary's karate-club task is to predict to which group each member belongs after the factional separation of the club.

The number of communities detected by the proposed method depends on β (Fig.4). For a range of β ($\sim 1 \leq \log \beta \leq \sim 2$) the correct number (two) of communities is gained. For a value of β in this range, the detected community for each node taken as a source and the correct group to which the member corresponding to this node belongs are compared in Fig. 5, which shows that the proposed method accurately predicts the groups to which individual members belong after factional separation of the club.

Moreover, varying the value of β revealed hierarchical structure of communities in the karate club (Fig. 6 and 7). For $\log \beta = 0.0$ the network as a whole is

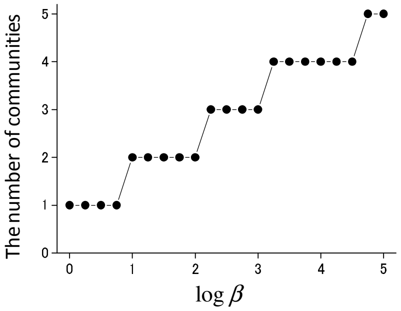


Fig. 4 The number of communities detected by our algorithm depends on β. Note that for a certain range of β the correct number of communities (two, indicated by broken line) is obtained.

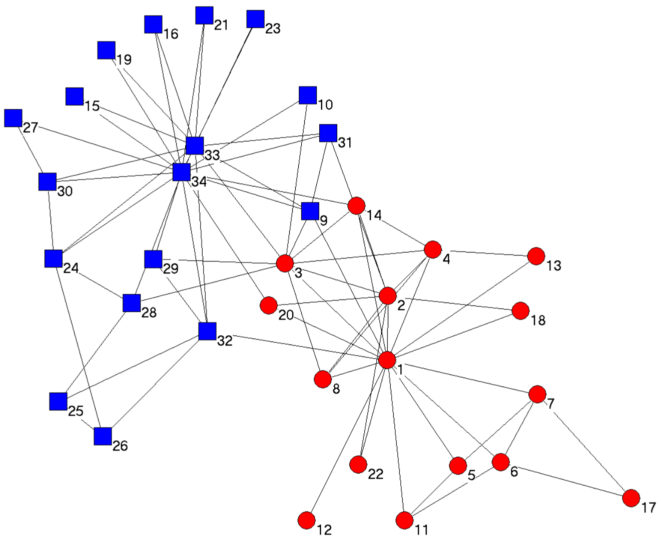


Fig. 5 Local detection of communities in Zachary's karate-club network for $\log \beta = 2.0$. The color (red or blue) of each node indicates the community detected by the proposed method for this node taken as a source. The shape (circle or box) of each node indicates the correct group to which the member corresponding to this node belongs. Note that nodes that have the same shape also have the same color.

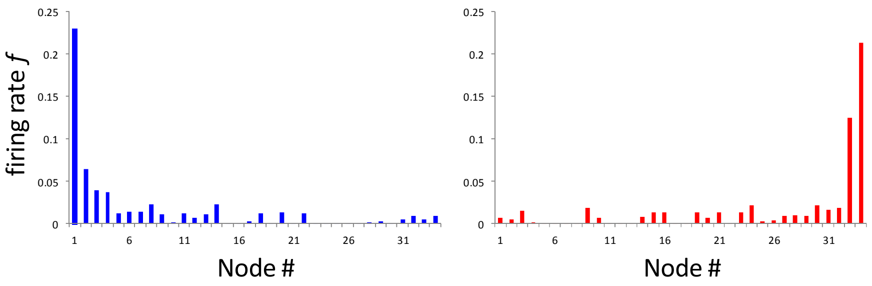


Fig. 6 For $\log \beta = 2.0$, the steady-state distribution of the firing rate $\{f_n\}$ $(n = 1, \cdots, N)$ for any node taken as a source falls into either of the two patterns (left or right). Nodes that lead to the left pattern and those that lead to the right pattern are indicated by blue and red icons, respectively, in Fig. 5.

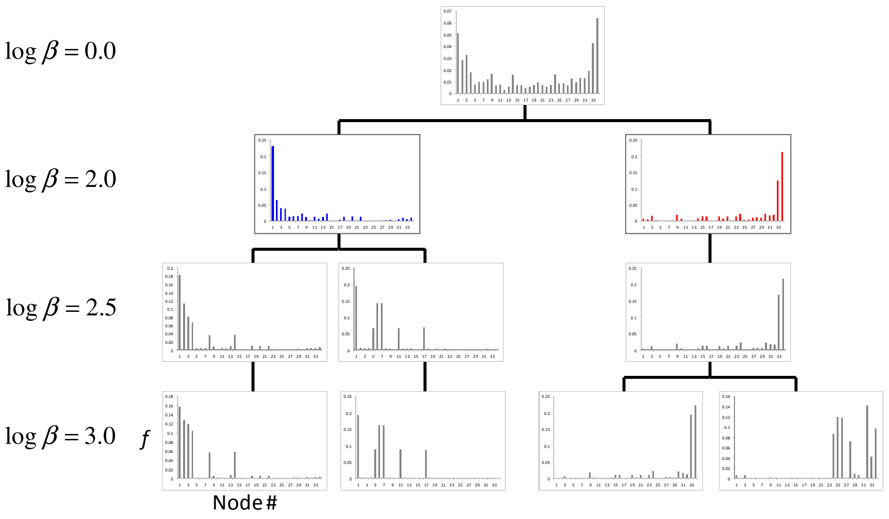


Fig. 7 Hierarchical structure of communities in Zachary's karate-club network

judged as a single community because the steady-state distribution of $\{f_n(t)\}\,(n=1,\cdots,N)$ is the same for any node taken as a source. As $\log\beta$ is increased to 2.0, the steady-state distribution falls into either of the two patterns shown in Fig. 6. The community corresponding to one of these patterns is separated into two groups when $\log\beta$ is further increased to 2.5. The remaining community is also separated into two groups when $\log\beta$ is further increased to 3.0. Thus we find hierarchical structure of communities of Zachary's karate-club network, as shown in Fig. 7.

4 Discussion

We have proposed a method for local detection of communities in networks. We have devised this method inspired by possible neural mechanisms of cue-triggered memory recall in the brain. Communities are compared to cell assemblies and source nodes to fraction of neurons whose activation serves as cue signals; cue-triggered activation of neurons belonging to a particular cell assembly is therefore compared to detection of a community to which a source node belongs. We have shown that the proposed method can correctly detect communities from a synthesized benchmark network (Fig. 3). Application of this algorithm to a real social network, the Zachary karate club network, has accurately replicated the factional separation of this club (Fig. 5). These results demonstrate the effectiveness of local detection of communities by the proposed method.

The computational complexity of local detection of communities by previously proposed algorithms is $\sim O\left(N_e^3\right)$ or $\sim O\left(N_e^2 < k >\right)$ [3-6], where N_e is the number of nodes in the explored portion of the network and $< k >$ is the mean number of links attached to individual nodes. The cost of computation by our algorithm, on the other hand, basically scales with the number of links L within the explored region. Connection density in real-world networks is generally sparse ($L_e << N_e^2$). Thus the proposed method is considered computationally more efficient than previously proposed ones. To confirm this, more elaborated study will be conducted in our forthcoming study.

The algorithm has parameters β and θ, and the number of detected communities depends on the values of these parameters (Fig. 2 and Fig. 4). In this study we have determined these values rather in a heuristic way, though we have found the value of β controls the resolution of communities (Fig. 7). Introduction of some measure to estimate the goodness of local detection of communities, such as the 'modularity' in global detection of communities [24], might be useful for developing more principled ways of determining the values of these parameters.

Networks examined in the present study are restricted to those having undirected links. Local detection of communities in networks having directed links is an important issue to be addressed in the next step of our study. For this, introduction of random jump from sink nodes to any other nodes, the prescription used for calculation of the PageRank values in the World Wide Web [18], might be useful.

Acknowledgments. This study was partly supported by KAKENHI (23500379) and KAKENHI (23300061).

References

1. Fortunato, S.: Community detection in graphs. Phys. Rep. 486, 75–174 (2010)
2. Newman, M.E.J.: Communities, modules and large-scale structure in networks. Nature Phys. 8, 25–31 (2012)
3. Bagrow, J.P., Bollt, E.M.: Local method for detecting communities. Phys. Rev. E 72, 046108 (2005)
4. Clauset, A.: Finding local community structure in networks. Phys. Rev. E 72, 026132 (2005)
5. Lancichinetti, A., Fortunato, S., Kertesz, J.: Detecting the overlapping and hierarchical community structure in complex networks. New J. Phys. 11, 033015 (2009)
6. Chen, Q., Wu, T.-T., Fang, M.: Detecting local community structures in complex networks based on local degree central nodes. Physica A 392, 529–537 (2013)
7. Funahashi, S., Bruce, C.J., Goldman-Rakic, P.S.: Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. J. Neurphysiol. 61, 331–349 (1989)
8. Churchland, A.K., Kiani, R., Shadlen, M.N.: Decision making with multiple alternatives. Nature Neurosci. 11, 693–702 (2008)
9. Hebb, D.O.: Organization of behaviour. Wiley, New York (1949)

10. Durstewitz, D., Seamans, J.K., Sejnowski, T.J.: Neurocomputational model of working memory. Nature Neurosci. Suppl. 3, 1184–1191 (2000)
11. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA 79, 2554–2558 (1982)
12. Wang, X.-J.: Neural dynamics and circuit mechanisms of decision-making. Curr. Opin. Neurobiol. 22, 1–8 (2012)
13. Okamoto, H.: Local detection of communities by an analogy to memory recall in the brain. Biol. Insp. Cog. Arch. 6, 12–17 (2013)
14. Okamoto, H.: Local Detection of Communities by Neural-Network Dynamics. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 50–57. Springer, Heidelberg (2013)
15. Tuckwell, H.: Introduction to theoretical neurobiology: nonlinear and stochastic theories, vol. 2. Cambridge University Press (1988)
16. Rabinovich, N.I., Volkovskii, A., Lecanda, P., Heurta, R., Abarbanel, H.D., Laurent, G.: Dynamical encoding by networks of competing neuron groups: winnerless competition. Phys. Rev. Lett. 87, 068102 (2001)
17. Collins, A.M., Loftus, E.F.: Spreading-Activation Theory of Semantic Processing. Psychological Review 82, 407–428 (1975)
18. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies Project (1998), `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`
19. Arenas, A., Diaz-Guilera, A., Perez-Vicente, C.J.: Synchronization reveals topological scales in complex networks. Phys. Rev. Lett. 96, 114102 (2006)
20. Arenas, A., Diaz-Guilera, A., Perez-Vicente, C.J.: Synchronization processes in complex networks. Physica D 224, 27–34 (2006)
21. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms Phys. Rev. E 78, 46110 (2008)
22. `http://santo.fortunato.googlepages.com/benchmark.tgz`
23. Zachary, W.W.: An information flow model for conflict and fission in small groups. J. Anthropol. Res. 33, 291–473 (1977)
24. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004)

Appendix: Synthetic Benchmark Network

The benchmark network examined in **3.1** was synthesized using the software downloaded from [22], under the following settings: Number of nodes 200; average degree 10; maximum degree 30; exponent for the degree distribution 2; exponent for the community size distribution 1; mixing parameter 0.2; minimum for the community sizes 20; maximum for the community sizes 50. The synthesized network has six communities having 23, 24, 36, 37, 38 and 42 nodes.

Learning Gestalt Formations for Oscillator Networks

Martin Meier, Robert Haschke, and Helge J. Ritter

Abstract. The binding of similar objects to a common group is an effortless task for humans. We know if things belong together or not by intuitively relying on a set of rules. In the area of visual perception, these rules can be described by the *Laws of Gestalt*. Although these laws are intuitive for humans to understand, a computational feasible formulation can be demanding. We present an improved approach to learn the computational formulations for these laws from labeled training data. The approach learns attraction and repelling interactions between features, which in turn can be used in artificial neural networks to decided whether input features belong to a common group or have to be separated. The technique is evaluated within different perceptual grouping scenarios and with two kinds of artificial neural networks.

1 Introduction

The perception of sensory stimuli is a complex problem that incorporates many facets in different levels of abstraction, both in human and artificial systems. For example, the task of reading a word from this page reaches from the activation of single nervous cells in the fovea, to a higher level where stimuli are bound together based on these activations to continuous areas, facilitating figure ground segmentation between dark letters and the page. Based on their proximity, letters which are not separated by white spaces are grouped together to form a word. Finally, the words which are not separated by punctuation marks form sentences. To this end, the initially perceived low level stimuli are processed in a hierarchy of increasing abstraction that requires the binding of features from the visual and syntactic level, leading to a semantic meaning in the end. During this process, the concepts of binding and segregation play an important role on all of these levels. This principle can be found in various modalities, e.g. the auditory systems employs a similar

Martin Meier · Robert Haschke · Helge J. Ritter
Bielefeld University, 33501 Bielefeld, Germany
e-mail: {mmeier,rhaschke,helge}@techfak.uni-bielefeld.de

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_7

process for the understanding of spoken language. Recent studies also suggest that these principles apply to haptic perception (Chang et al, 2007; Gallace and Spence, 2011).

Although this principle can be found in different perceptional modalities, the most prominent domain of research is the area of visual perception. In the early 19^{th} century, the *Laws of Gestalt* emerged as a branch of cognitive psychology. Actually being more descriptive than formal, these laws illustrate basic rules how humans perceive and process different kinds of visual stimuli. Some of the most prominent laws are shown in Fig. 1, an overview can be found in Wagemans et al (2012).

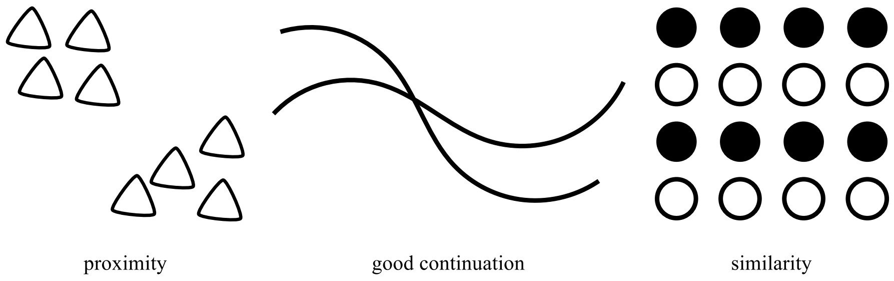


Fig. 1 Illustration of some of the most prominent *Laws of Gestalt*. Through proximity, the triangles in the leftmost panel form two distinct clusters. The two curved lines are distinguished based on their continuation and by similarity, the circles in the right panel are grouped into horizontal lines.

When looking at the illustration, the reader will intuitively recognize the described law and the respective groups. The proximity of features in the leftmost panel is recognized as two clusters, the intersecting lines are easy to distinguish from each other because of their smooth continuation and the circles in the rightmost panel form horizontal lines according to their similarity. From a computational perspective, this grouping task is far more demanding. One of the main problems that arise is how to generate "good" groups, which is closely related to the Binding Problem (Treisman et al, 1996). The Binding Problem expresses the decision, if features from an input domain have to be segregated or combined to form a meaningful figure based on their properties and the interaction between these properties. On the other hand, appropriate architectures which utilize the binding rules and generate a meaningful result from a set of features coupled by said rules are also required.

Neurophysiological research on the visual system of animals created the foundation for a type of artificial networks which are able to handle these grouping problems. By measuring the neural activity in the visual cortex while presenting bar patterns to cats, Gray and Singer (1989) discovered that spiking neurons in areas related to the stimulus synchronize whilst other areas do not synchronize. The work of Gray and Singer (1989) led to the development of *LEGION* (locally excitatory globally inhibitory oscillator networks) by Wang (1994), who used relaxation oscillators which are arranged in a two dimensional grid. An oscillator in this grid is coupled to its four neighbors with an excitatory connection that enables a synchrony of oscillators. Additionally, all oscillators are coupled with a global inhibitory component which facilitates desynchronization. When input stimuli are presented to this network, this combination of excitatory and inhibitory connections achieves a synchronization in oscillator spikes. An example where the LEGION network is applied to artificial letters is shown in Wang and Terman (1995).

In terms of the Binding Problem, the LEGION network employs a positive binding between features when they are in close proximity. This binding becomes negative through the global inhibitory component for features that are far away from each other.

Since Wang introduced the concept of combining excitatory and inhibitory connections in oscillator networks, this principle has been employed in a wide variety of networks. For example Li and Li (2011) modified the LEGION model by replacing the global inhibitor by local inhibitory shortcuts between oscillators. This method increases the synchronization speed of the network compared to the original LEGION. It was evaluated in an image segmentation task, where pixels form real world images were connected based on their proximity. Yu and Slotine (2009) employed FitzHugh-Nagumo oscillators to solve clustering, contour integration and image segmentation tasks. For each of these tasks, problem specific connection rules were developed. Oscillators from the network which synchronized their phases based on these connection rules represent the salient features from the input domain, for example similar regions in gray scaled images. Nomura et al (2011) also utilized FitzHugh-Nagumo oscillators for edge detection in gray scaled images. They arranged the oscillators in two grids, one activator and one inhibitor grid, to account for the two variables of these type of oscillator. Similar to the LEGION model, an oscillator here is coupled to its four neighbors. The coupling strength between neighboring oscillators is based on the intensity of corresponding pixels from the image. To obtain an edge map of the input image, the approach is applied two times, on the original and the inverted gray scale image. The final edge map is created as a union of both of these intermediate maps. By exploiting the synchrony in a network of Rössler oscillators, Breve et al (2009) were able to identify salient object in real work images. They employed a similar approach as Nomura et al (2011) by choosing individual couplings between oscillators based on the relative contrast in the image.

Most of the mentioned oscillator networks still employ task specific, hand crafted rules for each grouping problem. Due to the design of these networks, the rules have to express the compatibility, or interaction, between features in a scalar value, which can introduce additional complexity in terms of weighting and mapping from n dimensional feature vectors to a one dimensional interaction value. Wersing (2001) presented a quadratic consistency optimization (QCO) learning algorithm for interaction functions based on the topology of the Competitive Layer Model (CLM) (Ritter, 1990). The concept was later picked up by Weng et al (2006). The approach in Weng et al (2006) is to express the interaction function as a set of basis functions, which in turn are learned from labeled examples and simple n-dimensional distance vectors. The basis functions in this case are estimated with a vector quantization variant, the Activity Equilibrium Vector Quantization (AEV) (Heidemann and Ritter, 2001) in the space of the distance vectors. The attracting respectively repelling interaction weights are obtained by incorporating the labels from the examples. Although this approach achieves good results for different perceptual grouping problems, it does not incorporate the mutual information of the elements in the proximity space, leading to a sub optimal approximation of the interaction function. To gain a better approximation, the learning algorithm was recently improved (Meier et al, 2013a) by employing a vector quantization variant which explicitly incorporates information theoretic principles, the ITVQ algorithm (Rao et al, 2007).

Before we cover the ITVQ based learning algorithm, we will introduce two networks for perceptual grouping tasks in the following section. First the CLM, for which the original learning algorithm was developed and later on a network composed of Kuramoto oscillators which was presented in Meier et al (2013b) and can achieve grouping results of similar quality more time efficient while keeping the high grouping quality of the CLM. Building on the topological properties of the CLM, the original learning algorithm is outlined in section 3 and our proposed modifications, including an abstract of the ITVQ algorithm, are stated. Following the theoretical part, we evaluated the presented techniques within different perceptual grouping scenarios that resemble basic laws of Gestalt as illustrated in Fig. 1.

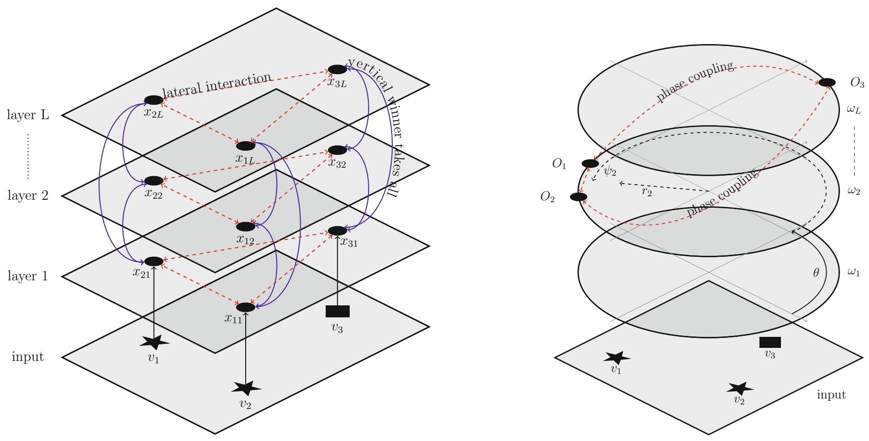
2 Artificial Networks for Perceptual Grouping

The topological layout of the CLM and the oscillator network is designed to allow an easy evaluation of the grouping results. In case of the CLM, the dynamics is designed such, that similar features are grouped in the same layer. This is possible because of the columnar arrangement of neurons. A *winner takes all* circuit per column assures that only one neuron per column can be active at a time. Therefore the grouping result automatically manifests itself by the active neurons per layer.

The network presented in Meier et al (2013b) employs a similar topology, but transfers the concept of layers to discrete frequencies in the phase space of Kuramoto oscillators (Kuramoto, 2003). This section covers details about the dynamics of both networks, followed by an analysis of the grouping behavior with artificial data.

2.1 Competitive Layer Model

The CLM consists of $N \times L$ neurons which are arranged in L layers. Neurons are indexed column wise with $r = 1,\dots,N$ denoting the feature index within each layer and $\alpha = 1,\dots,L$ denoting the layer index. A single neuron's activity is therefore denoted as $x_{r\alpha}$. A graphical representation of the CLM is shown in Fig. 2a. The neurons in each layer are coupled with a symmetric interaction function $f(v_r, v_{r'}) = f(v_{r'}, v_r) = f_{rr'}$ which describes the compatibility between two features v_r and $v_{r'}$. They are additionally coupled with a *winner takes all* (WTA) circuit in each column to assure that only one neuron in each column becomes active. Therefore, a single



(a) CLM **(b)** Kuramoto Oscillators

Fig. 2 Graphical representation of the two networks described in this article. The Competitive Layer Model in Fig. 2a employs a recurrent dynamic to drive neural activity, which in turn represents similar features from an input domain, to be in the same layer α. The oscillator network in Fig. 2b has a similar topology, but uses a frequency based grouping in oscillator phase space. The circles in the figure represent the state of the oscillators in the phase space in polar coordinates. The angle represents the phase θ_r and the radius the frequency ω_r of an oscillator O_r.

input feature v_r is represented by a column composed of L neurons $x_{r\alpha}$. Combining the lateral interaction and columnar WTA circuit, the recurrent CLM dynamics can be written as:

$$\dot{x}_{r\alpha} = -x_{r\alpha} + \sigma(J(h_r - \sum_{\beta=1}^{L} x_{r\beta}) + \sum_{r'=1}^{N} f_{rr'} x_{r'\alpha}) \; . \tag{1}$$

Here $J(h_r - \sum_\beta x_{r\beta})$ represents the WTA competition weighted by the constant J, h_r encodes the importance of feature v_r – which is set to 1 for each feature in the upcoming evaluation, because all features are equally important – and $\sigma(\cdot)$ is a linear threshold function. The lateral interaction is expressed as $\sum_{r'} f_{rr'} x_{r'\alpha}$, which calculates the support for the feature r from all other features r' in a given layer α. For a more comprehensive overview and a prove of the CLM convergence, we refer to (Wersing et al, 2001).

2.2 *Coupled Kuramoto Oscillators*

The oscillator model has a topology similar to the CLM, but replaces the L neurons in each column with a single oscillator of the Kuramoto type (Kuramoto, 2003), where an oscillator O_r is described by its phase θ_r and frequency ω_r. Additionally to the global coupling constant K, these oscillators are coupled individually by a symmetric matrix $M_{rr'} \equiv f(v_r, v_{r'})$. Thus, the phases θ_r of the oscillators evolve according to the following update rule:

$$\dot{\theta}_r = \omega_r + \frac{K}{N} \sum_{r'=1}^{N} f(v_r, v_{r'}) \cdot \sin(\theta_{r'} - \theta_r). \tag{2}$$

The interaction function f is limited to the interval $[-1, 1]$, where -1 and $+1$ represent strongest dissimilarity resp. similarity of features. Arenas et al (2006) introduced a correlation measure

$$\rho_{rr'}(t) = \langle \cos(\theta_r(t) - \theta_{r'}(t)) \rangle$$

between oscillator phases to trace the evolution of clusters in real world data like social networks over time. Related to this approach, an autocorrelation based technique was recently employed by Bassett et al (2013). By measuring the correlation between phases, these approaches also had to introduce a threshold above which the correlated oscillators are interpreted as a group. In turn, the setting of a threshold requires either domain knowledge or the introduction of additional heuristics.

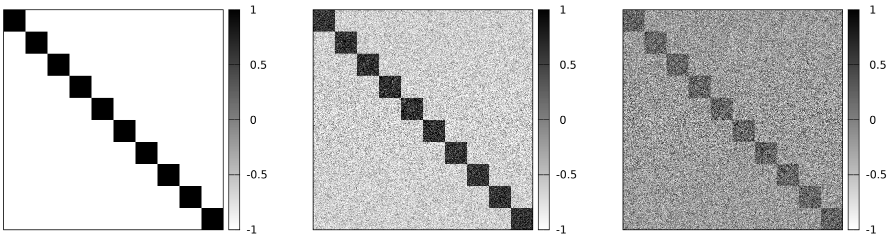
To relax the need for threshold based evaluations of grouping results, we introduced a topology into the oscillator network based on fixed frequencies. These oscillator frequencies are limited to discrete values $\omega_\alpha = \alpha \cdot \omega_0$, where $\alpha \in \{1, \ldots, L\}$ denotes the group index – following the CLM notation where α denotes the group/layer index. This discretization will further allow a very simple analysis of the grouping result. To cluster similar features to the same frequency ω_α, the frequency ω_r of each oscillator is updated according to the support $S_r(\alpha)$ an oscillator receives from the subset of oscillators currently occupying the discrete frequency $\omega_0 \cdot \alpha$. Similar to correlation based approaches, we employ the cosine similarity between the phases of the oscillators. This similarity measure is mapped to the interval $[0, 1]$, which is crucial to preserve the sign of the interaction function $f(v_r, v_{r'})$. Therefore, the frequencies are updated according to:

$$\begin{aligned} S_r(\alpha) &= \sum_{r' \in \mathcal{N}(\alpha)} f(v_r, v_{r'}) \cdot \tfrac{1}{2}\big(\cos(\theta_{r'} - \theta_r) + 1\big) \\ \omega_r &= \omega_0 \cdot \operatorname*{argmax}_{\alpha}(S_r(\alpha)) \end{aligned} \tag{3}$$

where $\mathcal{N}(\alpha)$ denotes the set of oscillators with frequency index α, i.e. forming the current perceptual group indexed by α. This updates the frequency of an oscillator O_r to the frequency $\omega_0 \cdot \alpha$, whose corresponding oscillators provide most support in terms of f-weighted phase similarity. It ensures that oscillators representing similar features will both phase-lock and converge to identical frequencies. Eq. (3) also boosts the phase-locking process, because synchronized phases do not tend to desynchronize anymore. Contrarily, oscillators representing dissimilar features will spread both in phase and frequency. The final grouping result is determined by oscillator subsets $\mathcal{N}(\alpha)$ having common frequency indices α. In terms of the CLM topology, an oscillator O_r can represent a whole column of neurons, because it possesses two degrees of freedom. The phase coupling (2) represents the lateral interaction over all layers whilst the frequency update (3) acts as the layer assignment.

2.3 Grouping with Artificial Data

We evaluated the grouping quality of the oscillator network compared to the CLM in Meier et al (2013b) with artificial data. The evaluation utilized networks with 1000 features and 100 layers/discrete frequencies. The test data contained 10 target groups and the interaction function $f_{rr'}$ contained different amounts of noise in form of randomly inverted interaction values. An example for these function in matrix form is shown in Fig. 3. Here a black pixel indicates an attracting interaction value $f_{rr'} = 1$ and a white pixel expresses $f_{rr'} = -1$.



(a) 0% inverted. **(b)** 20% inverted. **(c)** 40% inverted.

Fig. 3 Visualization of interaction functions with different amounts of inverted interaction values. Black pixels represent attracting interactions whilst white pixels indicate repelling interactions.

To investigate the behavior in more demanding scenarios than a simple, noise free grouping task as visualized in Fig. 3a, we successively added noise to the interaction matrices in form in randomly inverted interaction values while keeping the symmetry. The amount of noise is increased in steps of 1% from zero to a total of 49% noise in the interaction matrices. For each percentage, we simulated 500 trials with both networks. A single trial consisted of 1000 update steps and to account for different computational demands of both networks, a step is the update of each oscillator or neuron, respectively. After each trial, we calculated the grouping quality Q as proposed in Wersing et al (2001):

$$Q = \frac{1}{N^2}\sum_{r}^{N}\sum_{r'}^{N} q_{rr'}, \quad q_{rr'} = \begin{cases} 1\ if & t_r = t_{r'} \text{ and } a_r = a_{r'} \\ 1\ if & t_r \neq t_{r'} \text{ and } a_r \neq a_{r'} \\ 0\ else & \end{cases} \tag{4}$$

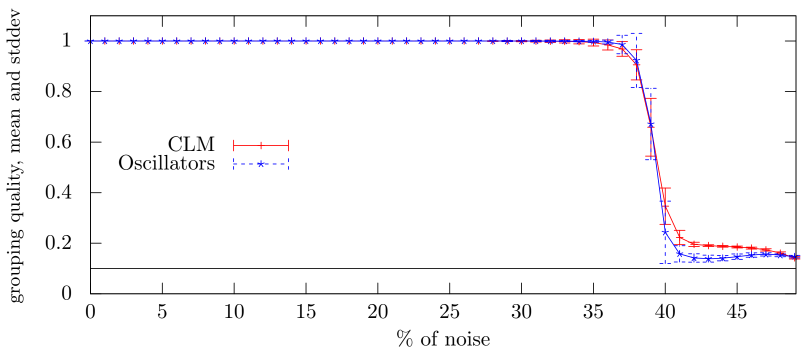


Fig. 4 Achieved grouping quality for both networks with respect to an increasing percentage of inverted interaction values. Both networks show nearly perfect grouping results for up to 35% of noise and decrease similar after that mark.

where t is the target label and a the assignment generated by the perceptual grouping network. This yields a value from 0 to 1, where 1 is a perfect grouping result.

The results for both networks in terms of grouping quality are shown in Fig. 4. The CLM and the oscillator network achieve similar results with a nearly perfect grouping quality for up to 35% of inverted interaction values. For percentages greater than 35%, the performance of both networks decreases similar.

Based on the fast convergence properties of other approaches (Yu and Slotine, 2009; Li and Li, 2011), the amount of updates steps each network needs to achieve these results is also of interest. Therefore, we measured the number of updates with respect to noise to achieve at least a 95% correct grouping quality, the results are shown in Fig. 5. The oscillator network needs far less updates than the CLM for up to 30% of noise, showing a nearly constant value whilst the CLM increases linear with

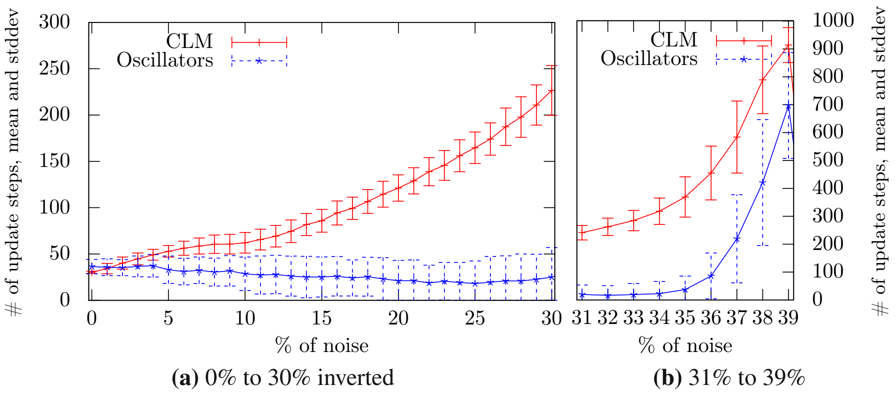


Fig. 5 Number of update steps needed to achieve a grouping quality of at least 95%

respect to noise. For greater amounts of noise, both networks show an exponential increase in required update steps. Based on these simulations, the oscillator network is a feasible alternative to the CLM, it maintains the same grouping quality but shows better convergence speed.

3 Learning Lateral Interactions

Handcrafting functions that represent attracting and repelling interactions can be difficult and error prone, resulting in reduced generalization abilities of these functions. To circumvent these problems, the concept in Weng et al (2006) is to learn these functions from labeled examples and to approximate the handcrafted interaction function by distance vectors $\mathbf{d}_{rr'} = (d_1(v_r, v_{r'}), \ldots, d_n(v_r, v_{r'}))$ in a $n-$dimensional proximity space D. This transformation assures that the required symmetry is preserved. Another important aspect of this distance vector is, that it relaxes the former requirement of hand crafted interaction functions to be one dimensional, as $f_{rr'}$ in Eq. (1). The set of distance vectors obtained from all training feature pairs is represented by a small set of representative prototypes by means of vector quantization.

The attracting respectively repelling interactions $f_{rr'}$ are created in a second step which incorporates the labels of the training data. To this end, the number of compatible and incompatible feature pairs from the training set is counted for each associated prototype. (Features are compatible if they share a common group.) These counts are finally used to determine the interaction weight c_i assigned to the prototype.

3.1 Original Learning Algorithm with AEV

The first formulation of the learning problem for interaction functions based on the layered CLM topology was presented in Wersing et al (2001). Starting from the desired grouping result of the CLM as depicted in Fig. 6, the target state of the CLM is expressed by a set of target vectors $\mathbf{y}_\alpha$. These target vectors in turn are generated by a set of labeled training examples $\mathbf{T} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$. A training example $\mathbf{y}_i$ is composed of a number of input features $\mathbf{y}_i = \{v_r^i, \ldots, v_{r'}^i\}$, where i indicates the group label, i.e. all features sharing the same label i belong to the same target group.

As proven in (Wersing et al, 2001), the CLM dynamics converges to a set of stable states which satisfy the consistency conditions

$$\sum_{r'} f_{rr'} x_{r'\beta} \leq \sum_{r'} f_{rr'} x_{r'\alpha(r)} \quad \forall r, \beta \neq \alpha(r) \tag{5}$$

Theses conditions express the assignment of a feature at position r to the layer $\alpha(r)$ and β represents all other possible layers.

Given a consistent labeling $\alpha(r)$ for a training sequence $\mathbf{T}$, the task is to learn an interaction function $f_{rr'}$ which fulfills the consistency conditions from (5) for all elements of $\mathbf{T}$. As proven in (Weng et al, 2006), these inequalities can be approximated by a matrix

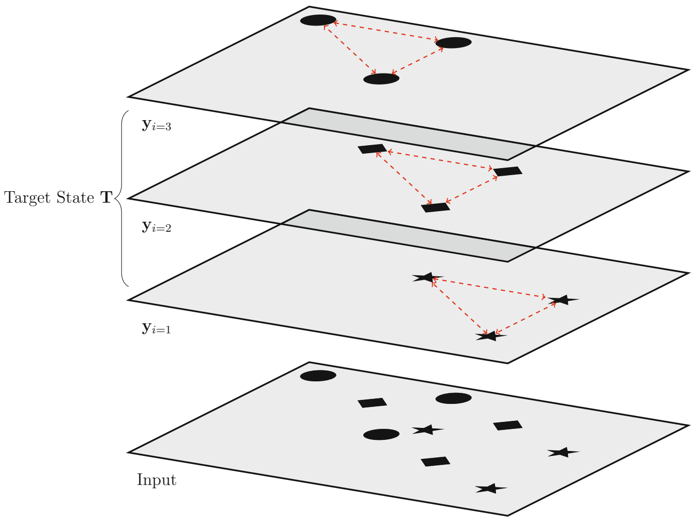


Fig. 6 Illustration of the learning problem. Given an input for the CLM, the interaction function $f_{rr'}$ should created neural activities based on the target state $\mathbf{T} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$, which is composed of the desired group labels $\mathbf{y}_i$, by combining attracting and repelling interactions between features.

$$\hat{F} = \hat{f}_{rr'} = \sum_{\gamma} \sum_{\mu \neq \gamma} (\mathbf{y}_\gamma - \mathbf{y}_\mu)(\mathbf{y}_\gamma - \mathbf{y}_\mu)^{\mathbf{T}} \tag{6}$$

where $\mathbf{y}_\gamma$ and $\mathbf{y}_\mu$ stand for all features in the γth resp. μth layer.

This interaction matrix only represents the discrete values which are present in the training sequence $\mathbf{T}$. To generalize it to a continuous function over the feature space, the interaction function is decomposed into a set of B symmetric basis interaction function $g^j_{rr'} = g^j(m_r, m_{r'})$. These basis interaction functions are a linear combination over the feature space in the way that

$$f_{rr'} = \sum_{j}^{B} c_j g^j_{rr'} \tag{7}$$

It is further described in Weng et al (2006), that when the the basis interaction functions are assumed to by binary step functions $g^j_{rr'} \in \{0, 1\}$, a disjunct partition of the feature space can be estimated through:

$$c_j = \sum_{r,r'} \hat{f}_{rr'} g^j_{rr'} \tag{8}$$

To preserve the required symmetricity of the interaction function, the feature space is transformed into a generalized proximity space $D = R^P$ where

$$\mathbf{d}_{rr'} = ((v_{r0} - v_{r'0})^2, \ldots, (v_{rP-1} - v_{r'P-1})^2) \tag{9}$$

and v_{ri} being the ith component of feature vector v_r.

Each of the so gained proximity vector $\mathbf{d}_{rr'}$ is then projected onto a multidimensional Voronoi map. This map consists of B cells and prototypes $\tilde{\mathbf{d}}_j$, where each cell is defined as

$$V_j = \left(v_r, v_{r'}\right) \mid \forall i \neq j : \| \mathbf{d}_{rr'} - \tilde{\mathbf{d}}_j \| \leq \| \mathbf{d}_{rr'} - \tilde{\mathbf{d}}_i \| \tag{10}$$

Because the Voronoi tessellation results in a disjunct partition, we can set $g^j_{rr'} = 1$ if $(v_r, v_{r'}) \in V_j$ and $g^j_{rr'} = 0$ otherwise, which fulfills the condition for Eq. (8).

It is now possible to learn an interaction function by calculating the proximity vector $\mathbf{d}_{rr'}$ for a pair of features $v_r, v_{r'}$ and search its nearest prototype vector $\tilde{\mathbf{d}}_j$ to get the corresponding coefficient c_j.

After the vector quantization phase, feature pairs are sampled randomly and their labels are examined. If a feature pair shares the same label, a positive interaction c_i^+ is counted for the closest prototype $\tilde{\mathbf{d}}_i$. In the case of different labels, the interaction counts as repelling and is therefore stored as negative interaction c_i^-. After a sufficient amount of random samples, the positive and negative interactions are summed up to create the interaction coefficient

$$c_i = c_i^+ - \lambda c_i^- \tag{11}$$

for each of the $i = 1, \cdots, B$ prototypes. Here λ is a weighting parameter to account for the fact, that typically more incompatible feature pairs exist. Finally, the coefficients are normalized by the coefficient with the biggest value.

The proposed vector quantization in Weng et al (2006) is Activity Equilibrium VQ (AEV) (Heidemann and Ritter, 2001) to estimate a set of $\tilde{\mathbf{d}}_i$, $i = 1, \ldots, N$ prototypes for the distance space D. In contrast to standard vector quantization, AEV measures the activity of each prototype based on the number of data points from the input it describes. If a prototype only represents a smaller subset of the input data than the remaining prototypes, it is repositioned in an exploration step, otherwise it is locally adapted. Combined with simulated annealing, this technique should avoid idle prototypes and place them uniformly distributed in regions of the feature space where input data is present.

3.2 Learning Algorithm with ITVQ

Although the original learning algorithm based on AEV already yields good results, the distribution of the prototypes has some drawbacks. It is desirable to find prototypes, which reflect the structure of the underlying grouping problem. This can be achieved by replacing the AEV clustering in the original algorithm with a vector quantization variant which explicitly incorporates the information density of the feature space, as presented by Rao et al (2007). The main idea there is to minimize the Cauchy-Schwartz (CS) divergence between the input data and the prototypes. The

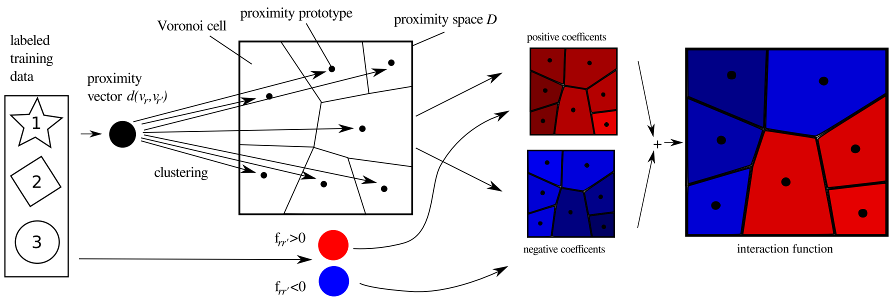


Fig. 7 Sketch of the learning algorithm. The labeled training data on the left is used twofold. Based on the distances between features, the proximity space is tessellated with vector quantization. The supplied labels are used to decide whether the corresponding proximity prototype represents a positive or negative interaction. If a prototype represents more features pairs which share the same label than pairs with different labels, it obtains a positive interaction value and a negative otherwise. This figure is adapted from Weng et al (2006).

CS divergence measures the “distance” between two probability density functions $p(x)$ and $q(x)$ as

$$D_{CS}(p,q) = -log\frac{\int p(x)q(x)dx}{\sqrt{\int p^2(x)dx \int q^2(x)dx}} \tag{12}$$

Following the derivation in Rao et al (2007) by using Renyi’s quadratic entropy (Rényi, 1976) for a dataset X

$$H(X) = -log\big(V(X)\big) = -log\Big(\int p^2(x)dx\Big) \tag{13}$$

and the cross entropy between two datasets X and X_0

$$H(X,X_0) = -log\Big(\int p(x)q(x)dx\Big), \tag{14}$$

the CS divergence can be estimated by

$$D_{CS} = 2H(X,X_0) - H(X) - H(X_0). \tag{15}$$

Given the datasets X_0, which represents the original data, and the set X of prototypes, the goal is to find the dataset X which minimize the cost function

$$J(X) = \min_X D_{CS}(X,X_0). \tag{16}$$

Differentiating $J(X)$ with respect to x_i and using the Parzen window technique for the dataset $X = (x_i), i = 1,\ldots,N$ with

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} G_{\sigma'}(x - x_i), \tag{17}$$

where $G_{\sigma'}(t) = e^{-\frac{t^2}{2\sigma'^2}}$ is a Gaussian kernel to estimate the pdfs of the input data, we get a simple fixed point update rule:

$$x_i^{t+1} = \frac{\sum_{j=1}^{N_0} G_\sigma(x_i^t - x_{0j})x_{0j}}{\sum_{j=1}^{N_0} G_\sigma(x_i^t - x_{0j})} - c\frac{\sum_{j=1}^{N} G_\sigma(x_i^t - x_j^t)x_j^t}{\sum_{j=1}^{N_0} G_\sigma(x_i^t - x_{0j})} + c\frac{\sum_{j=1}^{N} G_\sigma(x_i^t - x_j^t)}{\sum_{j=1}^{N_0} G_\sigma(x_i^t - x_{0j})}x_i^t \tag{18}$$

With $c = \frac{N_0}{N}\frac{V(X,X_0)}{V(X)}$. For a more comprehensive derivation please refer to Rao et al (2007).

Replacing the AEV algorithm with ITVQ for the learning of interaction prototypes while keeping the same estimation of interaction coefficients from Eq. (11) should lead to better perceptual grouping capabilities of the two networks described in section 2. We will evaluate the proposed changes within different scenarios and compare the grouping quality to the original approach in the following section.

4 Applications of Learned Gestalt Formations

Applying the theoretical principles, we will show examples of learned Gestalt Formations according to the Laws of Gestalt illustrated in Fig. 1. Starting from the rather simple to implement proximity, the examples grow in complexity to emphasis the need for a learning architecture.

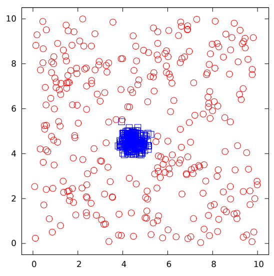
4.1 Proximity

The behavior of grouping objects which are, mostly in a spatial sense, close to each other is described by the Gestalt Law of proximity. Formulating this law by hand for computational purposes can be done in terms of the Euclidean distance between features as:

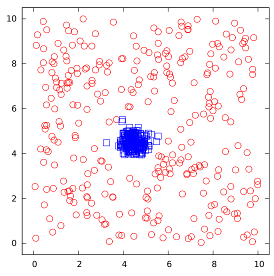
$$d(\mathbf{v_r}, \mathbf{v_{r'}}) = \sqrt{\sum_{i=1}^{n}(v_{r,i} - v_{r',i})^2}$$

Difficulties arise when it comes to define thresholds for the borders of classes, especially when multiple classes with different distributions are interleaving. Fig. 8a shows such a problem, where a dense cluster of Gaussian distributed features (blue rectangles in the figure) is embedded in a set of sparse uniform distributed features (red circles). In this case, two regions of distances are needed which represent features that belong together whilst the remaining distances represent no group relation.

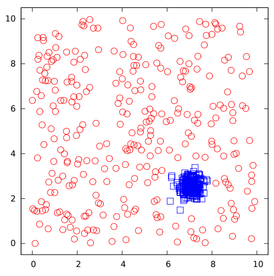
An example for a hand crafted proximity rule can be seen in Yu and Slotine (2009). They calculate the coupling strength between oscillators based on the Euclidean distance with an exponential decay. Also a threshold M in terms of the num-



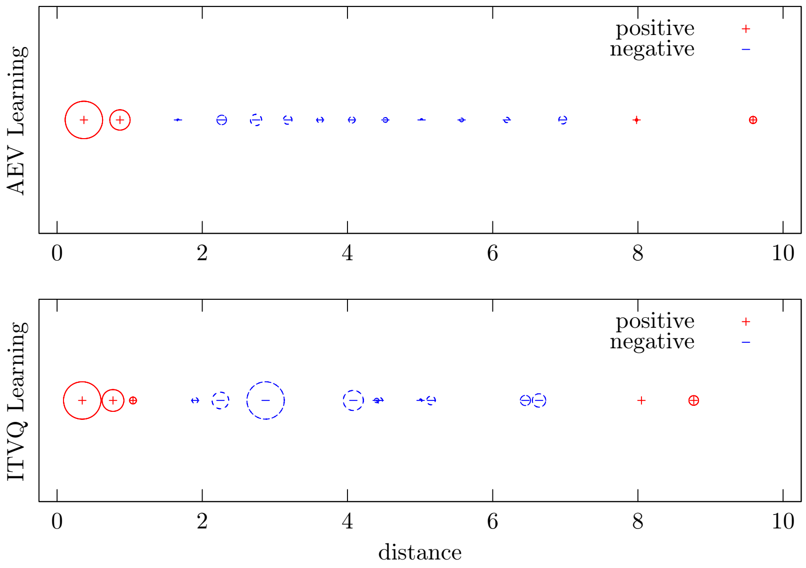
(a) Training input.



(b) Grouping of Fig. 8a.



(c) Grouping of a test input.



(d) Learned proximity prototypes.

Fig. 8 Training and test input for the learning of a proximity law based on the Euclidean distance between features. Fig. 8a shows the training data which consists of two classes. The blue rectangles represent 200 features with a Gaussian distribution ($\sigma = 0.4$). They are embedded in a sparse uniform distribution, represented by red circles. Fig. 8b shows the grouping result for the training input and Fig. 8c for another randomly generated test. The learned prototypes with AEV and ITVQ learning are shown in Fig. 8d. They contain two classes of positive connections, one for features in close proximity, representing the distances between the blue rectangles and one for features further away from each other.

ber of nearest neighbors is applied, such that features further away as the M closest features have a coupling value of zero.

By employing a data driven approach to learn the interaction function, the need to hand tune parameters can be removed. We used the learning approach from Sec. 3 with the Euclidean distance between features to learn an interaction function from the data shown in Fig. 8a. The data consists of 500 features, 200 of them are drawn from a Gaussian distribution with $\sigma = 0.4$ at a randomly chosen center, these features are displayed with blue rectangles. The remaining 300 features (indicated by red circles) are distributed uniform. For the example in Fig. 8 an interaction function with 15 prototypes is learned from the training input shown in Fig. 8a. The grouping of the input with the learned interaction function and ITVQ learning is shown in Fig. 8b, the grouping results for a test input with the same interaction function is shown in Fig. 8c. The learned one dimensional prototypes are visualized in Fig. 8d. Positive prototypes are indicated with a red $+$ and negative prototypes with a blue $-$. The radius of the circles indicated the weight of the prototype. Both algorithms generate two groups of positive prototypes separated by negative weighted prototypes. Noticeable is the different weighting between both variants. In case of the AEV learning, the negative prototypes are weighted similar, whilst the ITVQ learning created a large negative prototype close to the border to the positive prototypes for small distances, which indicates a stronger separation between these two classes.

To investigate the influence of different amounts of prototypes, we generated interaction functions with $5, 10, 15$ and 20 prototypes that are evaluated with both networks and learning variants. For each condition 100 trials were performed. The average grouping quality over these trials is shown in Fig. 9. For small numbers of prototypes, the ITVQ learning achieves slightly better results of 8%. With an increasing number of prototypes, the grouping quality of the ITVQ version is around 10% higher than the AEV learning over all cases. Also the oscillator model performs slightly better than the CLM in this task, but only about 3% over all cases.

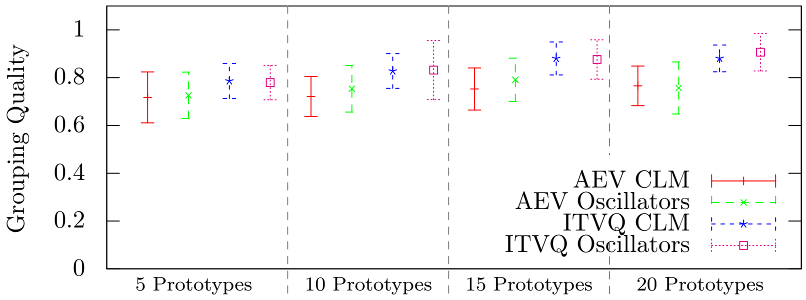


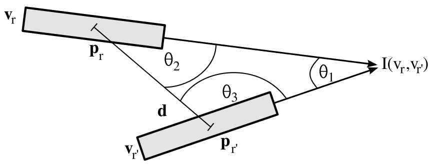
Fig. 9 Evaluation of the learned proximity interaction function with both learning variants and networks for $5, 10, 15$ and 20 prototypes. The ITVQ version of the learning algorithm achieves a 8% higher quality for the CLM case compared to AEV with only 5 prototypes and 14% for the oscillator case with 20 prototypes.

4.2 Learning Good Continuations

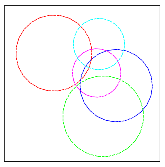
In Meier et al (2013a), we resembled the original contour grouping task from Weng et al (2006) as evaluation scenario for the improved learning of interaction functions. In this task, an interaction function is learned for oriented edge features. The distance between two oriented edges

$$d(v_r, v_{r'}) = (||p_r - p_{r'}||, \theta_1, \theta_2, \theta_3)^T$$

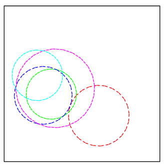
is defined by their Euclidean distance and the three angles θ_{1-3}, as shown in Fig. 10a. For three types of shapes, namely triangles, squares and circles, interaction functions are learned with AEV and ITVQ clustering for different numbers of prototypes. According to the findings from Weng et al (2006), the λ parameter from Eq. (11) is set to 2 in all trials.



(a) Oriented edge features.



(b) "Easy" problem.



(c) "Hard" problem.

Fig. 10 A sketch of the parameter of the distance function is shown in 10a. Figures 10b and 10c show examples for an easy grouping task and a hard one, respectively. In the hard example, the overlapping lines in the center and on the left are nearly indistinguishable.

In Fig. 11, 100 prototypes learned with ITVQ and AEV are shown for circular shapes composed of oriented edge features (as depicted in Fig. 10). Although the prototypes can only be evaluated qualitatively by visual inspection, the positive interaction prototypes in the zoomed areas suggest, that the ITVQ variant is more sensitive to different radii in the input data by generating more prototypes for these cases.

For the quantitative part of the evaluation, we varied the number of prototypes in steps of 50 starting with 50 for up to 200 prototypes. After learning an interaction function from eight shapes with different orientations, positions and sizes, each function was used in 200 trials to group randomly generated inputs, each input consisting of five shapes with varying sizes, positions and orientations. Because the oscillator network is less computationally demanding, we employ the same measure for counting update steps as before: A single step is the update of each neuron in each layer for the CLM or the update of each oscillator for the oscillator network, respectively. Based on the previous findings, each trial was limited to a total of 500 steps. The grouping quality Q is calculated according to Eq. 4.

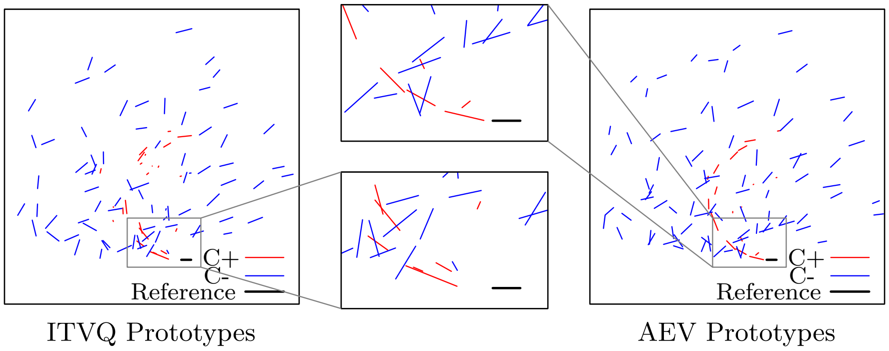


Fig. 11 Example of learned prototypes for circular training data. Red edges show attracting and blue edges represent repelling prototypes. The length of the edges encodes the interaction strength. All prototypes are positioned relative to the black feature. Although the prototypes are hard to inspect visually, ITVQ generates more positive prototypes which represent different radii of the input data. This can be seen in the zoomed part of the images. (best viewed in color)

The results for this evaluation are shown in Fig. 12. Especially for a small number of prototypes, the presented modification which uses ITVQ outperforms the CLM with AEV learning by 27%. This difference decreases with an increasing number of prototypes, but the grouping is still 12% better with 200 prototypes. Also, the oscillator network achieves a better grouping with both methods, which is especially significant for the case with AEV learning. The variance of the grouping results can be explained by the random generation of the shapes. Fig. 10b shows an example of an easy grouping task, while Fig. 10c is way more demanding because of largely overlapping features. Following up to the findings of Meier et al (2013b), we also calculated the average number of update steps for the CLM and the oscillator network which are needed to achieve the maximal grouping quality in each trail. The CLM takes an average of 88.7 steps to gain the maximal grouping quality while the oscillator network reaches this goal within an average of 24.9 steps.

4.3 Similarity of Textures

As an example for the learning of interaction functions which resemble the law of similarity, we employ a texture grouping scenario on real world images that we first presented in Meier et al (2014). The applied interaction function was originally developed for the CLM without the learning technique and presented in Ontrup et al (2004). The image preprocessing is based on two dimensional Gabor Filters with different sizes and orientations, which are applied on image patches. The dimensionality of the Gabor Filters is reduced using standard principal component analysis and combined with the patch location in image coordinates.

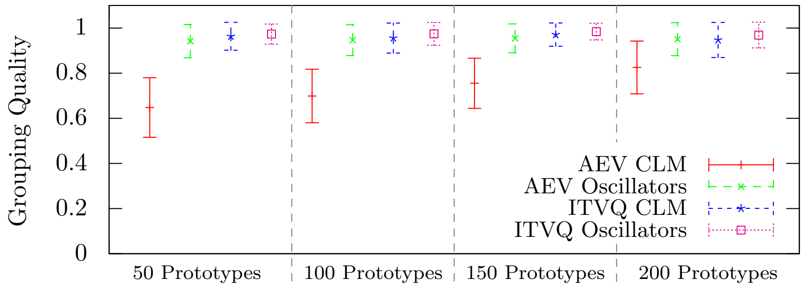


Fig. 12 This plot shows the mean grouping quality Q with standard deviation for each combination of learning algorithm and perceptual grouping network, each over 200 trials. The value is the average over all three types of shapes.

A Gabor Filter $g(\cdot)$ is complex valued and defined at a two dimensional position (x,y) as

$$g(x,y) = e^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2}\right)} e^{-i\frac{2\pi}{\lambda}(x-x_0)}. \tag{19}$$

Here x_0, y_0 is the center of the filter, $\sigma_{x,y}$ represents the width according to the axis and λ is the spatial frequency. To create a filter bank, the coordinated frame of the filter is rotated and scaled. We followed the findings from Ontrup et al (2004), which in turn are motivated by physiological experiments of visual perception from De Valois et al (1982), and used five different orientations and three sizes. This leads to a 30 dimensional feature vector. From this texture feature vector, the first four principal components are used. Including the proposed spatial decay from Ontrup et al (2004), the texture distance is augmented with the two dimensional position in image coordinates, leading to a distance vector of:

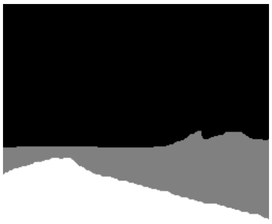
$$d(v_r, v_{r'}) = (|P_r^1 - P_{r'}^1|, |P_r^2 - P_{r'}^2|, |P_r^3 - P_{r'}^3|, |P_r^4 - P_{r'}^4|, ||p_r - p_{r'}||)^T$$

where P_r^i is the i-th principal component of the image patch represented by oscillator O_r and p_r is the position in two dimensional image coordinates.

As suggested in Ontrup et al (2004), the input image is partitioned in patches with a size of 8×8 pixels. On these patches the Gabor filters are applied. The interaction function is learned with 100 prototypes from the image in Fig. 13a in conjunction with the labels shown in Fig. 13b. The image has a size of 256×256 pixel, leading to a total of 1024 texture features after the partitioning and preprocessing. In regions of the label image, where an image patch spans over more than one class label, a simple majority vote is used to determine the actual label.

The grouping of a test image with the learned interaction function is shown in Fig. 13c. The overall structure of the image is well reflected by the grouping result, although small features like the end of the road, small branches of the trees on the right and the small pond on the left are not contained in the grouping result. Increasing the number of prototypes in this texture grouping scenario proved not to increase the grouping quality, but induces some spurious features in some of the

(a) Training image.


(b) Target labels.

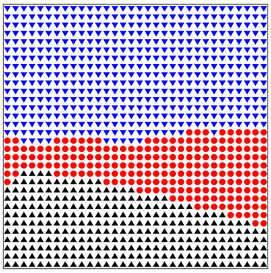
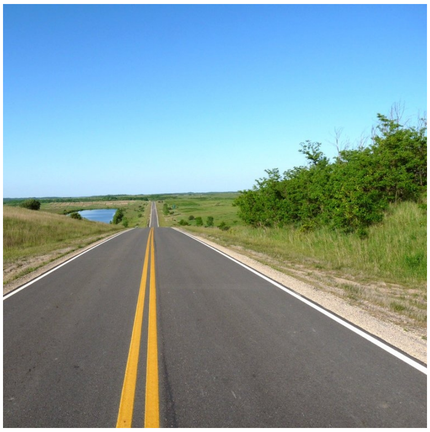

(c) Oscillator grouping.

Fig. 13 Training data used for the similarity of textures. The leftmost image shows the training input, which is split into patches and preprocessed with Gabor Filters. The hand labeled target groups for the training image are shown in the center and the result of the oscillator grouping is shown on the right. The same symbols/colors represent that corresponding features belong to the same perceptual group. (Images from Meier et al (2014))


(a) Test image.

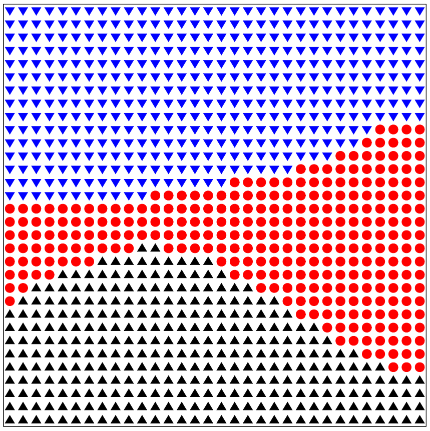

(b) Grouping result.

Fig. 14 Grouping result of a test image employing the learning interaction function from Fig. 13. The basic structure of the image, namely the road, background and sky, is presented well by the oscillator model. Smaller parts of the image, e.g. the distant part of the road and fine branches of the trees on the right are not represented in the segmentation. This may be due to the sub sampling of the input image into patches.

evaluation cases, e.g. small artifacts in the region of the pond and the branches of the trees.

5 Conclusion

Oscillator networks have been proven to be able to solve a broad spectrum of perceptual grouping tasks. Nevertheless, most of the state of the art approaches still rely on hand crafted compatibility rules specific to the grouping task at hand. Also, these rules can become rather complex depending on the problem. By employing a learning technique as the one presented here, this problem is relaxed. We additionally utilize an oscillator network which, in contrast to other approaches, does not require correlation based techniques for the evaluation of grouping results. We illustrated the feasibility of this approach with examples from different areas of visual perception and presented a modification of the learning algorithm from Weng et al (2006), that increases the grouping quality of perceptual networks. Evaluations with two perceptual grouping networks, the Competitive Layer Model and an oscillator network, revealed that the incorporation of ITVQ in the learning algorithm increases the grouping quality of these networks. Also, the learning approach is not limited to the application with the to presented network, but could be adapted to be used in conjunction with other types of grouping networks which employ attracting and repelling connections for their binding dynamics.

References

Arenas, A., Diaz-Guilera, A., Pérez-Vicente, C.J.: Synchronization processes in complex networks. Physica D: Nonlinear Phenomena 224(1), 27–34 (2006)

Bassett, D.S., Porter, M.A., Wymbs, N.F., Grafton, S.T., Carlson, J.M., Mucha, P.J.: Robust detection of dynamic community structure in networks. Chaos: An Interdisciplinary Journal of Nonlinear Science 23(1), 013,142–013,142 (2013)

Breve, F.A., Zhao, L., Quiles, M.G., Macau, E.E.: Chaotic phase synchronization and desynchronization in an oscillator network for object selection. Neural Networks 22(5), 728–737 (2009)

Chang, D., Nesbitt, K.V., Wilkins, K.: The gestalt principles of similarity and proximity apply to both the haptic and visual grouping of elements. In: Proceedings of the Eight Australasian Conference on User Interface, vol. 64, pp. 79–86. Australian Computer Society, Inc. (2007)

De Valois, R.L., Yund, W.E., Hepler, N.: The orientation and direction selectivity of cells in macaque visual cortex. Vision Research 22(5), 531–544 (1982)

Gallace, A., Spence, C.: To what extent do gestalt grouping principles influence tactile perception? Psychological Bulletin 137(4), 538 (2011)

Gray, C.M., Singer, W.: Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. Proceedings of the National Academy of Sciences 86(5), 1698–1702 (1989)

Heidemann, G., Ritter, H.J.: Efficient vector quantization using the wta-rule with activity equalization. Neural Processing Letters 13(1), 17–30 (2001)

Kuramoto, Y.: Chemical oscillations, waves, and turbulence. Dover (2003)

Li, C., Li, Y.: Fast and robust image segmentation by small-world neural oscillator networks. Cognitive Neurodynamics 5(2), 209–220 (2011)

Meier, M., Haschke, R., Ritter, H.J.: Learning of lateral interactions for perceptual grouping employing information gain. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 178–185. Springer, Heidelberg (2013)

Meier, M., Haschke, R., Ritter, H.J.: Perceptual grouping through competition in coupled oscillator networks. In: ESANN (2013b)

Meier, M., Haschke, R., Ritter, H.J.: Perceptual grouping by entrainment in coupled kuramoto oscillator networks. Network: Computation in Neural Systems 25(1-2), 72–84 (2014), `http://informahealthcare.com/doi/abs/10.3109/0954898X.2014.882524`, doi:10.3109/0954898X.2014.882524

Nomura, A., Ichikawa, M., Okada, K., Miike, H., Sakurai, T., Mizukami, Y.: Image edge detection with discretely spaced fitzhugh-nagumo type excitable elements. In: 2011 Joint 3rd Int'l Workshop on Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), pp. 1–8. IEEE (2011)

Ontrup, J., Wersing, H., Ritter, H.: A computational feature binding model of human texture perception. Cognitive Processing 5(1), 31–44 (2004)

Rao, S., Han, S., Principe, J.: Information theoretic vector quantization with fixed point updates. In: International Joint Conference on Neural Networks, IJCNN 2007, pp. 1020–1024 (2007), doi:10.1109/IJCNN.2007.4371098

Rényi, A.: Some fundamental questions of information theory. Selected Papers of Alfred Renyi 2(174), 526–552 (1976)

Ritter, H.: A spatial approach to feature linking. In: INNC (1990)

Treisman, A., et al.: The binding problem. Current Opinion in Neurobiology 6(2), 171–178 (1996)

Wagemans, J., Elder, J., Kubovy, M., Palmer, S., Peterson, M., Singh, M., von der Heydt, R.: A century of gestalt psychology in visual perception: I. perceptual grouping and figure-ground organization. Psychological Bulletin 138(6) (2012)

Wang, D.: Modeling global synchrony in the visual cortex by locally coupled neural oscillators. In: Eeckman, F.H. (ed.) Computation in Neurons and Neural Systems, pp. 109–114. Springer (1994)

Wang, D., Terman, D.: Locally excitatory globally inhibitory oscillator networks. IEEE Transactions on Neural Networks 6(1), 283–286 (1995)

Weng, S., Wersing, H., Steil, J., Ritter, H.: Learning lateral interactions for feature binding and sensory segmentation from prototypic basis interactions. IEEE Transactions on Neural Networks 17(4), 843–862 (2006)

Wersing, H.: Learning lateral interactions for feature binding and sensory segmentation. In: NIPS, pp. 1009–1016 (2001)

Wersing, H., Steil, J., Ritter, H.: A competitive-layer model for feature binding and sensory segmentation. Neural Computation 13(2), 357–387 (2001)

Yu, G., Slotine, J.J.: Visual grouping by neural oscillator networks. IEEE Transactions on Neural Networks 20(12), 1871–1884 (2009)

Analysing the Multiple Timescale Recurrent Neural Network for Embodied Language Understanding

Stefan Heinrich, Sven Magg, and Stefan Wermter

Abstract. How the human brain understands natural language and how we can exploit this understanding for building intelligent grounded language systems is open research. Recently, researchers claimed that language is embodied in most – if not all – sensory and sensorimotor modalities and that the brain's architecture favours the emergence of language. In this chapter we investigate the characteristics of such an architecture and propose a model based on the Multiple Timescale Recurrent Neural Network, extended by embodied visual perception, and tested in a real world scenario. We show that such an architecture can learn the meaning of utterances with respect to visual perception and that it can produce verbal utterances that correctly describe previously unknown scenes. In addition we rigorously study the timescale mechanism (also known as hysteresis) and explore the impact of the architectural connectivity in the language acquisition task.

1 Introduction

Natural language is the cognitive capability that clearly distinguishes humans from other living beings and is often called the key to intelligence. Humans not only utter short sounds to indicate an intention, but also describe procedural activities or may even completely think in natural language [10, 16]. However, language processing in the human brain and the acquisition of language has not yet been fully understood at the level of neural architectures. While hypotheses and models for innateness and universal grammars dominated the last 60 years (see [14] for a review on generativism), new neuroscientific findings and computational approaches led to alternative views towards emergence and constructivism (see [2, 26] for

Stefan Heinrich · Sven Magg · Stefan Wermter
University of Hamburg, Department of Informatics, Knowledge Technology
Vogt-Kölln-Straße 30, D - 22527 Hamburg, Germany
e-mail: {heinrich,magg,wermter}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/WTM/

reviews). With the tool sets of neural simulations and behavioural robotics we now can approach the following research question: what are the architectural characteristics for models of the human brain that favour the emergence of language?

1.1 Binding and Grounding in Computational Models

In the past researchers have suggested valuable models to explain the binding of language to experience or learned instances of certain roles, but also to ground language in embodied perception and action based on recent neuroscientific data and hypotheses. Recent computational models aimed at mimicking certain abstractions of circuits in the brain and tested them for instances of the binding and the grounding problem [21, 29].

To investigate systematicity in language processing, Frank studied empirically to what extent a neural architecture can bind learned words to novel roles (trained grammatical roles for which these words have not been trained) [17, 18]. For an *Echo State Network* (ESN) with an additional hidden layer a corpus of sentences was tested that stems from a small context free grammar, which allows to include recursions of relatives clauses. Compared to other *Recurrent Neural Network* (RNN) the ESN has a similar complexity in processing, but allows for easier training on the one hand and a more difficult in-depth analysis on the other hand. In the study it was found that language can be learned compositionally and that RNNs show strong systematicity, or in other words: generalisation for structural coarsely related sentences, both syntactically and semantically.

In various experiments Cangelosi et al. investigated the grounding of symbols in a computational model [5, 6, 7]. With the hypothesis that language can emerge from embodied interaction within an environment and a simultaneous exposure to words or “symbols”, a number of simulations were conducted. Firstly, stick-figure robots were supposed to perform actions with a number of proto-objects, for which they also perceived names. The study showed that the underlying neural feed-forward architecture can be trained to ground the label in the sensori-motor perception to produce a name for a perceived action or vice versa. Additionally, an analysis revealed that the architecture self-organised to a semantic representation in the hidden layer. Secondly, an iCub humanoid robot was set up to perform similar interaction tasks with increased complexity. In this study a similar neural architecture was tested, and it was shown that the labels for an object can be grounded in the visual perception. The robots in these approaches do not have full linguistic and compositional abilities, but can enrich their lexicon with simple mechanisms mimicking compositionality. These models are inspired by from research in developmental psychology and neuroscience to provide a better understanding of the emergence of complex cognitive and perceptual structures. Also, they provide a basis to test novel algorithms and methodologies for the development of effective interaction between humans and autonomous robotic systems. Both sets of studies emphasised the importance of integrating language and embodied perception.

In addition, early models captured the fusion of language and multi-modal perceptions or aimed at bridging the gap between formal linguistics and bio-inspired systems. For these approaches the idea is a certain abstraction of the environment and its representation in testing for language learning.

For instance, with the Cross-Modal Early Lexical Learning (CELL) framework, Roy and Pentland proposed a model of embodied word acquisition [40]. CELL is based on a multi-modal learning scheme, where semantic categories and object labels are learned simultaneously. Sequences of phonemes that are detected in a short time window are interpreted as words and associated to visual prototypes, represented by a histogram for the object's shape. The learning takes place in a semi-supervised fashion using a short-term memory for identifying the reoccurring pairs of acoustic and visual sensory data, which later are passed to a long-term representation of extracted audio-visual objects. In the experiment with data from caregiver-infant interactions it was shown that the system is able to pick up the ideal link of sounds forming a word (or in rare cases a onomatopoeic sound) to an object shape and thus associated a meaning to certain chains of phonemes. Although the model shows that the learning of language is much more effective, if the learning is grounded in visual perception, the study was constrained to the abstraction of words from input phonemes and the association of the words with shapes.

Furthermore, Rhode proposed a model for language comprehension and prediction based on an *Elman Recurrent Neural Network* (ERNN or more often called SRN for Simple RN) [38, 39]. The semantic part of the model was trained to abstract the meaning or "the message" of a sentence from a set of linguistic propositions, while the comprehension part of the network learned to extract this meaning from a sequence of words, which includes the distribution of the propositions. The network can be used also in the opposite direction in a way that it can predict the first word for a given meaning and can then predict the next words based on the feedback of the previous word and the meaning. The underlying claim of the model is that humans may learn to produce language based on the previously learned capability to formulate predictions as well as the simultaneous comprehension of language. In this architecture the RNN is used as a statistical tool that can predict a sequence based on a training with structured representation (predefined role binding) and does not attempt to capture embodied representations of the human cortex.

However, due to the vast complexity of language some models rely on well-understood Chomskyan formal theories, which are difficult to maintain in the light of recent neuroscientific findings, e.g. of non-infinite-recursive mechanisms and the evident involvement of various – if not all – functional areas in the human brain in language [35, 36]. A substantial number of studies indicate that the cognitive processes – including language processing – originate in multi-modal interactions with the environments and are encoded in terms of the overall goal involving all the relevant effectors [1, 4]. Other integrating or constructive models are constrained to single words, neglecting the temporal aspect of language, e.g. that both the representation on the level of speech sounds and the processing with a multi-time resolution are important [11, 24].

1.2 Language Acquisition in a Recurrent Neural Model

In a recent study Hinoshita et al. claimed that for human language acquisition just an “appropriate” architecture is sufficient and provided a model based on a *Multiple Timescale Recurrent Neural Network* (MTRNN) [25]. The RNN model learns language from continuous input of sentences composed of words and characters that stem from a small grammar. For the model no implicit information is provided on word segmentation and on roles or categories for words. Instead the input is modelled as streams of spike-like activities on character level. During training the architecture self-organises to the decomposition of the sentences hierarchically based on the explicit structure of the inputs and the specific characteristic of some layers. The authors found that the characteristics, e.g. the information processing on different timescales, indeed leads to a hierarchical decomposition of the sentences in a way that certain character orders forms words and certain word orders forms the sentences. Although in the study the model was reproducing learned symbolic sentences quite well, generalisation was not possible to test, because the generation of sentences was initiated by the internal state of some neurons, which had to be trained individually for every sentence.

In this chapter we incorporate embodied perception based on real world data in an MTRNN model and show that such a novel system is able to generalise to completely new situations by recomposing learned elements, and also self-organises towards the meaning of the learned verbal utterances. For both, the verbal utterances and the perception, we employ representations that are biologically inspired and avoid to provide structural information on the language. To acquire real world data and test the model in a language acquisition task in an embodied and situated agent, we employ an humanoid robot NAO that is supposed to learn language in interaction with different shaped and coloured objects. This work is an extension of the previous ICANN contribution [23], and in addition includes in-depth analyses of the roles of the network connectivity and the timescale concept in language acquisition.

1.3 Chapter Organisation

This chapter is organised as follows: With the related work in mind from the introduction, in Section 2 we will provide a detailed description of our model of an MTRNN extended by embodied perception. We include a complete formalisation to ease re-implementation. In Section 3 we will specify the scenario of the language learning robot as well as a complete description of the used representations of verbal utterances and embodied perception and the preceding encoding mechanisms. Then, in Section 4, follows our evaluation and the analysis. We report on the studies for generalisation capabilities as well as for the network behaviour and the impact of some key characteristics. Finally, in Section 5 we will discuss our findings, conclusions, and future prospects.

2 Extended MTRNN Model

To test for plausible characteristics for the semantic processing of verbal utterances, we incorporate both hypotheses into one model: a) speech is processed on a multiple-time resolution [24], and b) semantic circuits are involved in the processing of language [36]. We model the neural circuit as an RNN to achieve a reasonable biological plausibility, but also be able to analyse the networks behaviour on cortex level. More precisely, for our proposed model we employ the MTRNN to process verbal utterances over time [46], extended by several feed-forward layers to integrate embodied perceptions during the processing of utterances.

The MTRNN part is composed of an *Input- and Output* layer (IO) and two context layers called *Context fast* (Cf) and *Context slow* (Cs). Our extension part consists of an *Embodied Input* layer (EI), an *Embodied Fusion* layer (EF), and an *Embodied Controlling* layer (EC). Fig. 1 provides an overview of our architecture.

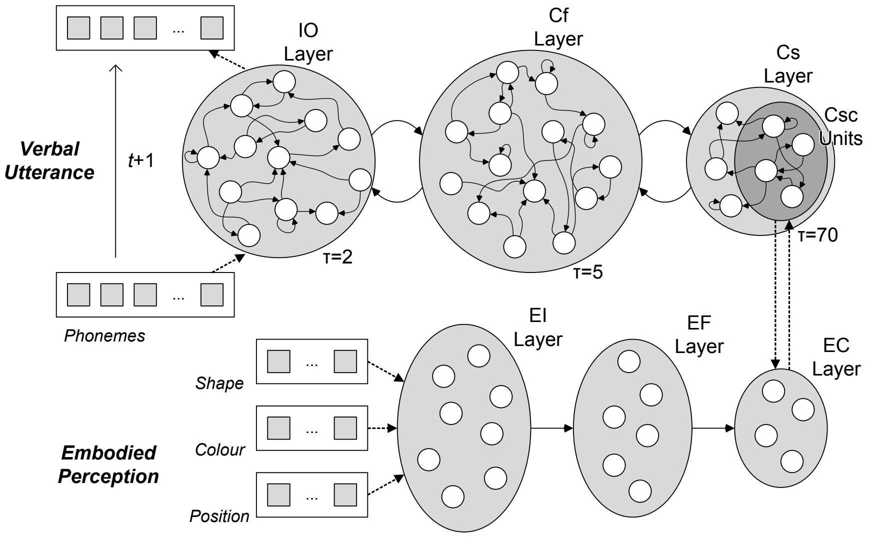


Fig. 1 Architecture of a Multiple Timescale Recurrent Neural Network extended by embodied perception from the scene. A sequence of phonemes (utterance) is processed over time, while the perceived embodied and situated information is constantly present.

During learning the MTRNN layers self-organise to the decomposition of a semantic meaning into a verbal utterance on phoneme level over time, while the feed-forward layers associate the meaning with the embodied perception. For production of utterances the feed-forward layers have the role of abstracting the meaning from the embodied input, whereas the MTRNN functions as a predictor of the next phoneme based on the context information and the previous sequence of phonemes.

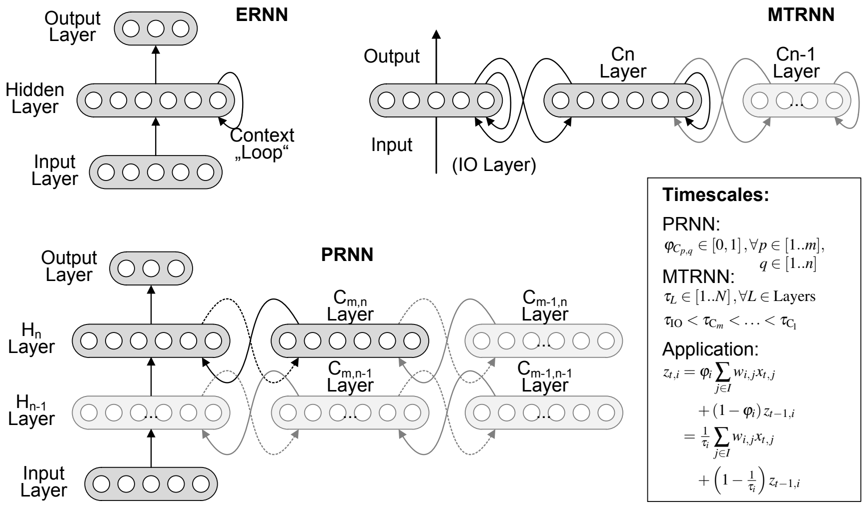
2.1 RNN Schematics

The MTRNN is composed of an *Input- and Output* layer that continuously produces an output from an input and from recurrent connections, as well as of an arbitrary number of coupled context layers. In general, the MTRNN is an extended *Elman Recurrent Neural Network* (ERNN) on the one hand and a special case of the *Plausibility Recurrent Neural Network* (PRNN) on the other hand [15, 43].

In contrast to the ERNN, the MTRNN allows for full connectivity of neurons to all neurons of the same and of adjacent layers. Also, all neurons process information based on incoming connections as well as their previous internal state. In principle the neurons maintain a fraction of previous information and process new information slower, based on a lagging parameter [33]. We call the parameter *hysteresis* φ, if we denote, which fraction of new information is taken into account (where $1-\varphi$ denotes, which fraction of old information is kept), or *timescale* τ, if we denote to which magnitude new information is processed slower and we can relate:

$$\varphi = \frac{1}{\tau} \quad . \tag{1}$$

Compared with the PRNN the MTRNN restricts this concept of hysteresis to an increasing slowness from the first to the last layer and also restricts the architecture to one horizontal set of layers only. A schematic comparison of the network architectures is shown in Fig. 2.



ERNN
Output Layer
Hidden Layer
Input Layer
Context „Loop“
MTRNN
Output
Input
(IO Layer)
Cn Layer
Cn-1 Layer
PRNN
Output Layer
H_n Layer
H_{n-1} Layer
Input Layer
$C_{m,n}$ Layer
$C_{m,n-1}$ Layer
$C_{m-1,n}$ Layer
$C_{m-1,n-1}$ Layer
Timescales:
PRNN:
$\varphi_{C_{p,q}} \in [0,1], \forall p \in [1..m], q \in [1..n]$
MTRNN:
$\tau_L \in [1..N], \forall L \in \text{Layers}$
$\tau_{\text{IO}} < \tau_{\text{C}_m} < \ldots < \tau_{\text{C}_1}$
Application:
$z_{t,i} = \varphi_i \sum_{j \in I} w_{i,j} x_{t,j} + (1-\varphi_i) z_{t-1,i} = \frac{1}{\tau_i} \sum_{j \in I} w_{i,j} x_{t,j} + \left(1 - \frac{1}{\tau_i}\right) z_{t-1,i}$

Fig. 2 Schematic comparison of Elman Recurrent Neural Network, Plausibility Recurrent Neural Network, and Multiple Timescale Recurrent Neural Network. The concept of timescales τ in the MTRNN is equivalent to the concept of hysteresis φ in the PRNN, but is restricted to increasing values τ for increasing numbers of layers to the right.

Based on the idea of introducing a parametric bias to the network [42], some neurons in the slowest context layer Cs of the MTRNN are also designated as *Context controlling units* (Csc units). The internal state of these units at the initial time step ($t = 0$) can be stored during training and can be used to initialise the generation of a meaningful sequence. By modulating these internal states, different other sequences can be generated.

2.2 Information Processing

In the MTRNN information is processed continuously with a constant firing rate as a sequence of T discrete-time steps. A sequence $s \in S$ is represented as a discretised flow of activations of the neurons in the IO layer $i \in I_{\mathrm{IO}}$. The input activation x of a neuron $i \in I_{\mathrm{all}} = I_{\mathrm{IO}} \cup I_{\mathrm{Cf}} \cup I_{\mathrm{Cs}}$ at time step t is calculated as:

$$x_{t,i} = \begin{cases} 0 & \text{iff } t = 0 \\ (1-\alpha)y_{t-1,i} + (\alpha)d_{t-1,i} & \text{iff } t \geq 1 \wedge i \in I_{\mathrm{IO}} \\ y_{t-1,i} & \text{iff } t \geq 1 \wedge i \notin I_{\mathrm{IO}} \end{cases} , \tag{2}$$

where $\alpha \in \,]0,1[$ is the feedback rate reflecting a *teacher forcing* (TF) signal of the desired output y^* to the input together with the generated output y of the last time step (see [12, 45]). The feedback is only given during training the MTRNN.

The internal state z of a neuron i at time step t is determined by:

$$z_{t,i} = \begin{cases} 0 & \text{iff } t = 0 \wedge i \notin I_{\mathrm{Csc}} \\ c_{0,i} & \text{iff } t = 0 \wedge i \in I_{\mathrm{Csc}} \\ \left(1 - \frac{1}{\tau_i}\right) z_{t-1,i} + \frac{1}{\tau_i} \sum_{j \in I_{\mathrm{all}}} w_{i,j} x_{t,j} & \text{otherwise} \end{cases} , \tag{3}$$

where $c_{0,i}$ is the initial internal state of the Csc units $i \in I_{\mathrm{Csc}} \subset I_{\mathrm{Cs}}$ (at time step 0), and $w_{i,j}$ are the weights from the jth to the ith neuron. In our model the MTRNN is specified by timescale values of $\tau = 2$, $\tau = 5$, and $\tau = 70$ for the IO, Cf, and Cs layers respectively, based on previous work [25, 46] and experiments in this study (see Section 4.3), indicating that these settings work well for language learning scenarios.

The output (activation value) y of a neuron i at time step t is defined by:

$$y_{t,i} = \begin{cases} \dfrac{\exp(z_{t,i} + b_i)}{\sum_{j \in I_{\mathrm{IO}}} \exp(z_{t,j} + b_j)} & \text{iff } i \in I_{\mathrm{IO}} \\ \mathrm{sig}(z_{t,i} + b_i) & \text{iff } i \notin I_{\mathrm{IO}} \end{cases} , \tag{4}$$

where b_i is the bias of neuron i. For the IO layer we employ a soft-max function due to the problem-specific representation, while for the neurons in the remaining layers we use a sigmoidal transfer function:

$$\mathrm{sig}\left(z_{t,i}+b_i\right)=\frac{1+2\kappa_a}{1+\exp\left(-\kappa_b\left(z_{t,i}+b_i\right)\right)}-\kappa_a \quad , \tag{5}$$

which is a logistic function with parameters κ_a for range and κ_b for slope. For our model we modulated the function with $\kappa_a = 0.35795$ and $\kappa_b = 0.92$ to capture the characteristics of the synchronic transfer function that has been proposed by LeCun for faster convergence in association tasks [32]. Although the choice of transfer functions is vast and well studied, more complex, but also more flexible functions exist [13]. In particular, the asynchonic tangens-hyperbolicus function and its synchronic equivalent – the logistic function – are used more often. We favour the use of a simple function since we are more interested in the general characteristics and also the comparability of the model and less in an optimal solution.

For the feed-forward extension the information processing follows analogously. Input perception p_s for a sequence is constantly present, while the output constitutes the activity for the initial internal states $c_0(s)$ of the Csc units for the sequence s. For all layers of the extension we use the same modified logistic transfer function.

2.3 Learning

During learning of the system the MTRNN is trained with verbal utterances, and self-organises the weights and also the internal state values of the Csc units. These self-organised values are then transferred to the EC layer and associated with the present embodied perception (EI layer). For training the MTRNN we use an adaptive variant of the *real-time backpropagation through time* (RTBPTT) algorithm [22, 45].

In the *forward pass* (FP) the error E is accumulating the error between the activation values (y) and the desired activation values (y^*) of the IO neurons at every time step as follows:

$$E(W)=\sum_{t}\sum_{i\in I_{\mathrm{IO}}} y^*_{t,i}\cdot\log\left(\frac{y^*_{t,i}}{y_{t,i}}\right) \quad , \tag{6}$$

where we use the *Kullback–Leibler divergence* as error function on IO neurons [31].

In the second step the partial derivatives of the calculated activation (y) and the desired activation (y^*) are derivated in a *backward pass* (BP):

$$\frac{\partial E}{\partial z_{t,i}}=\begin{cases} y_{t,i}-y^*_{t,i}+\left(1-\frac{1}{\tau_i}\right)\frac{\partial E}{\partial z_{t+1,i}} & \text{iff } i\in I_{\mathrm{IO}} \\ \mathrm{sig}'\left(y_{t,i}\right)\sum\limits_{k\in I_{\mathrm{all}}}\frac{w_{k,i}}{\tau_k}\frac{\partial E}{\partial z_{t+1,k}}+\left(1-\frac{1}{\tau_i}\right)\frac{\partial E}{\partial z_{t+1,i}} & \text{otherwise} \end{cases} \quad , \tag{7}$$

where the gradients are 0 for the time step $T+1$. The derivative for the sigmoidal transfer function is calculated as follows:

$$\mathrm{sig}'(y_{t,i}) = \frac{\kappa_b}{1+2\kappa_a}(y_{t,i}+\kappa_a)(1-y_{t,i}+\kappa_a) \quad . \tag{8}$$

Finally, with the determined gradients the weights w and biases b are updated:

$$w_{i,j}^{n+1} = w_{i,j} - \eta_{i,j}\frac{\partial E}{\partial w_{i,j}} = w_{i,j} - \eta_{i,j}\sum_t \frac{1}{\tau_i} x_{t,j}\frac{\partial E}{\partial z_{t,i}} \quad , \tag{9}$$

$$b_i^{n+1} = b_i - \beta_i\frac{\partial E}{\partial b_i} = b_i - \beta_i\sum_t \frac{1}{\tau_i}\frac{\partial E}{\partial z_{t,i}} \quad , \tag{10}$$

where the partial derivatives for w and b are the sums of weight and bias changes over the whole sequence respectively, and η and β denote the learning rates for the weight and bias changes. Here, we use individual learning rates for all weights and biases because we adapt them with respect to the gradient as described below.

The initial internal states $c_{0,i}$ of the Csc units define the behaviour of the network and are also updated as follows:

$$c_{0,i}^{n+1} = c_{0,i} - \zeta_i\frac{\partial E}{\partial c_{0,i}} = c_{0,i} - \zeta_i\frac{1}{\tau_i}\frac{\partial E}{\partial z_{0,i}} \quad \text{iff } i \in I_{\mathrm{Csc}} \quad , \tag{11}$$

where ζ_i denotes the learning rates for the initial internal state changes.

For training the association of the EC layer with the EI layer, we apply the *least mean square* (LMS) rule as error function [44]:

$$E(W) = \frac{1}{2}\sum_{i\in I_{\mathrm{EI}}}(y_i - y_i^*)^2 \quad , \tag{12}$$

$$\frac{\partial E}{\partial z_i} = \begin{cases} (y_i - y_i^*)\mathrm{sig}'(y_{t,i}) & \text{iff } i \in I_{\mathrm{EC}} \\ \mathrm{sig}'(y_i)\sum_{k\in I_{\mathrm{EC}}} w_{k,i}\frac{\partial E}{\partial z_k} & \text{iff } i \in I_{\mathrm{EF}} \end{cases} \quad , \tag{13}$$

where the desired output y^* corresponds to the activity derived from the $c_0$values:

$$y_i^* = \mathrm{sig}(c_i) \quad \forall i \in I_{\mathrm{EC}} \quad . \tag{14}$$

The adaptation of the weights and biases follows analogously.

2.4 Adaptive Learning Rates

In our approach the learning rates η, β, and ζ are adaptive, based on the local gradient information inspired by the *Resilient Propagation* (RPROP) algorithm [37]. In contrast to the original RPROP, learning rates are adapted and multiplied directly with the partial derivatives instead of only using the sign of the partial derivatives, to determine the change of the learning step:

$$\eta_{i,j}^{n} = \begin{cases} \min\left(\eta_{i,j}^{n-1}\xi^{+}, \eta_{\max}\right) & \text{iff } \left(\frac{\partial E}{\partial w_{i,j}} \cdot \frac{\partial E}{\partial w_{i,j}}^{n-1}\right) > 0 \\ \max\left(\eta_{i,j}^{n-1}\xi^{-}, \eta_{\min}\right) & \text{iff } \left(\frac{\partial E}{\partial w_{i,j}} \cdot \frac{\partial E}{\partial w_{i,j}}^{n-1}\right) < 0 \\ \eta_{i,j}^{n-1} & \text{otherwise} \end{cases} , \tag{15}$$

$$\beta_{i}^{n} = \begin{cases} \min\left(\beta_{i}^{n-1}\xi^{+}, \eta_{\max}\right) & \text{iff } \left(\frac{\partial E}{\partial b_{i}} \cdot \frac{\partial E}{\partial b_{i}}^{n-1}\right) > 0 \\ \max\left(\beta_{i}^{n-1}\xi^{-}, \eta_{\min}\right) & \text{iff } \left(\frac{\partial E}{\partial b_{i}} \cdot \frac{\partial E}{\partial b_{i}}^{n-1}\right) < 0 \\ \beta_{i}^{n-1} & \text{otherwise} \end{cases} , \tag{16}$$

where $\xi^{+} \in]1,\infty]$ and $\xi^{-} \in]0,1[$ are the increasing or decreasing factors respectively and $\eta_{\max} > \eta_{\min}$ are upper and lower bounds for both learning rates η and β. If the partial derivative of the current epoch n is pointing to the same direction as in the former epoch $n-1$, then the learning rate is increased. If the direction of the partial derivative is pointing to the other direction, then the minimum has been missed and the learning rate is decreased.

For the update of the initial internal states $c_{0,i}$ the learning rates ζ are adapted proportionally to the average learning rates η of all weights that are connected with unit i and neurons of the same (Cs) and the adjacent (Cf) layer:

$$\zeta_i \propto \frac{1}{|I_{\mathrm{Cf}}| + |I_{\mathrm{Cs}}|} \sum_{j \in (I_{\mathrm{Cf}} \cup I_{\mathrm{Cs}})} \eta_{i,j} \quad . \tag{17}$$

Since the update of the $c_{0,i}$ depends on the same partial derivatives (time step 0) as the weights, we do not need additional parameters in this adaptive mechanism.

2.5 Production

During testing the system approximates EC values from the embodied perception input at the EI layer. From the EC values the corresponding values of Csc units are calculates using the inverse of Eq. 14, which in turn initiate the generation of a corresponding verbal utterance. These processing steps are done in a single set of computation – no additional training or adaptation is necessary.

3 Scenario

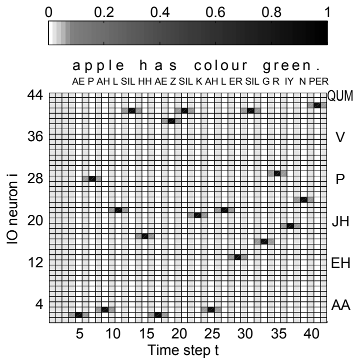
Our scenario for this model is the interaction between a human teacher and a robotic learner, which is supposed to learn language from scratch by grounding utterances in its embodied experience, but also is supposed to use its learned language to describe novel situations. We believe it is important to test the learning in a real environment to face the influence of natural noise and uncertainty of perception.

The robot is placed in a scene and receives an utterance from the teacher, who describes the scene, e.g. “THE APPLE HAS COLOUR GREEN”. The system should learn, in a self-organised way, how to bind the visual scene information with this verbal expression to be able to describe another scene like “THE BANANA HAS COLOUR GREEN” correctly. The focus of this study is on generalisation using possibly learned components.

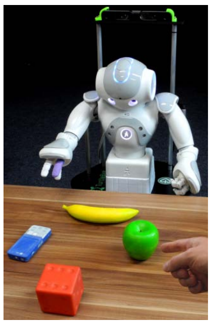
To control our setup, all verbal utterances stem from a small symbolic grammar as presented in Fig. 3a. However, every symbolic sentence is transformed into a phonetic utterance based on phonemes from the ARPAbet and four additional signs to express pauses and intonations in propositions, exclamations, and questions: $A = \{\text{'AA'}, \ldots, \text{'ZH'}\} \cup \{\text{'SIL'}, \text{'PER'}, \text{'EXM'}, \text{'QUM'}\}$, with size $|A| = 44$.

```
S → INFORM
INFORM → POS is a OBJ.
INFORM → OBJ has colour COL.
OBJ → apple | banana | dice | phone
POS → above | below | left | right
COL → blue | green | red | yellow
```

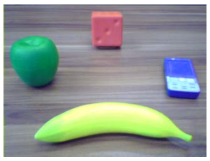
(a) Grammar.



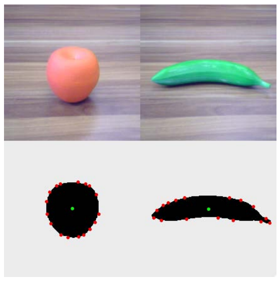
(b) Encoded utterance.



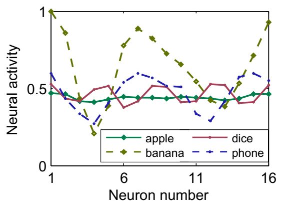
(c) Learner.



(d) Learner’s view.



(e) Perceived shapes.



(f) Shape representation.

Fig. 3 Representations and scenario of language learning in human-robot interaction

3.1 Utterance Encoding

To encode an utterance $u = (p_1, \ldots, p_{|u|})$ into neural activation over time, we adapted the encoding scheme suggested by Hinoshita et al. [25], but we use a phoneme-based instead of a symbol-based representation: The occurrence of a phoneme p_k is represented by a spike-like neural activity of a specific neuron at relative time step r. In addition, some activity is spread backward in time (rising phase) and some activity is spread forward in time (falling phase) represented as a Gaussian over the interval $[-\omega/2, \ldots, -1, 0, +1, \ldots, \omega/2]$. On absolute course of time t the peaks mimic priming effects in articulatory phonetic processing. For example the previous occurrence of the phoneme "P" could be often related to the occurrence of the phoneme "AH" leading to an excitation of the respective neuron for "AH", when the neuron for "P" was activated.

All activities of spike-like peaks are normalised by a soft-max function for every absolute time step t over the set of input neurons. A sketch of the utterance encoding is shown in Fig. 4.

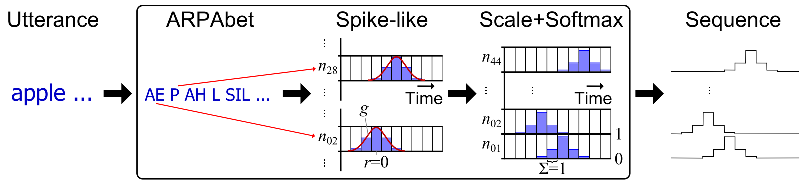


Fig. 4 Schematic process of utterance encoding. The input is a symbolic sentence, while the output is the neural activity over N neurons times T time steps.

The Gaussian g for p_k is defined by:

$$g_{k,r,i} = \begin{cases} \exp\left(\dfrac{-r^2}{2\sigma^2}\right) & \text{iff } p_k = A_i \\ 0 & \text{otherwise} \end{cases} , \tag{18}$$

where $r = 0$ is the mean and σ the filter sharpness factor. A peak occurs for the neuron $i \in I_{\text{IO}}$ with $|I_{\text{IO}}| = |A|$, if the phoneme p_k is equal to the ith phoneme in the phoneme alphabet A. From the spike-like activities the internal state z of a neuron i at time step t is determined by:

$$z_{t,i} = \begin{cases} \lambda \cdot \max\left(g_{k=1\ldots|u|,r=-\omega/2\ldots\omega/2,i}\right) & \text{iff } t = \mu + k\upsilon + r \\ 0 & \text{otherwise} \end{cases} , \tag{19}$$

where ω is the filter width, μ is a head margin to put some noise to the start of the sequence, υ is the interval between two phonemes, and λ is a scaling factor for the neuron's activity y^*.

The scaling factor depends on the number of IO neurons and scales the activity to $d \in]0, 0.9]$ for the specified soft-max function:

$$\lambda = \ln\left(\frac{0.9}{1.0 - 0.9}\left(|I_{\mathrm{IO}}| - 1\right)\right) \quad , \tag{20}$$

$$y^*_{t,i} = \frac{\exp\left(z_{t,i}\right)}{\sum\limits_{j \in I_{\mathrm{IO}}} \exp\left(z_{t,j}\right)} \quad . \tag{21}$$

For our scenario we set the constants to $\mu = 4$, $\omega = 4$, $\sigma^2 = 0.3$, and $\upsilon = 2$. The ideal neural activation for an encoded sample utterance is visualised in Fig. 3b.

3.2 Visual Perception Encoding

To encode the visual shape perception into sustained neural activity, we aim at capturing a representation that is biologically plausible, but on a level of abstraction of shapes as found in the *posterior infero-temporal* (PIT)/V4 area [34]. On an image taken by the NAO robot we employ the mean shift algorithm for segmentation [9], and the Canny edge detection as well as the contour finder for object discrimination [8, 41]. Subsequently, we calculate the centre of mass and 16 distances to salient points around the contour. Finally, we scale the distances by the square root of the object's area and order them clockwise – starting with the largest – to determine the characteristic shape, which is scale- and rotation-invariant. Fig. 3e provides two example results of this process, and Fig. 3f visualises typical characteristics for the employed object shapes (scaled to $[0, 1]$). Encoding of the perceived colour is realised by averaging the three R, G, and B values of the shape, while the perceived position is encoded by the two values of the centroid coordinate in the field of view. A sketch of the visual perception encoding is shown in Fig. 5.

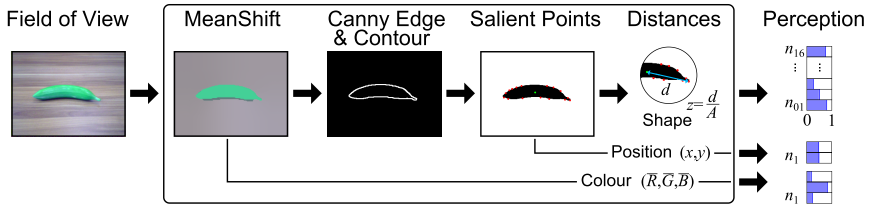


Fig. 5 Schematic process of visual perception encoding. The input is a single frame taken by the NAO camera, while the output is the neural activity over N neurons, with N is the sum over shape + colour + position features.

4 Evaluation and Analysis

To understand the dynamics of the architecture in this study, we are interested in evaluating the generalisation capabilities, and the role of some key characteristics like connectivity and timescales. We also aim at analysing the network behaviour in generating utterances for known as well as for novel scenes.

To test and analyse our model, we collected a data set consisting of all possible scenes and their respective verbal description. From the grammar we obtained 32 different combinations, which we set up as scenes and in turn used for collecting different examples. The corresponding verbal utterances were reasonably complex sequences with a length of 32 to 46 time steps (compare Fig. 3b). Subsequently, we ran a series of experiments, for which we carefully, but randomly divided the data into a training set and a test set (50:50) – making sure that every scene is included only in one of these sets – and trained ten randomly initialised systems. For every setup we repeated this process ten times with different distributions of data in training and test set to arrive at 100 runs for analysis. Compared to the previous ICANN contribution [23], the results are based on twice the number of runs per setup and per experiment. The parameters of the network and the meta-parameters were mostly chosen based on the experience in [22] and [25] and are detailed in Tab 1. The number of neurons in the input layers $|I_{\mathrm{IO}}|$ and $|I_{\mathrm{EC}}|$ are given by the input representations. The size of EC depends on and is equal to the size of Csc, which we determined with $|I_{\mathrm{Csc}}| = \lceil |I_{\mathrm{Cs}}|/2 \rceil$. As the termination criteria for the learning, we used a maximum number of epochs with $\theta = 50,000$ and minimal average errors on the IO and EI layers with $\varepsilon_{\mathrm{IO}} = 5.0 \times 10^{-4}$ and $\varepsilon_{\mathrm{EI}} = 5.0 \times 10^{-6}$. We favoured the use of fixed termination criteria over the use of a validation set to allow for comparisons on the meta-parameters.

Table 1 Standard parameter settings for evaluation

Param.	Description	Value
$\|I_{\mathrm{IO}}\|$	Number of IO neurons	44
$\|I_{\mathrm{Cf}}\|$	Number of Cf neurons	80
$\|I_{\mathrm{Cs}}\|$	Number of Cs neurons	23
$\|I_{\mathrm{Csc}}\|$	Number of Csc neurons	12
$\|I_{\mathrm{EC}}\|$	Number of EC neurons	12
$\|I_{\mathrm{EF}}\|$	Number of EF neurons	16
$\|I_{\mathrm{EI}}\|$	Number of EI neurons	21
$\mathbf{W}^0$	Initial weights range	± 0.025
$\mathbf{C}_0^0$	Initial Csc values range	± 0.01

Param.	Description	Value
τ_{IO}	Timescale of IO neurons	2
τ_{Cf}	Timescale of Cf neurons	5
τ_{Cs}	Timescale of Cs neurons	70
α	Teacher forcing	0.1
$\eta_{\max}$	Maximal learning rate	1.0
$\eta_{\min}$	Minimal learning rate	1.0×10^{-6}
ξ^+	Increasing factor	1.01
ξ^-	Decreasing factor	0.96
η^0, β^0, ζ^0	Initial learning rates	0.05

4.1 Generalisation

To be able to compare the generalisation capabilities, we use the standard measure F_1-score determined by precision and recall, and defined as follows:

$$p_{\text{precision}} = \frac{tp}{tp+fp} \quad , \quad p_{\text{recall}} = \frac{tp}{tp+fn} \quad ,$$

$$F_1\text{-score} = 2 \cdot \frac{p_{\text{precision}} \cdot p_{\text{recall}}}{p_{\text{precision}} + p_{\text{recall}}} \quad , \tag{22}$$

where we specify all correct and matching utterances as tp (true positives), all correct, but not matching utterances as fp (false positives), and strictly all incorrect utterances as fn (false negatives).

Table 2 Parameter variation in the generalisation experiment

Dimension	Parameter	Description	Values
1	$(\lvert I_{\text{Cf}}\rvert, \lvert I_{\text{Cs}}\rvert)$	Number of Cf, Cs neurons	$\{(40,11),(80,23),(160,47)\}$
2	$\lvert I_{\text{EF}}\rvert$	Number of EF neurons	$\{8,16,24\}$

The results in Tab. 3 show that the system can be trained perfectly in most cases, and also produces correct utterances for new scenes on a moderate level: For a suitable parameter setting, networks reach an F_1-score of up to 1.0 on the training set and 0.545 on the test set, with an average over all random seeds of 0.999 on the training set and 0.185 on the test set. From the chart in Fig. 6 for the same results for the test set only, we can learn that the size of the network dimension is important for ideal generalisation capabilities. The F_1-score on utterance level is clearly stricter than on word or on phoneme level, but we aim at evaluating, if the complete meaning of the scene was uttered correctly.

Table 3 Comparison of F_1-score for different network dimensions

$\lvert Cf\rvert/\lvert Cs\rvert$	40/11	40/11	40/11	80/23	80/23	80/23	160/47	160/47	160/47
$\lvert EF\rvert$	8	16	24	8	16	24	8	16	24
training set best	1.000	1.000	1.000	1.000	1.000	**1.000**	1.000	1.000	1.000
test set best	0.316	0.316	0.316	0.476	0.476	**0.545**	0.476	0.400	0.400
training set best avg *	1.000	1.000	1.000	1.000	1.000	**1.000**	1.000	1.000	1.000
test set best avg *	0.200	0.229	0.207	0.322	0.333	**0.336**	0.277	0.254	0.264
training set average	0.913	0.927	0.928	0.948	0.999	**0.999**	0.988	0.998	0.996
test set average	0.068	0.072	0.076	0.133	0.177	**0.185**	0.098	0.119	0.115

* Averaged over all best networks of all data set distributions.

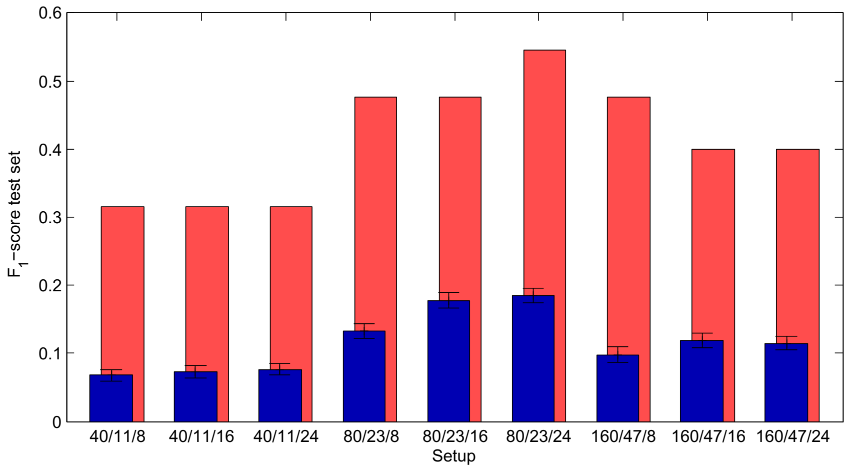


Fig. 6 Comparison of the F_1-score on the test set for the generalisation experiment. The dark/blue bars and the error bars present the average F_1-score and the standard error of means respectively, while the bright/red bars show the F_1-score of the best network for the respective setup.

Note that due to the random selection the system had to describe a scene in several cases, for which it had not seen any aspect (shape, colour, or position) before. This was intended to keep the scenario realistic and observe the effects.

In experiment we observed for incorrect utterances three types of errors: a) Minor substitution errors in terms of a single wrong phoneme or a pause that was too long ("SIL SIL" instead of "SIL"), b) word confusion errors, and c) phoneme chains without any meaning. Tab. 4 provides example results for observed errors. Errors of type (a) occurred often for networks in which the MTRNN part did not converge well to small average errors. For errors of type (b) we only found very few instances, and in these cases found the confused word mostly in the end of the sentence. A reason for this error was not found in this experiment, but further experiments (compare Sec. 4.3) indicates a link to the timescale parameter. The type (c) error appeared often in cases in which the training set and the test set are structural very different, e.g. when the test scene consisted of unknown aspects, as described above.

Table 4 Examples for different correct and incorrect utterances for errors (a), (b), and (c). Incorrect phonemes are emphasised bold.

correct	B AH N AE N AH SIL HH AE Z SIL K AH L ER SIL B L UW PER
substitution error (a)	R AY T SIL IH Z SIL AH SIL ***B*** AY S PER
substitution error (a)	B IH L OW SIL IH Z ***SIL SIL*** AH SIL AE P AH L PER
word confusion (b)	B AH N AE N AH SIL HH AE Z SIL K AH L ER SIL ***G R IY N*** PER
phoneme babbling (c)	***AE P AH AE SIL AH SIL AE AE Z K P L ER EH*** R EH D . . .

4.2 The Role of Connectivity and Pathways

During training of the system we found that the connection weights from the Cs to the Cf layer as well as from the Cf to the IO layer converged towards zero in many cases. This means that the highly dynamic networks organised themselves towards a directed flow of information from the context to the phonetic output instead of a mutual exchange of information.

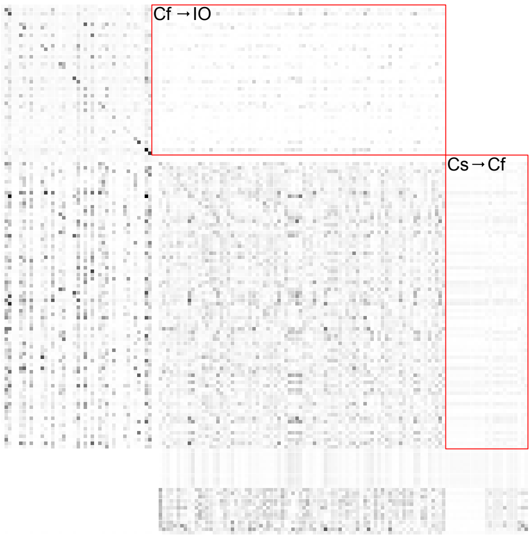
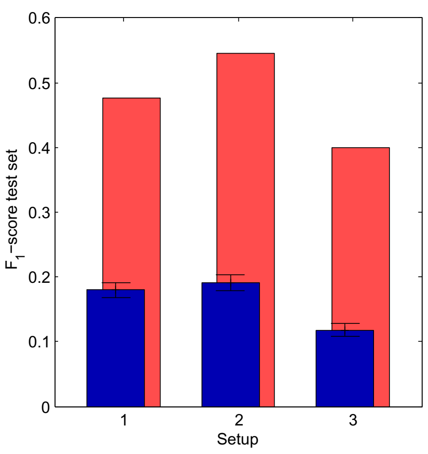


Fig. 7 Connectivity for an example network trained with the standard parameters and visualised as a Hinton diagram, where a square represents a connection weight from a neuron (horizontal dimension) to another neuron (vertical dimension). The diagram has been modified in a way that the strong connections are shown towards black (omitting the sign of the weights to increase readability), while weak connections are shown towards white

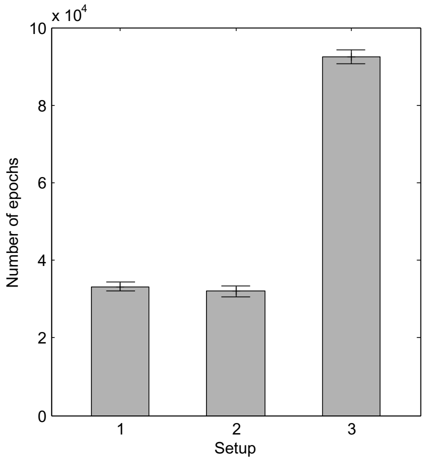
To test the hypothesis that the MTRNN architecture might already be more complex than necessary and should be studied with less initial connectivity, we set up an experiment with modified connectivity and compared the following setups:

1. No modification (baseline): all neurons of a layer are connected to all neurons of the same and of adjacent layers.
2. All neurons of a layer are connected to all neurons of the same and of adjacent layers, but the connection weights from Cs to Cf and from Cf to IO are initialised with 0.0 instead of ± 0.025.
3. Connections from Cs to Cf and from Cf to IO are removed.

We trained the networks with the procedure and the standard parameters as described above (see Tab. 1), but increased for the training the maximum number of epochs to $\theta = 100,000$, to ease the comparison of the training effort for the modifications. The results presented in Fig. 8 show that on the test data the F_1-score is slightly, but not significantly higher for setup 2 compared to setup 1, whereas the F_1-score is significantly ($p < 0.001$) lower for setup 3 compared to setup 1. However, the training effort for setup 2 is a bit but significantly ($p < 0.01$) smaller, and for setup 3 vastly larger (significant, $p < 0.001$) than for setup 1.



(a) F_1-score on test set.



(b) Training effort.

Fig. 8 Comparison of generalisation capability and training effort for modifications of the MTRNN connectivity. For (a) the dark/blue bars represent the average F_1-score, while the bright/red bars show the F_1-score of the best network for the respective setup. The error bars denote the respective standard error of means.

Note that for setup 2 we do not expect a higher F_1-score compared to setup 1, since in the training process all weights self-organise with respect to the partial derivatives. However, the results indicate that the introduced "bias" of having low connectivity from Cs to Cf and from Cf to IO leveraged the training process and led to faster convergence. For setup 3 the results show that having no backward connectivity makes the language acquisition problem much harder, indicating that backward connections are indeed necessary.

In terms of types of errors for the incorrect utterances we did not find considerable differences between setup 2 and setup 1, but a larger number of substitution errors for setup 3 compared to setup 1.

4.3 The Role of the Timescale Parameter

In preliminary experiments we confirmed that simpler RNNs cannot reproduce the generalisation capabilities of the MTRNN on the language acquisition. We tested both the ERNN with additional *parametric bias* units attached to the hidden layers (RNNPB, for details see [42]), as well as the MTRNN architecture with no timescale mechanism. Although the networks could learn the training data to some extent, the generation of utterances for novel scenes led to meaningless phoneme babbling. Basically the networks did not self-organise to the decomposition of the training sequences, but to reproduce them in whole, and thus generalisation ability was not evident.

Table 5 Parameter variation in the timescale experiment

Dimension	Parameter	Description	Values
1	τ_{Cf}	Timescale of Cf neurons	$\tau_{\mathrm{IO}} \cdot k,\, k \in \{1,2,3,4,5,6\}$
2	τ_{Cs}	Timescale of Cs neurons	$\tau_{\mathrm{Cf}} \cdot l,\, l \in \{2,6,10,14,18,22\}$

Because the concept of hysteresis or timescales seems crucial for the language acquisition task, we investigated the influence of the timescale parameter. In a rigorous experiment we systematically varied the combination of timescale values of the neurons in the Cf and in the Cs layer. More precicely we tested the 2-fold up to 6-fold of the timescale for Cf with respect to the timescale for IO (fixed to $\tau_{\mathrm{IO}} = 2$) and also the 2-fold up to 22-fold of the timescale for Cs with respect to the timescale for Cf. For every combination as shown in Tab. 5 we trained 100 networks in the procedure as described above and kept all other parameters fixed. In sum we tested for 36 combinations leading to 3600 trained networks. Since we are interested both in the influence on the convergence of the networks for the given data set as well as in the generalisation capabilities we define a mixed score:

$$\begin{aligned} F_{1,\mathrm{mixed}}\text{-score} = &(F_1\text{-score(training set average)} + F_1\text{-score(test set average)} \\ &+ F_1\text{-score(training set best avg)} + F_1\text{-score(test set best avg)})/4\,. \end{aligned}$$

The result of the experiment is visualised in Fig. 9, where high (desired) scores are shown in red and low scores are shown in blue. From the map we can obtain that using increasing timescales for the different layers increases the score. However, the scores do not differ much on a certain plateau: Networks for timescale ratio $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}}$ of 2 and $\tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}}$ of 6 or higher reached a score of > 0.6, but this score does not

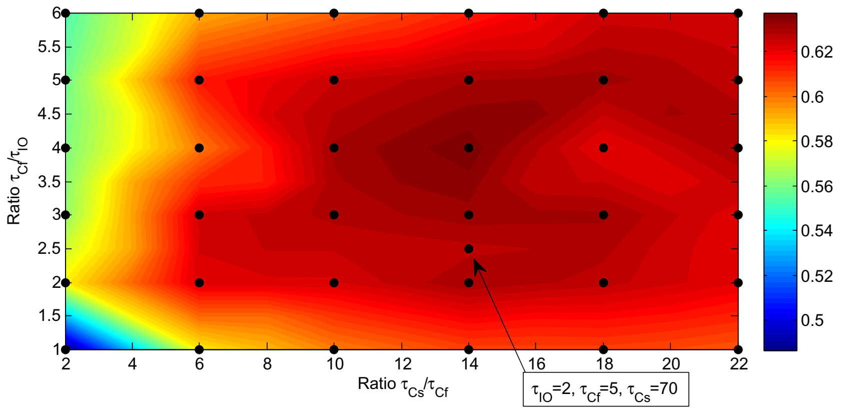


Fig. 9 $F_{1,\mathrm{mixed}}$-score for different combinations of timescale values of the Cf and the Cs neurons. Desired scores (high) are shown in red.

increase considerably for larger timescale ratios. In the results we can find some peaks e.g. for $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}} = 4, \tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}} = 14$ ($\tau_{\mathrm{Cf}} = 8, \tau_{\mathrm{Cs}} = 112$), but the differences in the score values compared to e.g. the baseline ($\tau_{\mathrm{Cf}} = 5, \tau_{\mathrm{Cs}} = 70$) are not significant.

To investigate the differences in the results for networks with smaller timescale ratio (both $\tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}}$ and $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}}$) we looked at the erroneous utterances that these networks produced on IO level. For both cases we noticed that the number of incorrect words as well as substitution errors in the end of the utterances occurred more often. The networks with smaller $\tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}}$ ratio generated syntactically correct but semantically not matching words more often for longer utterances, while networks with smaller $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}}$ in general started to generate meaningless phoneme babbling more often. In summary, the results indicate that:

- The timescale for neurons in Cf is ideally of the length of the number of time steps for an average word length. For example the average word length in our scenario is 3.156 phonemes or 6.313 time steps, while the average inter-word distance (distance between the beginning of words including pauses) is 4.208 phonemes or 8.417 time steps.
- The timescale for neurons in Cs is ideally equal to or larger than the number of time steps of the longest sequence for a high score. However, very large timescales increase the training effort significantly. Recall, in our scenario we used sequences with length up to 46 time steps.

In an additional test we investigated the first indication further. We modified our corpus of utterances in a way that we changed all translations from words to phonemes to half the number of phonemes for the first setup and to double the number of phonemes for the second setup. Again, we trained the networks with different ratios $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}}$ (compare Tab.5), while keeping the ratio fixed for the first setup with $\tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}} = 7$ and for the second setup with $\tau_{\mathrm{Cs}}/\tau_{\mathrm{Cf}} = 28$ due to the halved and doubled sequence lengths respectively. From the results in Fig 10 we can take that the estimate holds also for shorter and longer average word lengths as well.

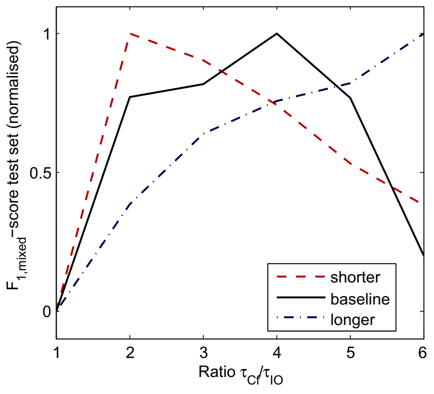


Fig. 10 Comparison of $F_{1,\mathrm{mixed}}$-score for different timescale values over shortened and prolonged average word lengths. The timescale ratio are varied for $\tau_{\mathrm{Cf}}/\tau_{\mathrm{IO}}$ layer only. For the first setup, all words have been artificially halved in length (to a minimal length of one phoneme) and for the second setup, all words have been doubled in length. Results have been normalised for each setup to increase readability.

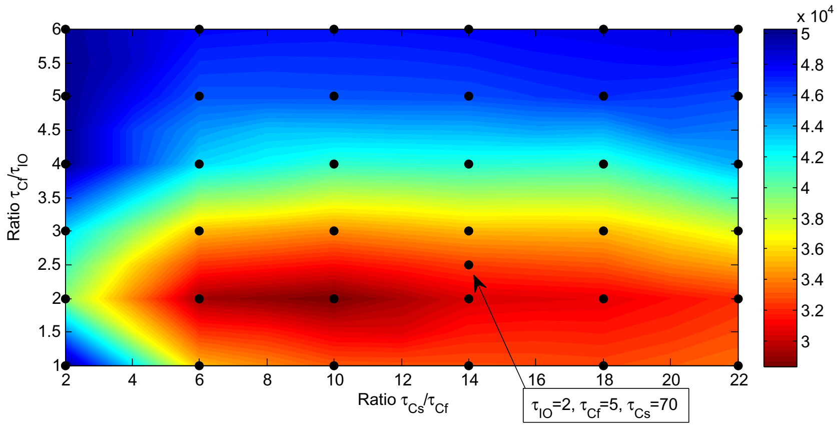


Fig. 11 Training effort (number of training epochs until termination) for different combinations of timescale values of the Cf and the Cs neurons. Desired (low) numbers are shown in red.

To also compare the difficulty to train the networks we looked at the average number of epochs until the training reached one of the termination criteria. The numbers are presented in Fig. 11, where low (desired) numbers are shown in red and high numbers are shown in blue.

For some combinations of timescale values around $\tau_{\text{Cf}}/\tau_{\text{IO}} = 2, \tau_{\text{Cs}}/\tau_{\text{Cf}} = 10$ ($\tau_{\text{Cf}} = 4, \tau_{\text{Cs}} = 40$) we found the smallest training effort, while for larger timescales both for Cf and Cs neurons the effort increases.

Combining both results, the scores on training and test data as well as the training effort can provide a rough estimate of good parameter values for practical applications. For example in Fig. 12 a possible combination is shown, where we weighted the proportion of the score five times over the proportion of the effort.

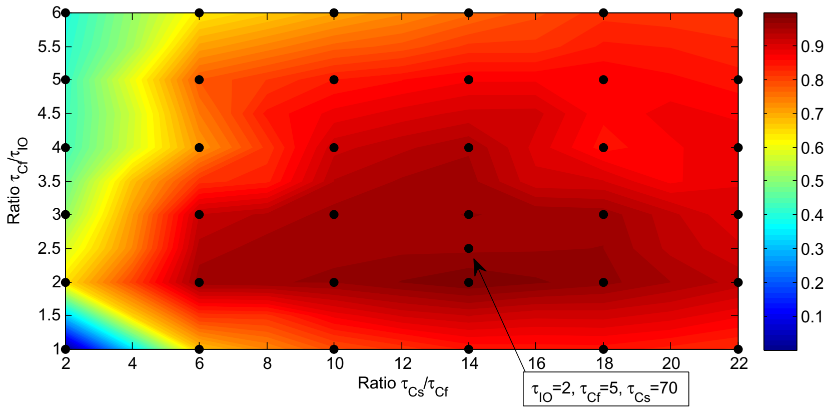
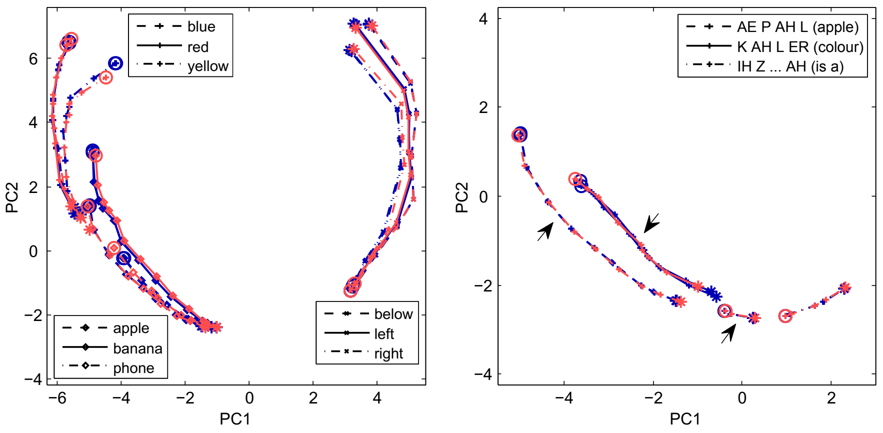


Fig. 12 Combination of $F_{1,\text{mixed}}$-score and training effort (5:1) for practical applications. Desired values are shown in red and may indicate good parameters.

4.4 Network Behaviour

To provide a better understanding of the system, we analysed the neural activity of the Cf layer for the trained networks. We aimed to test whether this layer had organised itself to represent the words in the utterances (compare [25]). Using *principle component analysis* (PCA), we reduced the dimensionality to visualise trajectories over time for specific words. The start and end point of the trajectory were defined as the first highest activity for the first phoneme and the last highest activity for the last phoneme of the word in the IO layer.

The results reveal several characteristics (see Fig. 13 for the trajectories of a typical network): Firstly, the neural activity in the Cf layer is nearly identical for the same words from trained utterances. Secondly, the same words from untrained utterances have a quite similar activity pattern. Thirdly, words of the same type (shape, colour, or position words) have very related activity patterns. From the data we can observe that the networks self-organise to patterns for words about shapes, colours, and positions. Fourthly, words with similar phonetic representations have different activities, if the type of the word is different. Low correlation was found of activity for phonetically similar but semantically different words.



(a) Words of similar type. (b) Words with similar phonetics.

Fig. 13 Comparison of neural activation in the Cf layer for different words. The dimensionality has been reduced from $|Cf|$ to two dimensions (PC1 and PC2) and the beginning ($*$) as well as the end ($\circ$) of the words have been marked. The dark/blue lines represent words from utterances of the training set and the bright/red lines show words from utterances of the test set. Arrows indicate the same phoneme "AH".

In addition, we found the tendency that the activation of a word primes the activation of other grammatically related words. In terms of trajectories it can be observed that the end point of the word "COLOUR" is close to the starting point of all colour words, and the end point of a position word is close to the starting point of "IS A ..." (compare Fig. 13a and b).

5 Discussion

The combination of visual perception and an architecture that includes different timescales in processing verbal sequences provides a system that self-organises towards the perceptual meaning of learned utterances in a real world scenario. Our experiments have shown that such a system apparently is able to understand verbal utterances and describe novel scenes with the correct corresponding verbal utterances. The analysis revealed that novel scenes are described by recomposing the correct words, which have been grounded in the perception of different shapes, colours, or positions.

Analysis of the errors for incorrect utterances revealed a) minor substitution errors, b) word confusion errors and c) phoneme babbling errors. In cases of type (a) listening humans would presumably consider this a normal inaccuracy and automatically correct the error. Errors of type (b) may indicate effects of the memorising capacity. For the trained networks we observed the word confusion error mostly in such cases, where timescale parameter values have been chosen sub-optimally. Neural activity in the Cs layer revealed that the networks seemingly could not produce the correct word, because they "forgot" the meaning of the scene at a certain time step and initiated the production of the most probable next word. Further research in the brain 's information habituation could clarify this observation. Case (c) clearly shows that generalisation was sometimes difficult. It is open to clarify, whether this degree of difficulty is inherent, e.g. if the error rate is comparable to certain learning stages in young children during early language learning [30].

During training we also observed that connectivity plays an important role for the behaviour of the network. Although we found that the connection weights from the Cf to the Cs layer as well as from the IO to the Cf layer in many cases converged towards zero, we learned in additional experiments that we cannot leave out the backward connections. We found that a more directed flow of information from the context to the phonetic output was the result of the training, but a certain feedback seems to be important as well. In the light of neuroscientific evidence the directed information flow from the conceptual network (reflected by Cs) to the articulatory areas (reflected by IO) is plausible [24]. Also, in computational studies, researchers found that network architectures of biological plausible integrate-and-fire-neurons tend to form a mostly feed-forward structure out of initial randomly connected network for recurring input patterns [27, 28]. However, for many cortical regions of the human brain, for example in vision, it was also reported that certain proportions of backward (feedback) connections exist and play an important role [19, 20].

The examination of the timescale parameter revealed that the hysteresis mechanism (the timescales) is a key element for learning complex sequences like longer phoneme chains. Firstly, our results confirm that the hysteresis mechanism may be a required architectural characteristic that favours the emergence of language. Secondly, our results suggest that ideal parameter values are indeed problem-dependent, but less ideal values still lead to good performances. For the language acquisition problem we suggest to choose the average word-length in time steps as timescale value for the fast context layer (Cf) and to choose the maximal length of the sequences as timescale value for the slow context layer (Cs). These results can perhaps get transferred to other problems, where one uses the average length of the fast dynamics and maximal length of the slow dynamics as the respective values.

The dependency that we found between the size of the architecture and the size of the problem is less desirable, but in line with experience from associator networks [32]. Further investigations should include the consideration of architectures that are dynamic in connectivity as well as in size. In addition, architectures should be tested with more complex scenes and verbal descriptions, including interrelations of multiple objects and embodied experience of a broader set of real world situations.

5.1 Conclusion

In conclusion our study supports that the embodiment of language in perception and a hierarchical structure with hysteresis mechanisms in terms of different timescales are important aspects of an appropriate architecture for language. For such an architecture a feasible constraint can be our mostly feed-forward but compositional structure, also suggested for the (visual) cortex [19].

We believe it is very important to intensively study further the architectural characteristics that both favour or hinder the emergence of language. More specifically, in addition to learning "that" language can be grounded in perception and be bound to experience we need to learn "how". For this we need to very carefully choose the assumptions that we are willing to make, to avoid to a) run into the *poverty of stimulus* (POS) pitfall [3] and b) research only into the parts instead of looking at the full system, which may result in more than the sum of the parts.

In the future we will further refine the architectural characteristics to identify the most important building blocks for natural language processing. The understanding of the brain's architecture for language can explain the humans' most important cognitive capability, but also can inform future software frameworks for service robots that should interact with and understand humans.

Acknowledgements. The authors would like to thank A. Saleh and H. Vöcking for support with the real world data acquisition, J. Dávila-Chacón and C. Weber for critical and inspiring discussions, as well as J. Bauer and C. Mönter for valuable revisions of early versions of this chapter.

References

1. Barsalou, L.W.: Grounded cognition. Annual Review of Psychology 59, 617–645 (2008)
2. Behrens, H.: Usage-based and emergentist approaches to language acquisition. Linguistics 47(2), 383–411 (2009)
3. Berwick, R.C., Pietroski, P., Yankama, B., Chomsky, N.: Poverty of the stimulus revisited. Cognitive Science 35(7), 1207–1242 (2011)
4. Borghi, A.M., Gianelli, C., Scorolli, C.: Sentence comprehension: effectors and goals, self and others. An overview of experiments and implications for robotics. Frontiers in Neurorobotics 4(3), 8 (2010)

5. Cangelosi, A.: Grounding language in action and perception: From cognitive agents to humanoid robots. Physics of Life Reviews 7(2), 139–151 (2010)
6. Cangelosi, A., Riga, T.: An embodied model for sensorimotor grounding and grounding transfer: Experiments with epigenetic robots. Cognitive Science 30(4), 673–689 (2006)
7. Cangelosi, A., Tikhanoff, V., Fontanari, J.F., Hourdakis, E.: Integrating language and cognition: A cognitive robotics approach. Computational Intelligence Magazine 2(3), 65–70 (2007)
8. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 8(6), 679–698 (1986)
9. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(5), 603–619 (2002)
10. Deacon, T.W.: The symbolic species: The co-evolution of language and the brain. W.W. Norton & Company (1997)
11. DeWitt, I., Rauschecker, J.P.: Phoneme and word recognition in the auditory ventral stream. Proceedings of the National Academy of Sciences 109(8), E505–E514 (2012)
12. Doya, K.: Recurrent networks: learning algorithms. In: Handbook of Brain Theory and Neural Networks, pp. 955–960. MIT Press (2003)
13. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys 2, 163–212 (1999)
14. Eisenbeiß, S.: Generative approaches to language learning. Linguistics 47(2), 273–310 (2009)
15. Elman, J.L.: Finding structure in time. Cognitive Science 14(2), 179–211 (1990)
16. Feldman, J.A.: From Molecule to Metaphor: A Neural Theory of Language. The MIT Press (2006)
17. Frank, S.L.: Strong systematicity in sentence processing by an echo state network. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 505–514. Springer, Heidelberg (2006)
18. Frank, S.L., Haselager, W.F., van Rooij, I.: Connectionist semantic systematicity. Cognition 110(3), 358–379 (2009)
19. Friston, K.: A theory of cortical responses. Philosophical Transactions of the Royal Society B: Biological Sciences 360, 815–836 (2005)
20. Gilbert, C.D., Li, W.: Top-down influences on visual processing. Nature Reviews Neuroscience 14, 350–363 (2013)
21. Harnad, S.: The symbol grounding problem. Physica D: Nonlinear Phenomena 42, 335–346 (1990)
22. Heinrich, S., Weber, C., Wermter, S.: Adaptive learning of linguistic hierarchy in a multiple timescale recurrent neural network. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part I. LNCS, vol. 7552, pp. 555–562. Springer, Heidelberg (2012)
23. Heinrich, S., Weber, C., Wermter, S.: Embodied language understanding with a multiple timescale recurrent neural network. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 216–223. Springer, Heidelberg (2013)
24. Hickok, G., Poeppel, D.: The cortical organization of speech processing. Nature Reviews Neuroscience 8(5), 393–402 (2007)
25. Hinoshita, W., Arie, H., Tani, J., Okuno, H.G., Ogata, T.: Emergence of hierarchical structure mirroring linguistic composition in a recurrent neural network. Neural Networks 24(4), 311–320 (2011)
26. Hoffmann, T., Trousdale, G.: The Oxford handbook of construction grammar. Oxford University Press (2013)

27. Iglesias, J., Eriksson, J., Pardo, B., Tomassini, M., Villa, A.E.: Emergence of oriented cell assemblies associated with spike-timing-dependent plasticity. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3696, pp. 127–132. Springer, Heidelberg (2005)
28. Iglesias, J., Villa, A.E.: Recurrent spatiotemporal firing patterns in large spiking neural networks with ontogenetic and epigenetic processes. Journal of Physiology-Paris 104(3-4), 137–146 (2010)
29. Jackendoff, R.: Foundations of language: Brain, meaning, grammar, evolution. Oxford University Press (2002)
30. Karmiloff, K., Karmiloff-Smith, A.: Pathways to language: From fetus to adolescent. Harvard University Press (2002)
31. Kullback, S.: Information Theory and Statistics. John Wiley, New York (1959)
32. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Orr, G.B., Müller, K.-R. (eds.) NIPS-WS 1996. LNCS, vol. 1524, pp. 9–50. Springer, Heidelberg (1998)
33. Mielke, A., Theil, F.: On rate-independent hysteresis models. Nonlinear Differential Equations and Applications NoDEA 11(2), 151–189 (2004)
34. Orban, G.A.: Higher order visual processing in macaque extrastriate cortex. Physiological Reviews 88(1), 59–89 (2008)
35. Pulvermüller, F.: The Neuroscience of Language: On Brain Circuits of Words and Serial Order. Cambridge University Press (2003)
36. Pulvermüller, F., Fadiga, L.: Active perception: sensorimotor circuits as a cortical basis for language. Nature Reviews Neuroscience 11, 351–360 (2010)
37. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the rprop algorithm. In: Ruspini, E.H. (ed.) Proceedings of the IEEE International Conference on Neural Networks (ICNN 1993), vol. 1, pp. 586–591. IEEE, San Francisco (1993)
38. Rohde, D.L.T.: A connectionist model of sentence comprehension and production. Ph.D. thesis, School of Computer Science, Carnegie Mellon University (2002)
39. Rohde, D.L.T., Plaut, D.C.: Connectionist models of language processing. Cognitive Studies 10(1), 10–28 (2003)
40. Roy, D.K., Pentland, A.P.: Learning words from sights and sounds: A computational model. Cognitive Science 26(1), 113–146 (2002)
41. Suzuki, S., Abe, K.: Topological structural analysis of digitized binary images by border following. Graphical Models and Image Processing 30(1), 32–46 (1985)
42. Tani, J., Ito, M.: Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 33(4), 481–488 (2003)
43. Wermter, S., Panchev, C., Arevian, G.: Hybrid neural plausibility networks for news agents. In: Ford, K., Forbus, K., Hayes, P., Kolodne, J., Luger, G. (eds.) Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999), Orlando, USA, pp. 93–98 (1999)
44. Widrow, B., Hoff, M.E.: Adaptive switching circuits. IRE WESCON Convention Record 4, 96–104 (1960)
45. Williams, R.J., Zipser, D.: Gradient-based learning algorithms for recurrent networks and their computational complexity. In: Chauvin, Y., Rumelhart, D.E. (eds.) Backpropagation: Theory, Architectures, and Applications. Lawrence Erlbaum Associates, NJ (1995)
46. Yamashita, Y., Tani, J.: Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. PLoS Computational Biology 4(11), e1000220 (2008)

Learning to Look and Looking to Remember: A Neural-Dynamic Embodied Model for Generation of Saccadic Gaze Shifts and Memory Formation

Yulia Sandamirskaya and Tobias Storck

Abstract. Looking is one of the basic sensorimotor behaviours, which entails representation of the visually perceived target and transformation of this representation in a motor signal, which moves the eye to center the target object in the field of view. Looking facilitates memory formation, bringing objects into the portion of the retinal space with a higher resolution. It also helps to align the internal representations of space with the physical environment. In this chapter, we present a neural-dynamic architecture, which integrates several processes, involved in looking, such as target selection, generation of motor signal, adaptation of gaze shift's amplitude, memory formation, scene exploration, and the coordinate transformations. We demonstrate the functioning of the architecture on a simulated robotic agent and provide a discussion of its implications in terms of neural-dynamic and cognitive modelling.

1 Introduction

When we look around a room, we don't even notice our own fast and frequent gaze shifts - saccades, - which scan the environment around us, bringing potentially interesting portions of visual input into our fovea for detailed examination. We, instead, have an illusion of a continuous perception of the room and objects in it, ready to be acted upon. We can direct actions at objects around us based on our memory, collected in the fixation periods between discrete saccadic eye movements. How does our neural system accomplish this task? This question has been asked for decades now [2, 12]. How may we build an artificial looking system with similar properties?

Let us follow the processes involved in looking and which our neural system uses to derive a useful representation of the body's surroundings from a sequence of saccades. First, the luminance of light, reflected from objects around us, induces

Yulia Sandamirskaya · Tobias Storck
Insititut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany
e-mail: {yulia.sandamirskaya,tobias.storck}@ini.rub.de

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_9

activity patterns on the retina of our eyes. This activity is projected onto the visual cortex and subcortical structures, in which the neuronal attentional mechanisms select a single target for the next saccade. Next, the precise and fast eye movement is generated and brings the interesting part of the visual input into the fovea. Falling onto the fovea, the features of the visual patch may be examined at a better resolution and the identity of the object in this location may be recognised. Moreover, at this moment, the location of the visual patch has to be stored in a useful way, i.e. so that it may be fused with information, collected from previous and upcoming fixations.

Several parts of this process have been examined experimentally, and both neurologically and behaviourally realistic models were developed. Despite the seeming simplicity of the looking system[1], the neural circuitry involved in saccades generation has an immense complexity. For instance, the selection of the saccade's target involves attentional processes, which include both bottom-up, saliency-based computation [17], and top-down influence of context, presence of distractors and alternative targets, memory, or task cues. The generation of a saccade towards the selected target, in its turn, is not trivial, since the variability in the neuromuscular system of the eye calls for a permanently running calibration process between the retinal frame of reference and the motor system [5, 15, 24]. Such calibration processes related to the control of gaze-shifts were found in cerebellar cortex and are hypothesised to be modulated by reward-related basal ganglia loops. Finally, the reference frame transformations between retinotopic, gaze-centred, head-, and body-centred coordinates are needed to fuse information between saccades, to generate saccades to remembered targets if intermediate saccades are involved (double-step saccades), as well as to directed, e.g., arm movements towards visually perceived targets. These processes are hypothesised to be located in the prefrontal cortex [9].

As summarized by Girard and Berthoz [13], several cortical and subcortical regions are involved in generation of the saccadic eye movements (or gaze shifts) in humans and primates. Each of these regions, in its turn, has a complex structure and the modelling and experimental work on untangling these structures – taking into account the behavioural and neurophisiological data – is far from being complete yet.

In this paper, we show how a neural-dynamic framework, based on Dynamic Field Theory, allows to implement an embodied, dynamic, autonomous, and adaptive model for looking behavior. We demonstrate the properties and function of the model by implementing the neural-dynamic architecture on an embodied (here, simulated) robotic agent, connecting its sensors (camera) and motors to the architecture. We introduce visual scenes to the robotic camera and observe the behaviour – both of the network and of the simulated robotic hardware. This approach enables a tight integration between modelling and behaviour analysis, linking the neural, architectural level with the behavioural level.

Our model integrates several functional components, which evolve around looking behaviour. In particular, the agent is able to calibrate itself and learns to look at objects based on a vision-based error-estimation module. The agent is able to

[1] Looking is probably the most basic behaviour, which arises on the intersection between the sensory input and motor control; compare it with arm movements, for instance.

adapt to abrupt changes in the sensorimotor plant or in the environment, as demonstrated in adaptation experiments. Moreover, the robot builds a memory of the observed scene, which integrates the feature information about the objects in the scene with spatial information. A transformation to a body-centred reference frame is performed to make memory independent of the current gaze direction of the camera. We deal with exploration of a scene and inhibition of return, which allows to scan the scene with several objects with different saliency. We demonstrate memory and double-step saccades, as well as simulate an experiment on saccadic adaptation.

Certainly, the model does not address all issues and does not tackle all problems, involved in understanding the saccade generating circuitry. However, it makes an important step towards this understanding, by, first, testing a framework, in which neuronal mechanisms may result in real world behaviour, bridging the mechanistic and behavioural levels. Critically, in this framework, the modelled neural circuitry may be simulated continuously in time and may be linked to noisy and varying real-world sensory inputs and effectors. Second, the framework allows to integrate many functional modules and thus study the saccadic system as a whole, in an integrative framework, instead of looking at its subsystems in isolation. Such a 'wholistic' approach provides additional constraints for modelling, which are set by considering the system as a whole, instead of looking at it as a sum of its components. Finally, the functionally-driven approach allows to look beyond a mechanistic devision of the saccadic system according to neurophysiological units, found in the brain, and allows to consider solutions, in which particular function is served by several such units and is not localised to a particular brain structure. We believe that such functional models will ultimately shed light on how the brain solves the problem of generating precise saccades.

The model, presented here, is not claimed to be complete and we also didn't aim at relating its elements to the neuronal structures, although such mapping will be exemplarily done in the discussion, but is an initial presentation of the neural-dynamic modelling framework with, hopefully persuasive, demonstration of its power to understand the neural circuitry in behavioral terms. The architecture may also have technical application, being a proof-of-concept demonstration of a self-calibrating robotic system, similar to a model presented in [3], but with an increased autonomy, which learns to look at objects in its environment and autonomously update the involved internal sensorimotor mappings when needed.

2 Methods

The architecture for generation of adaptive looking behaviour, which we present in this chapter, is built within Dynamic Field Theory (DFT) – a mathematical and conceptual framework for modelling cognition based on population-level descriptions of neuronal dynamics [1, 33]. In DFT, continuous activation functions, called Dynamic Neural Fields (DNFs), represent different perceptual and motor variables, critical for behaviour of an embodied agent. The DNFs may be coupled through mappings, analogous to weight matrixes of neural networks, which are subject for adaptation and learning. Critically, DNFs follow the dynamics, which allows for localised attractor patterns – peaks of activity, which make DNF architectures behaviourally robust – the behaviourally relevant states of the dynamics correspond to attractors and thus the correct behaviour may be guaranteed.

Here, we present the main functional and structural units of DFT, used in the architecture, whereas a more thorough discussion of the DFT framework may be found in [32, 29].

2.1 Dynamic Neural Fields

Dynamic Neural Fields describe activity of populations of biological neurons in terms of continuous functions, defined over behavioural parameters, to which the neurons are sensitive. These continuous functions are descriptions, which abstract away the discrete nature of individual neurons, as well as the spiking character of their activity. A differential equation (1) specifies how the activation functions evolve in time in DFT, as analysed by [42, 14, 1].

$$\tau \dot{u}(x,t) = -\,u(x,t) + h + \int f(u(x',t))\,\omega(|x'-x|)\,dx' + I(x,t). \tag{1}$$

In Eq. (1), $u(x,t)$ is the activation of a dynamic neural field (DNF) at time t; x is one or several behavioral parameters (e.g., color, pitch, space, or velocity), over which the DNF is spanned; τ is the relaxation time-constant of the dynamics; h is the negative resting level, which defines the activation threshold of the field; $f(\cdot)$ is the sigmoidal non-linearity shaping the output of the neural field when it is connected to other fields or self-connected; the self-connections of DNFs are shaped by a Mexican hat lateral interaction kernel, $\omega(|x'-x|)$, with a short-range excitation and a long-range inhibition; $I(x,t)$ is the external input to the DNF from the sensory systems or other DNFs.

Equation (1) defines an attractor for the activation function, which is determined by the external input, $I(x,t)$, the resting level of the field, h, and the lateral interactions, specified by the kernel, $\omega(|x-x'|)$. Critically in the DFT framework, a distinctive type attractor of a DNF is a localized activity peak, which may be “pulled up” by the lateral interactions from a distributed input with inhomogeneities. Such peaks of activation are units of representation in Dynamic Field Theory [33]. Because of the stability and attractor properties of the DNF dynamics, cognitive models formulated in DFT may be coupled to real robotic motors and sensors and were shown to generate cognitive behavior in autonomous robots [32, 6]. In particular, activity peaks stabilise decisions about detection of a salient object in the visual input and selection among alternatives, the stabilised representations are critical to linking to motor control and performing cognitive operations on representations [29].

2.2 Autonomous Control in DNF Architectures

A single DNF converges to an attractor – one or several localised activity peak(s) – in response to its inputs and would stay in this state unless inputs change, bringing about a different attractor. In this "passive" mode, a DNF could serve as a detection mechanisms, but wouldn't be able to autonomously control actions of an agent. To overcome this limitation, the DFT framework has been extended recently with dynamical structures, which enable activation and deactivation of DNFs depending not only on the sensory inputs, but also on the cognitive task, which the agent faces [31, 27]. These dynamical structures are zero-dimensional DNFs, or dynamical nodes, with an activation following the Equation (2).

$$\tau\dot{n}(t) = -n(t) + h + cf(n(t)) + I(t). \tag{2}$$

The dynamics of activation function $n(t)$ of a dynamical node is equivalent to dynamics of a zero-dimensional DNF. This dynamics is bi-stable: the node can be in an active (i.e., activation is above the threshold, defined by the sigmoidal output function f) or in an inactive state. An active state may be sustained event if the external input, $I(t)$, which initially activated the node, decreases below the activating levels. The self-excitatory term with a strength c accounts for this behavior. If the self-excitation parameter is high enough, the node may stay active even if the initial input, which caused its activation, ceases completely. In this case, an external inhibitory input is needed to deactivate the node.

For autonomous control of DNFs, two such nodes are introduced for every elementary (cognitive) behaviour in the agent's repertory: an *intention node* activates DNFs, which bring about a particular (motor or intrinsic) action, and a *condition of satisfaction node* is activated when the desired action outcome is perceived to be achieved and the intention can be inhibited [30].

In our looking architecture, these nodes play an important role in starting and finishing different phases of the gaze shift – the saccade initiation and termination, activation of the fixation, learning, and memory formation processes.

2.3 Learning with DNFs

The basic learning mechanism in the DFT is the formation of memory traces of positive activation of a DNF [41]. The memory trace – called *preshape* in DFT – is a dynamical layer, which receives input from the respective DNF and projects its output back to this DNF. The memory trace projection facilitates activation of the DNF at previously activated locations (positive preshape), or inhibits DNF at previously activated location (negative preshape, which accounts for habituation and exploration). The preshape layer follows the equation (3), [29].

$$\begin{aligned}\tau_l\dot{P}(x,t) = \lambda_{build}\Big(-P(x,t) + f\big(u(x,t)\big)\Big)f\big(u(x,t)\big) - \\ -\lambda_{decay}P(x,t)\Big(1 - f\big(u(x,t)\big)\Big).\end{aligned} \tag{3}$$

Here, $P(x,t)$ is the strength of the memory trace at site x of the DNF with activity $u(x,t)$ and output $f\big(u(x,t)\big)$, λ_{build} and λ_{decay} are the rates of build-up and decay of the memory trace. The build-up of the memory trace is active on the sites with a high positive output $f\big(u(x,t)\big)$, the decay is active on the sites with a low output. The memory trace $P(x,t)$ is an additive input to the DNF dynamics.

Two DNFs may be coupled through a higher-dimensional memory structure, similar to a weight matrix in the standard neural networks. In DFT, such weight matrix is adapted through the mechanism of memory trace formation, which is in this case equivalent to a Hebbian learning process. The coupling is strengthen between locations in two DNFs, which are activated simultaneously, according to Equation (4).

$$\begin{aligned}\tau\dot{W}(x,y,t) = \varepsilon(t)\Big(-W(x,y,t) + f(u_1(x,t)) \times f(u_2(y,t))\Big)\cdot\\ \cdot\Big(f(u_1(x,t)) \times f(u_2(y,t))\Big). \qquad (4)\end{aligned}$$

Here, the weights function, W(x, y, t), which couples two DNFs, $u_1(x,t)$ and $u_2(y,t)$ has an attractor at the intersection between positive outputs of the DNFs. The intersection is computed as a sum between the output of u_1, expanded along the dimensions of the u_2, and the output of the u_2, expanded in the dimensions of the u_1, augmented with a sigmoidal threshold function (this neural-dynamic equivalent to the Kronecker product is denoted by the $\times$ symbol). Learning is only active at the intersection of the active regions in fields $u_1(x,t)$ and $u_2(y,t)$ to prevent spontaneous forgetting of previously learned associations. The shunting term $\varepsilon(t)$ limits learning to time intervals, specified by the autonomous control of the overall architecture, as will be exemplified in our model.

The learning process is functionally robust if the coupling is updated only when the two input DNFs are in a correct, behaviourally meaningful, state. In the looking architecture, presented here, we combine the elements of intentionality with learning dynamics to demonstrate how the sensorimotor mapping, involved in looking behaviour, may be autonomously learned.

2.4 DFT in Modelling Saccade Generation

In our work, we rely on several properties of DFT, which already have been probed in modelling certain aspects of saccades, in particular the target selection and time course of decision making in saccade preparation. The following models are particularly relevant to our work, since they use the same mathematical model for the layer, which performs selection of saccadic targets and thus our model "inherits" the properties, established for this layer.

The first model, which used a dynamic neural field as a layer for target selection in a saccades generation system was introduced by Kopecz and colleagues [18]. The planned eye movement was represented in a neural field with visual and task-related (pre-) input converging on this field. The model could account for a transition from averaging between two presented targets to precise saccades to one of the two targets depending on the distance between the targets and the strength of the memory for the two locations from preceding saccades. The architecture features an active fixation system, similar to the one we use in our model.

The effects of lateral interactions in superior colliculus (SC) on the saccade reaction time were studied by Trappenberg and colleagues [38] in their work, which provides a link between behavioural studies of saccade generation and the underlying neural substrate. This model considers integration of different input sources for the selection of a single target in SC with a dynamic neural field and may account for experimentally observed delays in saccade initiation. Behavior of the model closely resembles activity of neurons and the model can account for many observations related to saccade initiation.

Willimzig et al [40] has also studied the time-course of saccadic decision making within dynamic field theory. To initiate a saccade, the system has to overcome fixation, which may be more or less difficult depending on the fixated object and the overall scene. The transition from fast averaging saccades to more time-lagged selective saccades is again demonstrated here using DNF layer for target selection. A similar DNF framework for movement representation was used in modelling preparation of arm movements [7].

These previous models have demonstrated the power of DFT in preparation and planning of saccades. Here, we extend these models to an architecture, which may actually realise the planned saccades, check for their accuracy, and adapt saccades' amplitude if needed.

3 The Model

3.1 The Overall Architecture for Looking

Figure 1 depicts the overall architecture for looking. It is a fairly complex network, since it accomplishes several functions apart from generating gaze shifts towards visually perceived targets. The architecture may be described in terms of the following interconnected modules. The *perception* module integrates over time and stabilises the visual input from a simulated robotic camera, as well as the proprioceptive input from the motor system of the simulated robotic agent. The precise saccades to the visually perceived targets are generated in the *saccade generator system*. The amplitude of the saccades for every retinal position and current gaze angle is learned in a set of gain maps within the *learning* module. There is one set of gain maps for each of the two motors of the robot. Next, the *fixation system* tracks the object between successive saccades and triggers memory processes in the *memory system*, which, on the one hand, steers exploration of the scene, decreasing the competitive advantage of those object in the scene, which are already put to memory, and, on the other hand, creates an allocentric representation of the objects in the scene, which can be used to generate saccades from memory, double step saccades (using the *planning saccades system*), or arm movements towards the target. Next, we will walk through the most important parts of this architecture, explaining how different functions of the network are brought about.

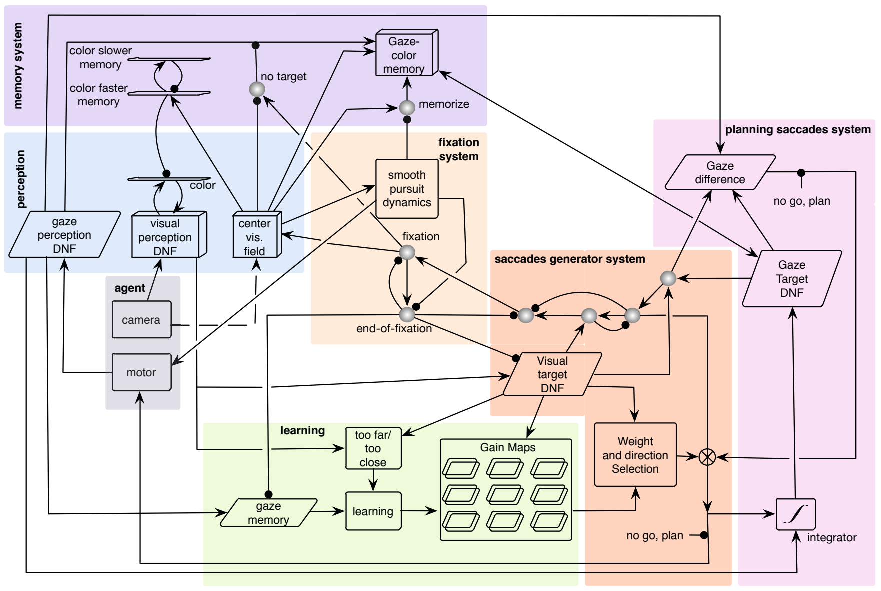


Fig. 1 The overall architecture

3.2 *Perception*

The perceptual system consists of a three-dimensional *visual perception DNF*, which spans the dimensions of color and the retinotopic space (modelled to be a cartesian space here, but a polar version with foveal expansion is possible as well). The RGB output of the robotic camera is split into three color channels – hue, saturation, and value. The saturation channel is used to perform the basic figure-ground segregation and create a course saliency map, which highlights regions in the visual space, for which the hue value is extracted and input to the visual perception DNF. The hue (color) dimension is additionally stabilised by a coupled one-dimensional color DNF. The visual perception DNF requires this supporting input to form a stabilised activity peak over a selected object in the scene. This support from the color field is suppressed for objects, which are already stored in memory. Thus, such objects have a disadvantage in the competition to be selected for the next looking act[2].

The *center of visual field* is another three-dimensional DNF, which receives input from the central portion of the camera image. This field may only build activity peaks from the camera input when the *fixation node* is active and provides an additional boost to this DNF (i.e., raises its resting level), signalling that a saccadic gaze shift has been finished. An activity peak in the center of visual field DNF activates

[2] This ‘habituation’ happens along color dimension, but habituation along retinal space is also possible [36], both processes have to be balanced to account for human looking data. Here, we keep this system simple since accounting for experimental data is not the focus of the work reported here.

two processes in the architecture: memory formation and smooth pursuit dynamics, briefly described in the next section.

The *gaze perception DNF* receives input from the motor system of the robotic agent and represents the current proprioceptive state of the motor system.

3.3 Motor Control

The motor system of the agent consists of two motorised joints – the pan and tilt joints of the camera-head unit. The pan joint corresponds to rotation of the camera head around the vertical axis between the two cameras of our robot (only one of the cameras was used in this work; the camera's optical axis is not crossing the axis of the pan joint). The tilt joint corresponds to the incline of the camera – a rotation around the horizontal axis (which is also not aligned with the image plane of the cameras). Note that the arrangement of the motor rotation axes and the camera image provides for a non-linear mapping between the reference frame of the camera image ('retinal' reference frame) and the reference frame of the motor system.

The two motors – the pan and tilt motors – are servo motors, which may receive both position and rotation speed commands. We have used speed control in this work to make the motor system somewhat confirm with the biological motor systems (still, buying into significant simplifications compared to muscle control). The two motors coarsely correspond to the horizontal and vertical control of the eye. The rotation of the eye-ball was not modelled here.

During saccades, the saccade generator system sends velocity commands directly to the two motors of the camera head. During smooth pursuit movement, a dynamical system, which has an attractor at the visually perceived target, provides the velocity input for the two motors and performs visual servoying around the target object. The visual servoying towards the target is not possible during saccadic gaze shifts, since they are performed too fast for the visual processing to influence their course.

Next, we will describe the system, which may generate correct velocity commands based on the visual perception of the target in the retinal frame of reference.

3.4 Saccade Generator

Since the saccades (in humans and primates) are very fast movements[3], the neural system needs to generate the complete velocity profile, which will bring the eye's fovea onto the target. In our architecture, this velocity profile is generated by a neural oscillator, as described next. In particular, the neural oscillator is part of a central control unit, which generates the saccades gaze shifts and controls the temporal dynamics of the overall architecture. This unit consists of six dynamical nodes, depicted in Figure 2.

[3] Which ecologically makes sense, to minimise the time when the eye moves and both vision and calibration with the outside world are disturbed.

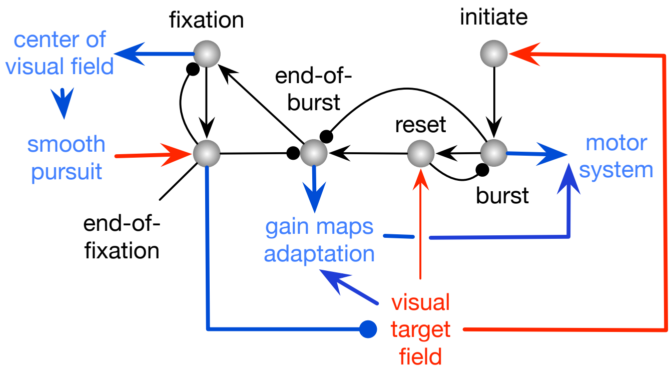


Fig. 2 A neural circuit, which generates the gaze-shift velocity profile. Grey shaded circles denote the six central nodes, which control gaze shift generation. Black arrows show excitatory connections between the nodes, lines with filled circles – the inhibitory connections. Blue arrows are outputs of the system to other parts of the architecture, red arrows are inputs to the nodes structure.

The *burst* and *reset* nodes constitute the neural oscillator. If a constant input is provided to the burst node, this pair of nodes gets activated and deactivated in alternation: the burst node activates the reset node, which in its turn deactivates the burst node and consequently looses its own activation, and the cycle repeats. In our setting, however, this system, generates a single "oscillation" in the following way. The reset node is lifted to be self-sustained by the input from the *visual target field* and only looses its activation when the whole gaze-shift action is finished (i.e. when the visual target field is inhibited by the *end-of-fixation node*). The active reset node keeps the burst node inhibited until a new target is selected for the gaze shift.

Activation of the burst node drives the motor system of the agent by sending velocity commands to both the pan and tilt motors. The amplitude of the burst specifies the peak velocity of the gaze shift and, implicitly, its amplitude. The busts's amplitude is set by the adaptive *gain maps* – one for the horizontal movement, generated by the pan motor, and one for the vertical movement, generated by the tilt motor. The adaptive dynamics of the gain maps will be presented in Section 3.5.

The burst node is activated by the *initiate gaze shift node*, which is effectively the intention node of the gaze shift behaviour. The *end-of-burst node* detects when one burst is finished and activates the *fixation node*, which, eventually, drives the fixation and the smooth pursuit systems of the architecture. When the fixation system brings the object in the center of the visual field and the memory for this object is updated, the *end-of-fixation node* is activated and resets the gaze shift elementary behaviour by inhibiting the visual target field, which provided the initial input to this system.

Figure 3 shows the time-course of activation of the five nodes from Figure 2 for two subsequent gaze shifts, demonstrating how the autonomous organisation of the architecture works – like a clock, turned on by the target field when a new target to

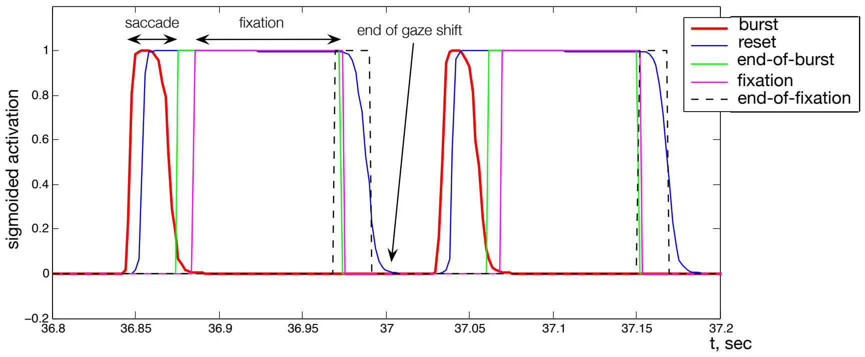


Fig. 3 A sequence of two gaze shifts, each consisting of a saccadic part, driven by the Burst node (red line shows sigmoided activation of the Burst node), and a fixation part, in which the target object is tracked, memory is formed, and adaptation is performed if needed (magenta line shows activation of the Fixation node), the gaze shift is finished when the End-of-fixation node is activated (dashed black line) and the Reset node (green line) is inhibited, releasing the Burst node from inhibition, which may now generate the next saccade.

look at is detected and turned off by the fixation dynamics, when the target is centred in the camera's view and its location and features (color) are stored in memory.

3.5 *Saccade Amplitude and Gain Maps*

The peak velocity of the saccadic gaze shift is defined by the values, stored in the *gain maps*, which are initially learned and are constantly adapted in a learning process after each gaze shift. Each gain map is defined over the same space as the visual target DNF – the retinotopic (here, camera image) space. There is one gain map for each starting gaze angle, i.e. for every possible state of the eye (here, of the camera motors). This means that the gain map is a four-dimensional structure, with two visual (retinal) dimensions and two motor dimensions. Further, there is one such map associated with each of the motors of the system (the pan and tilt motors).

The maps control the precise vector of the saccadic gaze shift as follows. The output of the visual target field is multiplied with a slice of the four-dimensional gain map, selected by the currently perceived (before the gaze shift) gaze position (the pan and tilt values stored in the *gaze memory DNF*). When a peak is built in the visual target field, this multiplication effectively selects a region on the gain map, which is specified by the location of the activity peak in the visual target field. Thus, the gain maps function as synaptic weights between the visual target field and the circuitry, which generates motor commands. Moreover, these synaptic weights are modulated by the input, which specifies the gaze angle before the saccade. Formally, the gain maps constitute a tuneable (or steerable) map between the retinal and the motor frames of reference [9]. The result of the multiplication of the output of the visual target field with the gain map is integrated and is connected to the output of the *saccade generator system* (Figure 2), amplifying the amplitude of the burst of the neural oscillator. Figure 4 illustrates this process.

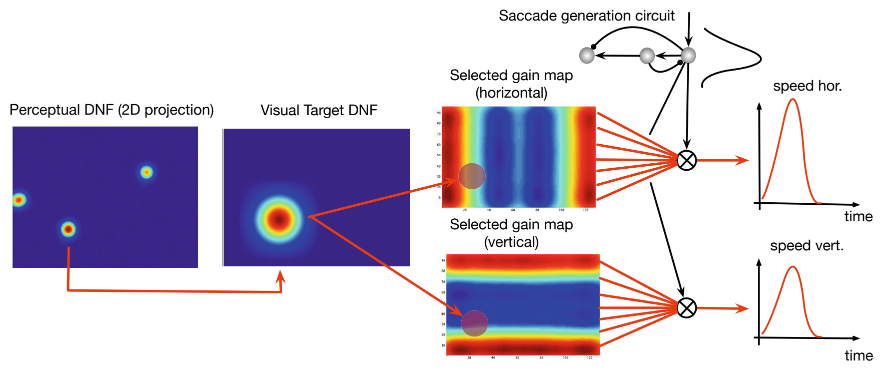

Saccade generation circuit
Selected gain map (horizontal)
Perceptual DNF (2D projection)
Visual Target DNF
speed hor.
time
Selected gain map (vertical)
speed vert.
time

Fig. 4 Circuitry to generate saccades with a precise amplitude

The gain maps are initially homogeneous – all values in them are set to ones. When a visually perceived target appears in this state, a saccade is generated with such an amplitude in both motors that does not bring the target object into the central portion of the camera image (retina). Thus, the fixation system does not get engaged, but the error estimation module is activated and estimates whether the saccade was too long or too short (*too far / to close* module in Figure 1) in each of the four direction in the image: left, right, up, and down (corresponding to four direction of the eye movements' 'synergies'). The learning mechanism of Equation (5) updates the gain map at the position, which corresponds to the active region of the visual target field and the gaze angle before the saccade. The direction of adaptation (increase of decrease of the values in the selected region of the gain map) is defined by the output of the error estimation module.

$$\tau_l \dot{G}^{h,v}(x,y,k,l,t) = \varepsilon^{h,v}(t) f(u_{EoS}(t)) \Big(f(u_m(k,l,t)) \times f(u_{tar}(x,y,t)) \Big). \qquad (5)$$

Here, $G^{h,v}(x,y,k,l,t)$ are two sets of gain maps (for the '*h*orizontal' and '*v*ertical' components of movement). Each of the $k \times l$ gain maps in the two sets is defined over the dimensions of the visual target DNF, $u_{tar}(x,y,t)$. Each set spans $k \times l$ different initial motor states (the pan and tilt joint angles in our setup). The gains change in the map(s), which are selected by the output of the motor DNF, $f(u_m(k,l,t))$, at the locations, which are set by the activity peak in the target DNF. $f(u_{EoS}(t))$ is the output of the end-of-saccade node, which is required to be positive (saccade finished) for learning to become active. $\varepsilon^{h,v}(t)$ is the error in each of the movement components, τ_l is the learning rate.

Thus, after each unsuccessful saccade the gain maps are corrected slightly in a localised region. After sufficient experience with looking at visual targets in different locations in the image and from different starting gaze angles, the complete gain maps are learned and the system is able to perform precise saccades from any configuration. The maps are updated locally over a few gaze-shifts if an unexpected change in the sensorimotor plant happens.

3.6 Memory Formation and Exploration

After a precise saccade, the target object falls into the central part of the visual field, as detected by the *center of visual field DNF*. The object is being tracked now by the *smooth pursuit dynamics*, which sets an attractor at the visually perceived target and visually servoys the camera at the objects. When the object is centred in the visual field, its memory is formed in the three-dimensional *color-gaze memory field*, which is critical for creation of an allocentric memory for the object and for generation of memory saccades. The part of the architecture, which forms memory during fixation and biases the perceptual system during exploration is shown in Figure 5.

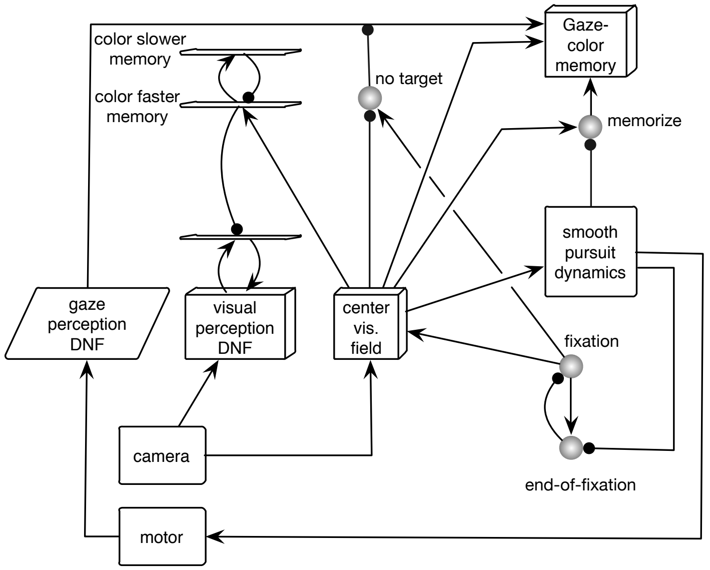


Fig. 5 Part of the architecture responsible for memory formation and exploration (habituation)

Here, the activated *center of visual field DNF* activates the smooth pursuit dynamics, which fixates the object in the center of visual field, even if the object moves. As long as movement is generated by the smooth pursuit dynamics – i.e. the object is not yet centred in the visual field and its location in gaze coordinates is not stable, – the memory formation process is suppressed by a lacking boosting input from the *memorise node*. When the object is centred in the visual field of view, the memory formation process is activated in the three-dimensional *color-gaze memory DNF*, which forms an allocentric memory of the objects in the visual scene, represented in gaze-coordinates. If there is no object perceived in the center of visual field (the *no target node* is activated), the corresponding location in the color-gaze memory DNF is inhibited. If there was an object memory stored at this location, its memory representation ceases.

On the other hand, the *center of visual field DNF* boosts the *color faster memory field*, which builds a memory representation of the color of the currently observed object. This representation inhibits the visual perception DNF along color dimension for the subsequent saccades. Thus, when the next saccade target is selected, the colors that are already in memory have less chances to induce a peak in the perception DNF. The *color slower memory* builds up preshape activity hills on a more slower time scale. These preshape hills eventually delete the peaks from the faster color memory field, releasing the respective color to participate in the target selection during exploration.

3.7 Prediction and Memory-Driven Saccades

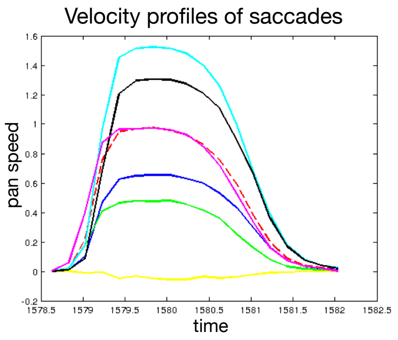
On the right side of Figure 1, the planning saccade system is depicted. In this system, the saccades' targets are represented in the body-centred gaze coordinates. This gaze-target DNF receives input either from the gaze-memory DNF, if saccade is to be performed to a memorised object, or from an integrator, which predicts the gaze coordinate of a visually perceived target without performing a saccade. The latter path allows the system to perform double-step saccades, when two targets are presented to the system and the agent has to saccade to them in a sequence only when both targets are no longer visible. In this case, the system "simulates" two saccades from the fixation point and stores the predicted gaze angle after each such virtual saccade. These stored gaze-representations of the targets' locations can then be used to generate eye movement towards both targets, even though the retinal representation of the second target would shift after the first saccade.

4 Results of Simulated Experiments

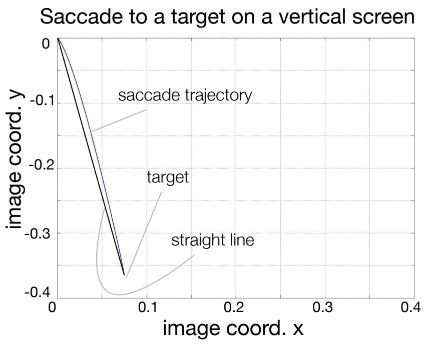
The architecture, sketched in Figure 1, was implemented using an open-source C++ framework *cedar*, `www.cedar.ini.rub.de` [20], which allows to build neural-dynamic architectures using a graphical user interface and simulate them efficiently on a conventional computer. The simulation basically consists in solving the coupled differential equations (the core equations, which form the building units of the architecture, were presented in Section 2 of this chapter) using Euler method. The *cedar* framework also offers an interface to robotic hardware, including cameras. The interface may be used both with real and simulated robots. In this work, we have used a simulated CoRa robot [16], in particular its camera, mounted on a motorised pan-tilt unit. Although *cedar* offers plotting routines to visualise the architecture during its function for monitoring purposes, we have collected the data from simulation and have analysed and plotted it offline, using standard MATLAB routines. The data consists of matrices of activation of the dynamic neural fields and nodes of the architecture. Here, we exemplify some of the architecture's functions based on this collected data from simulated experiments.

4.1 Gaze Shifts Generation

Saccades, generated by humans or primates, have particular properties, which are well-studied in a controlled experimental settings. Here, we demonstrate that the basic properties of real saccades may be replicated in our system. Note that the architecture was not tuned to model the dynamic properties of human saccades, but still the trajectories and velocities of saccades resemble the respective behavioural plots. Figure 6a shows examples of velocity profiles, generated by the neural-dynamic gaze-shift generator in the simulated robot for gaze shifts of different amplitudes. Note the constant duration of the gaze shift, as typically observed in eye movement studies, and the varying peak velocity for saccades of different aptitudes. This property is part of the 'main sequence', postulated in studies of the primates' saccade generating system.



(a) Generation of saccades of different amplitudes. Demonstration of the relation between the peak velocity and amplitude of saccadic gaze shifts.



(b) Trajectory of an oblique saccade.

Fig. 6 Illustration of the basic properties of the saccade generating system

Figure 6b shows the trajectory of an eye movement (projection of the gaze direction on a vertical plane, in which the target is presented) towards a target, which requires activation of both the vertical and horizontal movement systems. The trajectory is very close to a straight line, as observed experimentally. In our architecture, this property is achieved by a mechanisms, similar to the mechanism proposed by [35] for generation of two-dimensional saccades: both horizontal and vertical movements are produced by the same neural burst generator, which output is scaled differently for the two components of the movement. This results in almost straight oblique saccades, when both the target and the gaze direction are projected on a vertical plane. The shape of saccade trajectories in 3D, when targets are located on a horizontal plane at different distances, is yet to be established experimentally and can be simulated in our framework.

4.2 Scene Exploration and Memory Formation

Figure 7 demonstrates how a visual scene (here, consisting of three coloured objects) is explored by the system.

On the left, four snapshots from the robot simulator show the simulated camera image and the visual scene in front of the robotic camera head. In the first snapshot, the robot observes the scene with three objects. The red object induces the largest blob in the camera image (since it is closer to the robot on the table) and is selected by the visual perception DNF as the saccade's target. The robot performs a saccadic gaze shift followed by a fixation dynamics towards the red object (second snapshot). Now, only the yellow object is visible in the camera image and is selected by the robot for the next gaze shift. When the yellow object is centred in the camera, both blue and red objects are visible (third snapshot). Although the red object is more salient in the camera image again, the blue object is selected for the next saccade, because of the inhibitory influence of the memory on the selection dynamics in the visual perception DNF. Finally, the blue object is fixated by the system (forth snapshot) and its representation is stored in the gaze-based scene representation.

On the right of Figure 7, the summed activations of the saliency in the camera image and the visual target DNF are shown. Note that during exploration, many locations have significant saliency in the camera image (in fact, more than shown in the figure – these are only the regions, which where active for longer periods of time). In the visual target DNF, to the contrary, only the locations selected for the saccade targets leave traces. The third plot shows the gaze-based (body-centred) representation of the visual scene, built-up during scene exploration: after each successful saccade, the gaze angles of the camera head are stored, which correspond to the object in the scene, i.e. which bring the object in the center of camera field of view (if the body of the robot does not move relative to the scene). The forth plot shows the projection of line of sight of the camera on the table surface during the experiment, as viewed from above. These projections show the scan paths of the camera head over the scene.

In this demonstration, it may be seen how a visual scene triggers a sequence of saccades in the system. Each object is fixated by the camera in a succession and the gaze angle of the robot during fixation is stored as a self-sustained peak in the memory field. The resulting memory representation may be used to direct saccades to memorised, but currently not observed objects, as well as used for control of movements, generated by other effectors of the robot (e.g., reaching), even if the robot looks away from the object (e.g., to look at the arm or the next object in a longer sequence of actions).

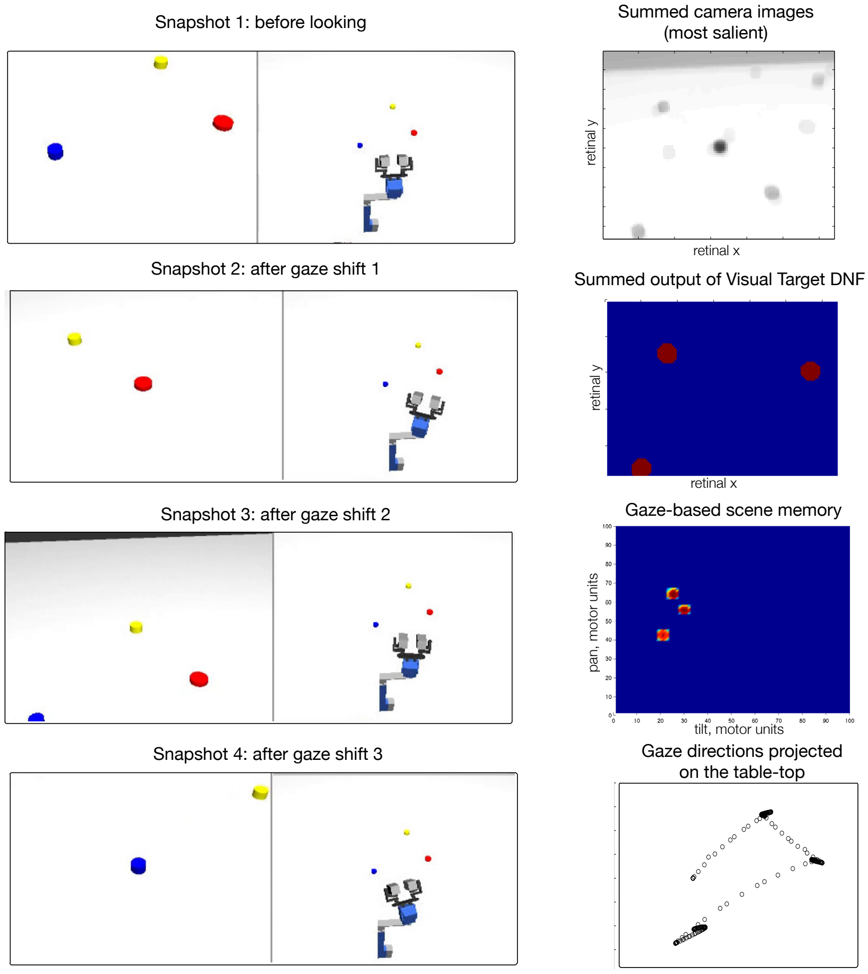


Fig. 7 Left: Simulated robot exploring the scene. **Right**: the summed visual input to the architecture, the summed activity of the visual target DNF, gaze-based memory of the scene at the end of exploration, and the gaze-trajectory, projected on the table-top (sampled at 100 frames per second).

4.3 Gain Maps Learning

The precise gaze shifts, demonstrated in the previous experiment, are the result of a learning processes, in which the gain maps, which specify the saccades' amplitudes, are adapted. Figure 8 shows the convergence of the learning process for a single location in one of the gain maps, in which the initial error of more than 3 cm is reduced over a few saccades (five here) to values below 0.5 cm.

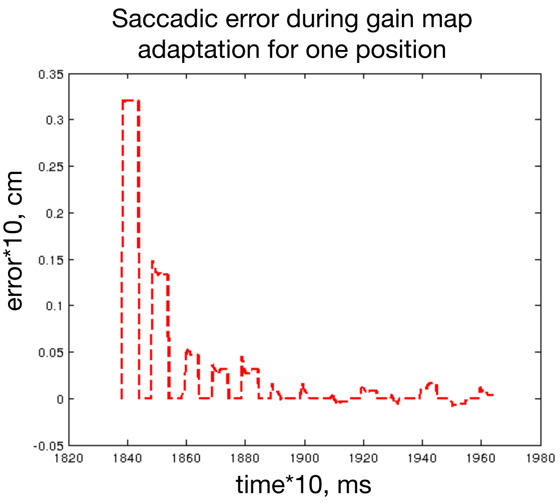


Fig. 8 Convergence of gain map for a single location

The two four-dimensional gain maps, learned for the horizontal and vertical movements are shown in Figure 9. The maps reflect the geometry of the robot and implicitly encode the amplitudes of the pan and tilt shifts for different shifts in the retinal frame of reference. Note that the mapping between the image and motor coordinates is non-linear here and changes significantly with initial gaze angle of the camera head. Especially with changing initial tilt-configuration, the amplitude of the motor signal changes for the same shift in the retinotopic coordinates. The pan-configurations play a lesser role in our robotic architecture, since changes in initial pan do not change the mapping between shifts in retinal and motor coordinate frames much.

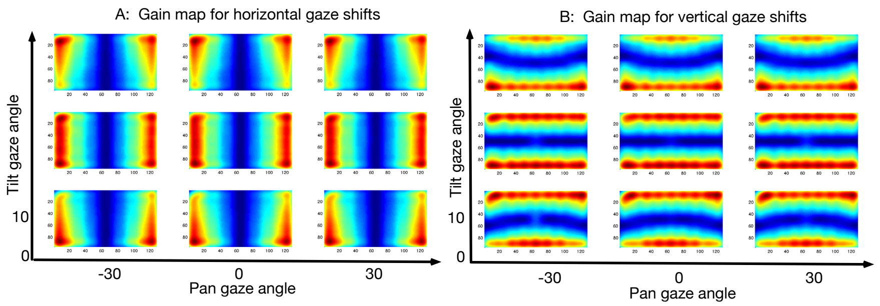
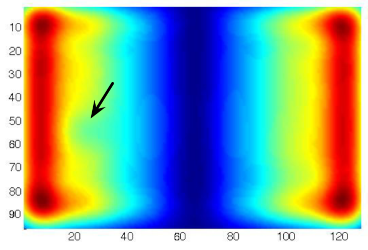


Fig. 9 Gain maps learned by the system. Nine slices are shown for three selected pan and tilt values of the initial pose (gaze angle) of the camera head.

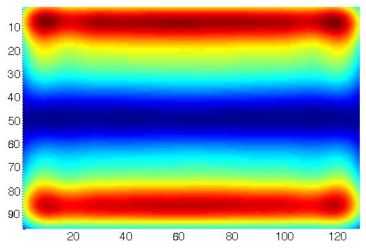
4.4 Modelling Adaptation Experiments

Adaptation of the amplitude of the saccadic gaze shift, as demonstrated in [24] is exemplified in Figure 10. Here, the robot first learned the complete gain maps and was able to perform precise saccadic gaze shifts from any starting configuration. In a scenario, which simulated the adaptation experiment, the target object was shifted horizontally during the saccadic gaze shift, so that the saccade landed (in the case, shown here) behind the target, i.e. the saccade was too long. The perceived error was used to updated the gain map, similarly as during the initial learning process, so that the new, adapted, gain map generated a saccade of the amplitude, which brought the shifted target in the fovea.

The adaptation is only effective for a localised region both in terms of retinal location of the target and the gaze angle prior to the saccade. In combination, the adaptation generalises to a region in the allocentric (here meaning gaze-angle independent) reference frame. This result is conform with recent experimental studies [44].



(a) Map for horizontal gaze shift



(b) Map for vertical gaze shift

Fig. 10 Gain maps adapted at one location. The adapted region is marked with the arrow.

The time-course of the simulated adaptation experiment is shown in Figure 11. This figure shows the increase of saccadic error at the offset of the adaptation experiment (seventh saccade shown in the figure) and a gradual decrease of the error back to the optimal level in the course of several saccades.

5 Discussion

This paper has introduced a computational framework and a neural-dynamic architecture for generation of adaptive looking behaviour in an embodied agent. The behaviour and the computational network share several characteristics with the human looking system.

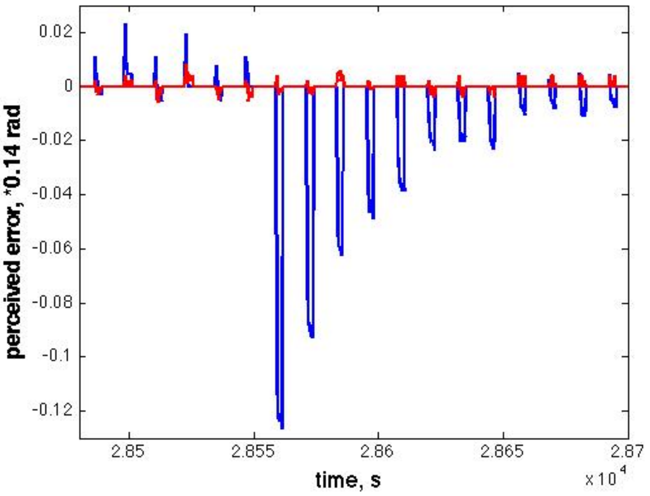


Fig. 11 Time course of adaptation: error magnitude for the adapted location over the course of adaptation experiment

5.1 Strengths and Limitations of the Architecture

The power of the framework we used in our modelling is in its dynamics. Thus, the introduced architecture is a process model and allows to model not only the structure, but also the dynamics of neural processes within this structure. Moreover, this dynamics is autonomous and embodied, which means that it may be connected to real sensors and motors and produce behaviour in real time.

The dynamic fields theory provides for stability of the building units of the architecture and enables coordination between different subsystems.The stability property of the dynamic neural fields has been established theoretically [1, 14, 42] and leads to robust, controllable, behaviour. The behaviourally relevant states of the neural system are represented in this framework as attractors, which persist long enough to have impact on the downstream structures. Transitions between attractors are instabilities and their course is autonomously controlled by an interconnected set of dynamical nodes, which organise behaviour of the architecture in time.

In the architecture, many different functional subsystems are integrated, some of them were developed in recent years in the DFT framework, others are introduced here for the first time (e.g. the adaptive weights coupled to a neural oscillator, the motor-based scene memory, the error-detection network, the coordination between smooth pursuit dynamics and saccade generation). Several functions of the looking system are implemented in our model, such as memory formation, formation of allocentric scene representation, habituation, scene exploration, adaptation, and learning. Although not all properties of looking behaviour in humans and primates have been accounted for in our current system, the framework has a potential to be extended and refined while keeping the behavioural stability of the overall network.

Another important characteristics of our architecture is that it is a function-based model. This allows to avoid narrowing modelling too much on single brain areas, as has been advocated early on in studies of the saccadic system [39, 13]. Thus, the architecture presented here is a behavior-based, functional model, which may be mapped on the substrate of neural circuits, involved in generation of eye movements. This mapping onto neuronal substrate, however, even for a single functional module of our architecture requires considering interactions among different neural circuits. This avoids a simplifying view, when one cortical or subcortical region is made responsible for a single function and every function is assigned to a single brain region. Since our architecture was inspired by the neural and behavioural findings about biological saccadic systems, we will provide a brief discussion of the neural plausibility and relevance of our model in Section 5.2.

Several properties of the saccadic system have not been modelled here. Thus, our current system cannot produce saccades with different durations. A more flexible in this respect neural oscillator could solve this problem, or substituting the oscillator with a resettable integrator, as in classical saccade generation models [28]. This modification will also solve the problem that an interrupted saccade, currently, will be resumed, but won't end at the correct pose.

5.2 Discussion of the Architecture in Relation to the Neural Mechanisms of Saccades Generation

The currently most widely accepted picture of the saccade generating circuitry includes the following neuronal structures [13]: superior colliculus (SC), saccadic burst generators in the reticular formation, cerebellum, basal ganglia, and cortical structures. Here, we review briefly how each of these brain areas is reflected in our architecture.

5.2.1 Superior Colliculus

The SC is considered to be responsible for representation of the amplitudes and directions of saccades in retinal coordinates. In our architecture, the visual (retinotopic) target field accomplishes this role, since activity peaks in this DNF encode the saccades targets in retinal coordinates. But also the spatial component of the perceptual DNF probably corresponds to one of the SC layers. This field selects the saccadic target, but does not keep this representation fixed during the eye movement, but tracks the visual input to some extent. The burst, reset, and fixate nodes, roughly correspond to the three different types of saccade-related neurons, found in SC [21, 43]. The nodes perform the spatial-to-temporal transformation, which converts the location of the activated region on the visual target map into a temporal signal encoding the desired speed of the eyes, similar to other neural models, e.g. [26]. The nodes of the saccade generating circuit may be seen in close connection to the visual target DNF, and a more precise model of SC would have a number of layers with the topology of the visual target DNF and interconnected among each other like the nodes in Figure 2. We are not modelling the log-polar topology of the SC here – our visual target DNF has a cartesian structure, but the log-polar organisation of this field can also be used [38].

5.2.2 Saccadic Burst Generators

Our saccade-generating circuit is similar to the burst, buildup, and fixate neurons, proposed in several models for saccade generation [13]. This circuit provides for the temporal properties and dynamics of saccades, in particular the relation between velocity, amplitude, and duration of the saccade, time of its initiation, and reaction to perturbations. The mechanisms we used to produce two-dimensional saccades are closely related to mechanisms proposed in earlier models [8, 10], which include a shared burst generator driving the two components of the eye movement through different gain factors. This setup results in straight oblique saccades. Experimentally, it has not yet been decided on the nature and substrate of the saccade generating motor signal [13], but an alternative to our solution would be a classical model by [28], in which the motor command is integrated to achieve the target pose during saccade generation. This solution may be realised in our architecture, but would require the transformation from the retinal representation of the target to its motor representation to be learned before this integration leads to saccades of the correct amplitude.

5.2.3 Cerebellum

Fine-tuning of the saccadic amplitude, as well as corrections for changes in motor plant are found to happen in the cerebellum [26, 22]. This structure is also considered to be responsible for taking the starting position of the eye into account and compensating for the non-linearities of the motor system. In our architecture, the gain maps correspond to this function of the cerebellar structures, in particular, they attenuate saccades and provide for adaptation in the saccadic circuitry. The gain maps also reflect to models of the cerebellum as the locus of supervised learning, responsible for long-term calibration and adaptation of saccades' gains. E.g., a similar, but more abstract and relying on different error signals model is discussed in [4] . Gain adaptation in our framework is similar to adaptation process described in [11, 5]. Although in several models [19, 26, 23] the control of saccades' accuracy is controlled by the cerebellum only, cortical structures are probably also involved in this process [13]. Which, again, shows that the correspondence of an established function to the neural substrate might not be unequivocal.

5.2.4 Basal Ganglia

Error signal, generated by the 'too-far, too-close' module in our architecture, could be neurally associated with inferior olive, as in the model of [34]. The error in our model is determined directly based on the visual input after the saccades, whereas in their model the error is defined based on proprioception, using the memory of the saccade signal, the non-confirmation of the correct cascade by the fixation system, and the motor signal of the corrective saccade. In our framework, the error estimation module takes input from the visual memory of the saccadic target and the perceived visual representation of the target after the saccade and estimates whether the target after the saccade is on the same or on the different side of the midline compared to the original location of the target on the retina. This very basic operation delivers meaningful results, which drive learning in the correct direction, even when the corrective saccade also cannot be performed in the correct direction yet, since the system is completely uncalibrated. An elaborated visual processing mechanism to estimate visual error after saccade, which drives adaptation, has been discussed in relation to experimental work on saccadic adaptation [24].

Since basal ganglia are probably also involved in overall temporal coordination of saccade generation, our gaze shift generator nodes circuitry, which is responsible for temporal coordination of the gaze shifts, fixation, adaptation, and memory formation, could partially reside in this region.

5.2.5 Cortical Structures: Adaptation and Spatiotopic Visual Maps

The location of adapted region in our gain maps depends on a combination of the retinal position of the target and the position of the eye before the saccade. Consequently, adaptation effectively influences the target location in the allocentric, body centred frame of reference. This is conform with a recent experimental finding [44], which establishes that adaptation influences saccade targeting for the same position in the allocentric space. The spatial spread of the adapted region in our architecture corresponds to the experimental findings as well [15]. Our gaze-based memory representation of the visual scene resembles recent evidence, which shows that the motor representation is what is updated in the double-step paradigm and is probably used to plan multiple saccades [25]. Both these functions – learning of the saccade's amplitude and formation of memory for the visually observed scene – at least partially are solved by cortical structures [37].

6 Conclusions and Outlook

The architecture presented here demonstrates how looking behaviour may be learned autonomously and lead to formation of memory in body-centered coordinates, which may be used to direct actions at objects around us. The model demonstrates what it takes to create an illusion of the stable perception from a sequence of fast eye movements, in particular the critical role in this process of adaptation and learning, the temporal coordination between different processes and sensorimotor structures, and interplay between memory formation and exploration. Considering the integrated, dynamical system of functional modules allowed us to reveal how closely interconnected different functions of the looking system may be. There are different directions, in which this architecture may be developed. To add neural plausibility and account for neural and behavioural data is one of them, whereas extending the architecture towards a system, capable of learning to generate arm movements towards visually observed targets and thus creating a more complex self-calibrating embodied agent is another possible direction.

Acknowledgements. The authors gratefully acknowledge the financial support of DFG SPP *Autonomous Learning*, within Priority Program 1567.

References

1. Amari, S.: Dynamics of pattern formation in lateral-inhibition type neural fields. Biological Cybernetics 27, 77–87 (1977)
2. Aslin, R.N.: Perception of visual direction in human infants. In: Visual Perception and Cognition in Infancy, pp. 91–119 (1993)
3. Chao, F., Lee, M.H., Lee, J.J.: A developmental algorithm for ocularmotor coordination. Robotics and Autonomous Systems 58(3), 239–248 (2010)
4. Dean, P., Mayhew, J.E., Langdon, P.: Learning and maintaining saccadic accuracy: a model of brainstem-cerebellar interactions. Journal of Cognitive Neuroscience 6(2), 117–138 (1994)
5. Ebadzadeh, M., Darlot, C.: Cerebellar learning of bio-mechanical functions of extraocular muscles: modeling by artificial neural networks. Neuroscience 122(4), 941–966 (2003)
6. Erlhagen, W., Bicho, E.: The dynamic neural field approach to cognitive robotics. Journal of Neural Engineering 3(3), R36–R54 (2006)
7. Erlhagen, W., Schöner, G.: Dynamic field theory of movement preparation. Psychological Review 109, 545–572 (2002)
8. Fuchs, A.F., Kaneko, C.R.S., Scudder, C.A.: Brainstem control of saccadic eye movements. Annual Review of Neuroscience 8(1), 307–337 (1985)
9. Gail, A., Andersen, R.: Neural dynamics in monkey parietal reach region reflect context-specific sensorimotor transformations. The Journal of Neuroscience 26(37), 9376–9384 (2006)
10. Gancarz, G., Grossberg, S.: A neural model of the saccade generator in the reticular formation. Neural Networks (1998)
11. Gancarz, G., Grossberg, S.: A neural model of saccadic eye movement control explains task-specific adaptation. Vision Research 39(18), 3123–3143 (1999)
12. Gibson, J.J.: The perception of the visual world (1950)
13. Girard, B., Berthoz, A.: From brainstem to cortex: computational models of saccade generation circuitry. Progress in Neurobiology 77(4), 215–251 (2005)
14. Grossberg, S.: Nonlinear neural networks: Principles, mechanisms, and architectures. Neural Networks 1, 17–61 (1988)
15. Hopp, J.J., Fuchs, A.F.: The characteristics and neuronal substrate of saccadic eye movement plasticity. Progress in Neurobiology 72(1), 27–53 (2004)
16. Iossifdis, I., Bruckhoff, C., Theis, C., Grote, C., Faubel, C., Schöner, G.: CORA:An Anthropomorphic Robot Assistant for Human Environment. In: Proceedings of the 2002 IEEE Int. Workshop on Robot and Human Interactive Communication, Berlin, Germany, September 25-27, pp. 392–398 (2002)

17. Itti, L., Koch, C.: Computational modeling of visual attention. Nature Reviews Neuroscience 2, 1–11 (2001)
18. Kopecz, K., Schöner, G.: Saccadic motor planning by integrating visual information and pre-information on neural, dynamic fields. Biological Cybernetics 73, 49–60 (1995)
19. Lefèvre, P., Quaia, C., Optican, L.M.: Distributed model of control of saccades by superior colliculus and cerebellum. Neural Networks 11 (1998)
20. Lomp, O., Zibner, S.K.U., Richter, M., Rañó, I., Schöner, G.: A Software Framework for Cognition, Embodiment, Dynamics, and Autonomy in Robotics: cedar. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 475–482. Springer, Heidelberg (2013)
21. Munoz, D.P., Wurtz, R.H.: Saccade-related activity in monkey superior colliculus. I. Characteristics of burst and buildup cells. Journal of Neurophysiology 73(6), 2313–2333 (1995)
22. Optican, L.M.: Sensorimotor transformation for visually guided saccades. Annals of the New York Academy of Sciences 1039, 132–148 (2005)
23. Optican, L.M., Quaia, C.: Distributed Model of Collicular and Cerebellar Function during Saccades. Annals of the New York Academy of Science 956, 164–177 (2002)
24. Pelisson, D., Alahyane, N.: Sensorimotor adaptation of saccadic eye movements. Neuroscience & Biobehavioral Reviews 34, 1103–1120 (2010)
25. Quaia, C., Joiner, W.M., FitzGibbon, E.J., Optican, L.M., Smith, M.A.: Eye movement sequence generation in humans: Motor or goal updating? Journal of Vision 10(14) (2010)
26. Quaia, C., Lefèvre, P., Optican, L.M.: Model of the control of saccades by superior colliculus and cerebellum. Journal of Neurophysiology 82(2), 999–1018 (1999)
27. Richter, M., Sandamirskaya, Y., Schöner, G.: A robotic architecture for action selection and behavioral organization inspired by human cognition. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2012)
28. Robinson, D.A.: Oculomotor control signals. In: Lennerstrand, G., Bach-y Rita, P. (eds.) Basic Mechanisms of Ocular Motility and Their Clinical Implications, pp. 337–374. Pergamon Press, Oxford (1975)
29. Sandamirskaya, Y.: Dynamic Neural Fields as a Step Towards Cognitive Neuromorphic Architectures. Frontiers in Neuroscience 7, 276 (2013)
30. Sandamirskaya, Y., Richter, M., Schöner, G.: A neural-dynamic architecture for behavioral organization of an embodied agent. In: IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL EPIROB 2011) (2011)
31. Sandamirskaya, Y., Schöner, G.: An Embodied Account of Serial Order: How Instabilities Drive Sequence Generation. Neural Netw. 23(10), 1164–1179 (2010)
32. Sandamirskaya, Y., Zibner, S.K.U., Schneegans, S., Schöner, G.: Using Dynamic Field Theory to extend the embodiment stance toward higher cognition. New Ideas in Psychology 31(3), 322–339 (2013)
33. Schöner, G.: Dynamical Systems Approaches to Cognition. In: Sun, R. (ed.) Cambridge Handbook of Computational Cognitive Modeling, pp. 101–126. Cambridge University Press, Cambridge (2008)
34. Schweighofer, N., Arbib, M.A., Dominey, P.F.: A model of the cerebellum in adaptive control of saccadic gain. Biological Cybernetics 75(1), 19–28 (1996)
35. Scudder, C.A.: A new local feedback model of the saccadic burst generator. Journal of Neurophysiology 59(5), 1455–1475 (1988)
36. Spencer, J.P., Schöner, G.: Embodied Approach to Cognitive Systems: A Dynamic Neural Field Theory of Spatial Working Memory. In: . . . Annual Conference of the Cognitive . . . , pp. 2180–2185 (2006)

37. Steve, N.G., Charles, T., Benoît, G.: Saccade learning with concurrent cortical and subcortical basal ganglia loops. arXiv preprint arXiv:1312.5212, 1–34 (2013)
38. Trappenberg, T.P., Dorris, M.C., Munoz, D.P., Klein, R.M.: A model of saccade initiation based on the competitive integration of exogenous and endogenous signals in the superior colliculus. Journal of Cognitive Neuroscience 13(2), 256–271 (2001)
39. Tweed, D., Vilis, T.: A two dimensional model for saccade generation. Biol. Cybern. 52, 219–227 (1985)
40. Wilimzig, C., Schneider, S., Schöner, G.: The time course of saccadic decision making: dynamic field theory. Neural Networks: the Official Journal of the International Neural Network Society 19(8), 1059–1074 (2006)
41. Wilimzig, C., Schöner, G.: The Emergence of Stimulus-Response Associations from Neural Activation Fields: Dynamic Field Theory. In: Proceedings of the Twenty-Seventh Annual Cognitive Science Society, pp. 2359–2364. Cognitive Science Society, Stresa (2005)
42. Wilson, H.R., Cowan, J.D.: A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. Kybernetik 13, 55–80 (1973)
43. Wurtz, R.H., Optican, L.M.: Superior colliculus cell types and models of saccade generation. Current Opinion in Neurobiology 4, 857–861 (1994)
44. Zimmermann, E., Burr, D., Morrone, M.C.: Spatiotopic Visual Maps Revealed by Saccadic Adaptation in Humans. Current Biology (2011)

How to Pretrain Deep Boltzmann Machines in Two Stages

Kyunghyun Cho, Tapani Raiko, Alexander Ilin, and Juha Karhunen

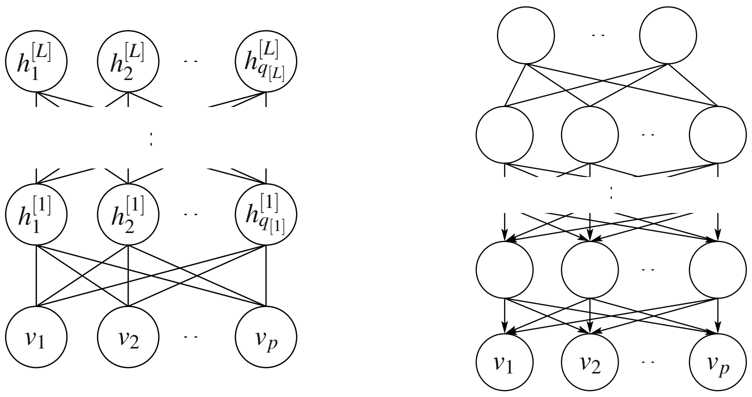
Abstract. A deep Boltzmann machine (DBM) is a recently introduced Markov random field model that has multiple layers of hidden units. It has been shown empirically that it is difficult to train a DBM with approximate maximum-likelihood learning using the stochastic gradient unlike its simpler special case, restricted Boltzmann machine (RBM). In this paper, we propose a novel pretraining algorithm that consists of two stages; obtaining approximate posterior distributions over hidden units from a simpler model and maximizing the variational lower-bound given the fixed hidden posterior distributions. We show empirically that the proposed method overcomes the difficulty in training DBMs from randomly initialized parameters and results in a better, or comparable, generative model when compared to the conventional pretraining algorithm.

1 Introduction

Deep Boltzmann machine (DBM), proposed in [36], is a recently introduced variant of Boltzmann machines which extends the widely used restricted Boltzmann machine (RBM) to have multiple layers of hidden neurons. It differs from the popular deep belief network (DBN) which is built by stacking multiple layers of RBMs [22] in that every edge in the DBM model is undirected. This construction of DBMs facilitates propagating uncertainties across multiple layers of hidden variables.

Although it is straightforward to derive a learning algorithm for DBMs using a variational approximation and stochastic maximum likelihood method, recent research (for example [36] and [15]) has shown that learning the parameters of DBMs is not trivial. Especially the generative performance of the trained model, commonly measured by the variational lower-bound of log-probabilities of test samples, tends to degrade as more hidden layers are added.

Kyunghyun Cho · Tapani Raiko · Alexander Ilin · Juha Karhunen
Department of Information and Computer Science,
Aalto University School of Science, Finland
e-mail: firstname.lastname@aalto.fi



(a) Deep Boltzmann Machine (b) Deep Belief Network

Fig. 1 Illustrations of (a) a deep Boltzmann machines (DBM) and (b) a deep belief network (DBN)

In [36] a greedy layer-wise pretraining algorithm was proposed to initialize the parameters of DBMs. It was shown that the proposed algorithm largely overcomes the difficulty of learning a good generative model.

Along this line of research, we propose another strategy of pretraining DBMs in this paper. The proposed scheme is based on an observation that training DBMs consists of two separate stages; approximating the posterior distribution over hidden units and updating parameters to maximize the lower-bound of the log-likelihood given the approximate posterior distribution.

Based on this observation, the proposed algorithm in this paper pretrains a DBM in two stages. During the first stage we train a simpler, directed deep model such as DBNs or stacked denoising autoencoders (sDAE) to obtain an approximate posterior distribution over hidden units. With this approximate posterior distribution fixed, we train an RBM that learns a distribution over a combination of data samples and their corresponding posterior distributions over the hidden units. It is then trivial to finetune the model as one only needs to simply free hidden variables from the fixed approximate posterior distribution obtained in the first stage.

We show that the proposed algorithm helps learning a good generative model which is empirically comparable to, or better than the pretraining method proposed in [36]. Furthermore, we discuss the degrees of freedom in extending the proposed approach. The preliminary results of this work were presented in our conference paper [13] and workshop paper [12].

2 Deep Boltzmann Machines

We start by describing deep Boltzmann machines (DBM) [36]. A DBM with L layers of hidden neurons is defined by the following negative energy function:

$$
\begin{aligned}
-E(\mathbf{v},\mathbf{h} \mid \theta) = &\sum_{i}^{N_v} v_i b_i + \sum_{i}^{N_v}\sum_{j}^{N_1} v_i h_j^{[1]} w_{i,j} + \sum_{j}^{N_1} h_j^{[1]} c_j^{[1]} + \\
&\sum_{l=2}^{L}\left(\sum_{j}^{N_l} h_j^{[l]} c_j^{[l]} + \sum_{j}^{N_l}\sum_{k}^{N_{l+1}} h_j^{[l]} h_k^{[l+1]} u_{j,k}^{[l]}\right),
\end{aligned} \tag{1}
$$

where $\mathbf{v} = [v_i]_{i=1\dots N_v}$ and $\mathbf{h}^{[l]} = \left[h_j^{[l]}\right]_{j=1\dots N_l}$ are N_v binary visible units and N_l binary hidden units in the l-th hidden layer, where $1 \le l \le L$. See Figure 1 (a) for the graphical description of the model.

$\mathbf{W} = [w_{i,j}]_{i=1\dots N_v, j=1\dots N_1}$ is the set of weights between the visible neurons and the first layer hidden neurons. $\mathbf{U}^{[l]} = \left[u_{j,k}^{[l]}\right]_{i=1\dots N_l, j=1\dots N_{l+1}}$ is the set of weights between the l-th and $(l+1)$-th hidden neurons, where $1 \le l < L$, in this case. b_i and $c_j^{[l]}$ are a bias to the i-th visible neuron and the j-th hidden neuron in the l-th hidden layer, respectively. We use θ to denote a set of all these parameters.

With the energy function, a DBM can assign a probability to each state vector

$$\mathbf{x} = [\mathbf{v}; \mathbf{h}^{[1]}; \cdots; \mathbf{h}^{[L]}]$$

using a Boltzmann distribution

$$p(\mathbf{x} \mid \theta) = \frac{1}{Z(\theta)} \exp\{-E(\mathbf{x} \mid \theta)\}, \tag{2}$$

where

$$Z(\theta) = \sum_{\forall \mathbf{x}} \exp\{-E(\mathbf{x} \mid \theta)\}.$$

Under this formulation, the conditional distribution of each hidden layer l is

$$p(h_j^{[l]} = 1 \mid \mathbf{h}^{[l-1]}, \mathbf{h}^{[l+1]}, \theta) = f\left(\sum_{k=1}^{q_{l-1}} h_k^{[l-1]} u_{kj}^{[l-1]} + \sum_{i=1}^{q_{l+1}} h_i^{[l+1]} u_{ji}^{[l]} + c_j^{[l]}\right), \tag{3}$$

and the conditional distribution of the visible layer is

$$p(v_i = 1 \mid \mathbf{h}^{[1]}, \theta) = f\left(\sum_{j=1}^{q_1} h_j^{[1]} w_{ij} + b_i\right), \tag{4}$$

where

$$f(x) = \frac{1}{1 + \exp\{-x\}} \tag{5}$$

is a logistic sigmoid function.

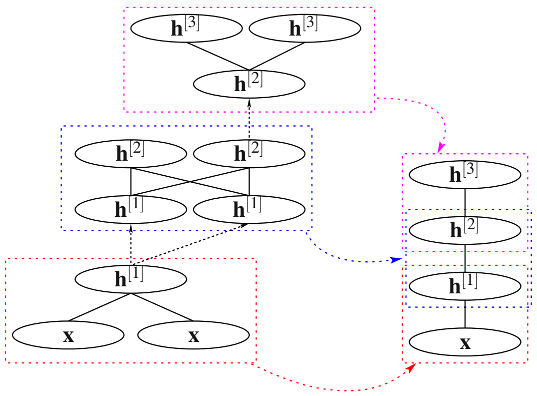


Fig. 2 Illustration of the layer-wise pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* either the pretrained models or the activations of the hidden units of the pretrained models.

3 Training Deep Boltzmann Machines

Using the probability defined by a DBM in Eq. (2) the parameters of the DBM can be learned by maximizing the log-likelihood

$$\mathscr{L}(\theta) = \sum_{n=1}^{N} \log \sum_{\mathbf{h}} p(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta)$$

given N training samples $\{\mathbf{v}^{(n)}\}_{n=1,\dots,N}$, where

$$\mathbf{h} = \left[\mathbf{h}^{[1]}; \cdots ; \mathbf{h}^{[L]}\right].$$

The gradient is stochastically estimated by taking the partial derivative of the log-likelihood function $\mathscr{L}$ with respect to each parameter θ using only a subset of training samples. This estimate is then used to update the parameters, effectively forming a stochastic gradient ascent method. A standard way of computing gradient results in the following update rule for each parameter θ:

$$\nabla_\theta \mathscr{L} = \left\langle -\frac{\partial E(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta)}{\partial \theta} \right\rangle_{\mathrm{d}} - \left\langle -\frac{\partial E(\mathbf{v}, \mathbf{h} \mid \theta)}{\partial \theta} \right\rangle_{\mathrm{m}}, \tag{6}$$

where $\langle \cdot \rangle_{\mathrm{d}}$ and $\langle \cdot \rangle_{\mathrm{m}}$ denote the expectation over the data distribution $P(\mathbf{h} \mid \mathbf{v}, \theta)D(\mathbf{v})$ and the model distribution $P(\mathbf{v}, \mathbf{h} \mid \theta)$, respectively [7]. D denotes an empirical data distribution.

Although the update rule in Eq. (6) is well defined, it is intractable to compute both terms in the update rule exactly. Hence, an approach based on variational approximation together with Markov chain Monte Carlo (MCMC) sampling was proposed in [36].

First, variational approximation is used to compute the expectation over the data distribution. It starts by approximating the posterior distribution over the hidden variables $p(\mathbf{h} \mid \mathbf{v}, \theta)$, which is intractable unless $L = 1$, by a factorial distribution

$$Q(\mathbf{h}) = \prod_{l=1}^{L} \prod_{j=1}^{N_l} \left(\mu_j^{[l]}\right)^{h_j^{[l]}} \left(1 - \mu_j^{[l]}\right)^{1-h_j^{[l]}}.$$

Each variational parameter $\mu_j^{(l)}$ can then be estimated by the following fixed-point equation:

$$\mu_j^{[l]} \leftarrow f\left(\sum_{i=1}^{N_{l-1}} \mu_i^{[l-1]} w_{ij}^{[l-1]} + \sum_{k=1}^{N_{l+1}} \mu_k^{[l+1]} w_{kj}^{[l]} + c_j^{[l]}\right), \tag{7}$$

where f is a logistic sigmoid function in Eq. (5). Note that $\mu_i^{(0)}$ is fixed to v_i and the update rule for the top layer does not contain the second summation term, that is $N_{L+1} = 0$.

This variational approximation provides the values of variational parameters that maximize the following variational lower-bound (right-hand side) of the true log-probability of $\mathbf{v}$ with respect to the current parameters θ:

$$\log p(\mathbf{v} \mid \theta) \geq \mathbb{E}_{Q(\mathbf{h})}\left[-E(\mathbf{v}, \mathbf{h})\right] + \mathscr{H}(Q) - \log Z(\theta), \tag{8}$$

where

$$\mathscr{H}(Q) = -\sum_{l=1}^{L} \sum_{j=1}^{N_l} \left(\mu_j^{[l]} \log \mu_j^{[l]} + (1 - \mu_j^{[l]}) \log(1 - \mu_j^{[l]})\right)$$

is an entropy functional.

Due to the layered structure of a DBM, it is possible to analytically sum out the odd-numbered hidden layers (see, e.g., [20]). If we denote the odd-numbered and even-numbered hidden layers by $\mathbf{h}_+$ and $\mathbf{h}_-$ respectively, we may, then, rewrite Eq. (8) into

$$\log p(\mathbf{v} \mid \theta) \geq \mathbb{E}_{Q(\mathbf{h}_-)}\left[-\sum_{\mathbf{h}_+} E(\mathbf{v}, \mathbf{h}_+, \mathbf{h}_-)\right] + \mathscr{H}(Q) - \log Z(\theta). \tag{9}$$

Since the first term of the gradient in Eq. (6) is approximated in this way, each gradient update step does not increase the true log-likelihood directly but its variational lower-bound.

Second, the expectation over the model distribution is computed by persistent sampling (see, for example, [43]). By persistent sampling, we mean that we do not wait for a sampling chain to converge before computing each update direction, but run the chain for only a few steps and continue using the same chain over consecutive updates. The simplest approach is to use Gibbs sampling, while there have been some work in applying more advanced sampling methods [35, 34, 16, 8]. In this paper, we use coupled adaptive simulated tempering (CAST) which was recently proposed in [35].

This approach closely resembles variational expectation-maximization (EM) algorithm (see, for example, [5]). Learning proceeds by alternating between finding the variational parameters μ and updating the DBM parameters θ to maximize the given variational lower-bound using the stochastic gradient method.

It has, however, been known and will be shown in the experiments in this paper that training a DBM using this approach starting from randomly initialized parameters is not trivial [36, 15, 11]. The difficulty of training without any pretraining was illustrated in [36] and [15] by a lower log-likelihood achieved by a DBM trained without any pretraining. Furthermore, the lack of proper initialization of the parameters was found to result in the upper-level hidden neurons not being able to capture any interesting features of an input data in [11].

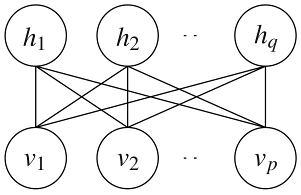
3.1 Layer-Wise Pretraining

In [36] a pretraining algorithm to initialize the parameters of DBMs was proposed. The proposed pretraining algorithm greedily trains each layer of a DBM by considering each layer as an RBM, similarly to a pretraining approach used for training deep belief networks (DBN) [22]. However, the pretraining algorithm for DBMs differs from DBNs[1] due to the undirectedness of all the edges in a DBM, which requires that the pretraining algorithm for DBMs must take into account that each hidden unit in a DBM receives a signal from both upper and lower layers (see Eq. (3)).

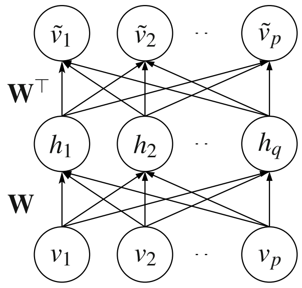
The algorithm proposed in [36] modifies the structure of RBMs to cope with this difference. For the bottom two layers, an RBM is modified to have two copies of visible units with tied weights such that the additional set of visible units supplies signal that compensates for the lack of signal from the second hidden layer. Similarly, an RBM that consists of the top two layers has the two copies of hidden units. For any pair of intermediate hidden layers, an RBM is constructed to have two copies of both visible and hidden units. See Figure 2 for an illustration.

Recently, in [38] the same authors were able to show that the variational lower bound is guaranteed to increase by adding the top hidden layer using the proposed pretraining scheme. Their proof, however, only applies to the top layer, which means that the guarantee only works for pretraining a DBM having two hidden layers.

[1] Compare Figures 1 (a) and (b) for the difference between a DBM and a DBN. The edges of the DBN except for those between the top two hidden layers are *directed*, pointing downward.



(a) Restricted Boltzmann Machine



(b) Denoising Autoencoder

Fig. 3 Illustrations of (a) a restricted Boltzmann machine (RBM) and (b) a denoising autoencoder (DAE)

4 Restricted Boltzmann Machines and Denoising Autoencoders

Here we briefly discuss restricted Boltzmann machines (RBM) and single-layer denoising autoencoders (DAE) which will constitute an important part of the two-stage pretraining algorithm that will be described in the next section.

An RBM is a special case of DBMs, where the number of hidden layers is restricted to one, $L = 1$ [42] (see Figure 3 (a)). Due to this restriction it is possible to compute the posterior distribution over the hidden units conditioned on the visible neurons exactly and tractably. The conditional probability of each hidden unit $h_j = h_j^{[1]}$ is

$$p(h_j = 1 \mid \mathbf{v}, \theta) = f\left(\sum_i w_{ij} v_i + c_j\right),$$

where f is a logistic sigmoid function from Eq. (5).

This allows exact and efficient computation of the positive part of the gradient in (6). However, the negative part, which is computed over the model distribution, still relies on persistent sampling, or more approximate methods such as contrastive divergence (CD) [20].

There have been extensive research on improving training RBMs using various techniques. In [9, 10] the authors proposed enhanced gradient which exploits the fact that the gradient update of RBMs is not invariant to the bit-flipping transformation and showed that it outperforms the traditional gradient. In [16, 34, 8], advanced sampling methods for computing the negative part of the gradient based on tempering were proposed and shown to improve and stabilize learning.

A single-layer DAE is a special form of multi-layer perception network with a single hidden layer and a tied set of weights [45] (see Figure 3 (b)). A DAE is a network that reconstructs a corrupted input vector as well as possible by minimizing the following cost function

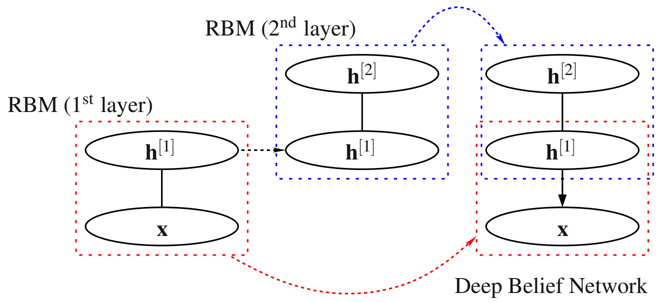


Fig. 4 Illustration of the stack of RBMs. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models.

$$\sum_{n=1}^{N} \left\| g_v \left(\mathbf{W} g_h \left(\mathbf{W}^\top \phi(\mathbf{v}^{(n)}) \right) \right) - \mathbf{v}^{(n)} \right\|^2, \tag{10}$$

where $g_h(\cdot)$ and $g_v(\cdot)$ are component-wise nonlinear functions. ϕ is a stochastic corruption process that corrupts the input $\mathbf{v}^{(n)}$ stochastically.

There is an important difference in training DAEs compared with training RBMs. In DAEs, the objective of learning is not to learn a distribution but to minimize the reconstruction error. This does not require computing a computationally intractable normalizing constant, which often leads to easier learning.[2]

These two models are important, because they can be stacked to form more powerful hierarchical models [22, 21, 2].

A deep belief network (DBN) is constructed by stacking multiple layers of RBMs [22], and a stacked DAE (sDAE) can be built by stacking DAEs on top of each other [46]. With probabilistic interpretation, one may consider these stacked models as having multiple layers of latent random variables where their posterior distributions can be computed by recursively obtaining (approximate) posterior means of the hidden units layer-wise. See Figure 4 for the illustration.

5 A Two-Stage Pretraining Algorithm

In this paper, we propose an alternative way of initializing parameters of a DBM compared with the one described in Section 3.1. Unlike the conventional pretraining strategy we employ an approach that obtains approximate posterior distributions over hidden units and initializes parameters separately.

Before proceeding to the description of the proposed algorithm, we first divide the hidden layers of a DBM into two sets as we have done in Section 3. Let us denote a vector of hidden units in the odd-numbered layers as $\mathbf{h}_+$ and the respective vector in the even-numbered layers as $\mathbf{h}_-$. Note that due to the structure of DBMs,

[2] Despite this difference in the learning objective, recent research such as [3] suggests that the DAE approximates a data generating distribution as well.

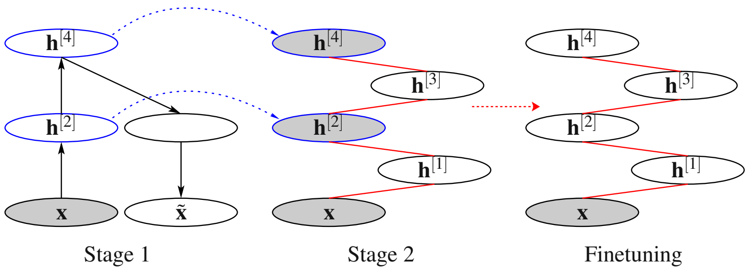


Fig. 5 Illustration of the two-stage pretraining of a deep Boltzmann machine. The dashed directed lines indicate *copying* of either pretrained models or the activations of the hidden units of the pretrained models. In this figure, a deep autoencoder is used to learn an arbitrary approximate posterior in the first stage. The red-colored edges indicate that the weights parameters learned in the second stage are used as initial values when finetuning the DBM. Note that the parameters learned in the first stage are discarded immediately after the first stage.

it is possible to explicitly sum out $\mathbf{h}_+$, which halves the space of hidden variables. Similarly we define μ_+ and μ_- as variational parameters of the hidden units in the odd-numbered layers and the even-number layers, respectively.

5.1 Stage 1

During the first stage we focus on finding a good set of variational parameters μ_- of $Q(\mathbf{h}_-)$ that has a potential to give a reasonably high variational lower-bound in Eq. (8). In other words, we propose to first find a good posterior distribution over hidden units given a visible vector regardless of parameter values of a DBM.

Although it might sound unreasonable to find a good set of variational parameters without any fixed parameter values, we can do this by *borrowing* posterior distributions over latent variables from another model.[3]

We propose here to utilize either a DBN or a sDAE, described in Section 4, to find good approximate posterior distributions over hidden units in the even-numbered hidden layers. However, it is possible to use any model that finds a good binary hierarchical posterior distribution.

DBNs and sDAE's described in Section 4 are natural choices to find a good approximate posterior distribution over units in the even-numbered hidden layers. One justification for using either of them is that they can be trained efficiently and well (see, e.g., [1] and references therein). It is rather a trivial task where one iteratively trains each even-numbered layer as either an RBM or a DAE on top of each other.

[3] A similar approach of borrowing the posterior means of hidden variables has been proposed in [23]. The authors of that paper initialized the variational Bayesian nonlinear blind source separation model with the posterior distribution borrowed from kernel principal component analysis.

5.2 Stage 2

Once a set of initial variational parameters μ_- is found from a DBN or an sDAE, we train a model that has predictive power of the variational parameters given a visible observation. It can be simply done by letting an RBM learn a joint distribution of $\mathbf{v}$ and μ_-. In other words, we train an RBM on a set of data samples, each of which is a concatenation of $\mathbf{v}$ and μ_-.

The structure of the RBM is directly derived from the DBM such that the visible layer of the RBM corresponds to the visible layer and the even-numbered hidden layers of the DBM and the hidden layer to the odd-numbered hidden layers of the DBM. The connections between them can also follow those of the DBM. This corresponds to finding a set of DBM parameters that *fit* the variational parameters obtained in the first stage.

One way to understand what happens during the second stage is to consider what an RBM has been trained for. If we assume that we use actual samples from $Q(\mathbf{h}_-)$ instead of the variational parameters μ_-, training the RBM maximizes

$$\mathscr{L}_2(\theta) = \log \sum_{\mathbf{h}_+} \mathbb{E}_{Q(\mathbf{h}_-)} \exp\{-E(\mathbf{v}, \mathbf{h}_-, \mathbf{h}_+)\} - \log Z(\theta). \tag{11}$$

However, since we use the variational parameters which are the mean of $Q(\mathbf{h}_-)$, the actual quantity being maximized is the lower-bound of Eq. (11) according to the Jensen's inequality and the linearity of the expectation:

$$\mathscr{L}_2(\theta) \geq \sum_{\mathbf{h}_+} \{-E(\mathbf{v}, \mu_-, \mathbf{h}_+)\} - \log Z(\theta). \tag{12}$$

It is easy to see that this corresponds to maximizing the variational lower-bound of the DBM, if we group the three terms of Eq. (9) such that

$$\log p(\mathbf{v} \mid \theta) \geq \underbrace{\mathbb{E}_{Q(\mathbf{h}_-)} \left[-\sum_{\mathbf{h}_+} E(\mathbf{v}, \mathbf{h}_+, \mathbf{h}_-) \right] - \log Z(\theta)}_{\text{(a)}} + \mathscr{H}(Q).$$

The two terms grouped as (a) in the above equation is equivalent to Eq. (12) which is maximized during the second stage.[4]

Once the RBM has been trained, we can use the learned parameters as initializations for training the DBM, which corresponds to freeing $\mathbf{h}_-$ from its variational posterior distribution obtained in the first stage. Finetuning of the initialized parameters can be performed according to the standard procedure given in Section 3.

The overall steps of the proposed learning algorithm are presented in Algorithm 1, and a simple illustration is given in Figure 5.

[4] The entropy functional in Eq. (9) can be ignored, as it is constant with respect to the parameters θ.

Algorithm 1. Two-Stage Pretraining Algorithm

```
Input Training data $\mathbf{X}_{N \times D}$, the number of layers $L$ and the number of units in each layer $N_1, \ldots, N_L$
$\mathbf{Q} = \mathbf{X}$
$\mathbf{X}_- = []$
for $l = 1 \to L$ do
    if odd $l$ then
        continue
    end if
    Train a DAE/RBM with $N_l$ hidden units with $\mathbf{Q}$
    Set $\mathbf{Q}$ to $Q(\mathbf{h})$ from the DAE/RBM
    Append $\mathbf{Q}$ to $\mathbf{X}_-$
end for
Train an RBM with $\sum_{\text{even } l} N_l$ hidden units with $\mathbf{X}_-$
Return parameters $\theta$ of the trained RBM
```

5.3 Discussion

It is quite easy to see that the proposed algorithm has a high degree of freedom to plug in alternative algorithms and models in both the stages.

The most noticeable flexibility can be found in Stage 1. Any other machine learning model that gives reasonable posterior distributions over multiple layers of binary hidden units can be used instead of DBNs or sDAE's. For instance, a stack of recently proposed variants of RBMs such as spike-and-slab RBMs [14, 27] can be used.[5] Also, instead of stacking each layer at a time, one could opt to train deep autoencoders at once using recently proposed learning algorithms for feedforward neural networks (see, for instance, [31, 32, 41, 29]).

In Stage 2, one may use a DAE instead of an RBM. It will make learning faster and therefore leave more time for finetuning the model afterward. Also, the use of different algorithms for training an RBM can be considered. For quicker pretraining, one may use contrastive divergence [20] with only a small number of Gibbs sampling steps per update, or for better initial models, tempering-based advanced MCMC sampling methods such as parallel tempering [16, 8] or tempered transition [34] could be used.

Another obvious possibility is to utilize the conventional pretraining algorithm proposed in [36] during the first stage. This approach gives approximate posterior distributions over all hidden units $[\mathbf{h}_-; \mathbf{h}_+]$ as well as initial values of the parameters that may be used during the second stage. In this way, one may use either an RBM or a fully visible BM (FVBM) during the second stage starting from the initialized parameters. When an RBM is used in the second stage, one could simply discard μ_+.

[5] One potential restriction on the choice of a single-hidden-layer neural network is that it must be computationally inexpensive and easy to compute the posterior distribution over the hidden variables.

Table 1 Algorithms used in the experiment. (S) – the pretraining algorithm from [36].

	Stage 1	Stage 2	Finetuning
DBM	$\times$	$\times$	DBM
$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	sDAE	RBM	DBM
$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	DBN	RBM	DBM
$\text{DBM}^{\text{S\&H}}$	(S)	$\times$	DBM
$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	(S)	RBM	DBM
$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$	(S)	FVBM	DBM

One important point of the proposed algorithm is that it provides another research perspective in training DBMs. The existing pretraining scheme developed in [36, 37] was based on the observation that under certain assumptions the variational lower-bound could be increased by learning weight parameters layer wise. However, the success of the proposed scheme suggests that it may not be the set of parameters that need to be pretrained, but the set of variational parameters that determine how tight the variational lower-bound is and their corresponding parameters. This way of approaching the problem of training DBMs enables us to potentially find another explanation on why training large DBMs without pretraining is not trivial.

6 Experiments

In the experiments, we train DBMs on two datasets which are a handwritten digit dataset[6] (MNIST) [26] and Caltech-101 Silhouettes dataset[7] [28]. We used the MNIST and Caltech-101 Silhouettes datasets because experimental results of using DBMs for both datasets are readily available for direct comparison [39, 35, 30].

We train DBMs with varying numbers of units in the hidden layers; 500-1000, 500-500-1000, 500-500-500-1000. The first two architectures were used in [39, 35], which enables us to directly compare our proposed algorithm with the conventional pretraining algorithm.

For learning algorithms, we extensively tried various combinations. They are presented in Table 1. In summary, a $DBM^{stage\ 1}_{stage\ 2}$ denotes a deep Boltzmann machine in which its superscript and subscript denote the algorithms used during the first and second stages, respectively.

We used contrastive divergence (CD) to train RBMs in the first stage, and the persistent CD [44] with coupled adaptive simulated annealing (CAST) was used in the second stage. DAEs were trained using stochastic gradient descent (SGD) algorithm with backpropagation [33].

When a DBM was finetuned, we estimated the variational parameters by the variational approximation with at most 30 mean-field updates (see Eq. (7)). The model statistics, the negative part of the gradient, was computed by CAST. When the conventional pretraining algorithm is used, we do not explicitly make

[6] `http://yann.lecun.com/exdb/mnist/`
[7] `http://people.cs.umass.edu/~marlin/data.shtml`

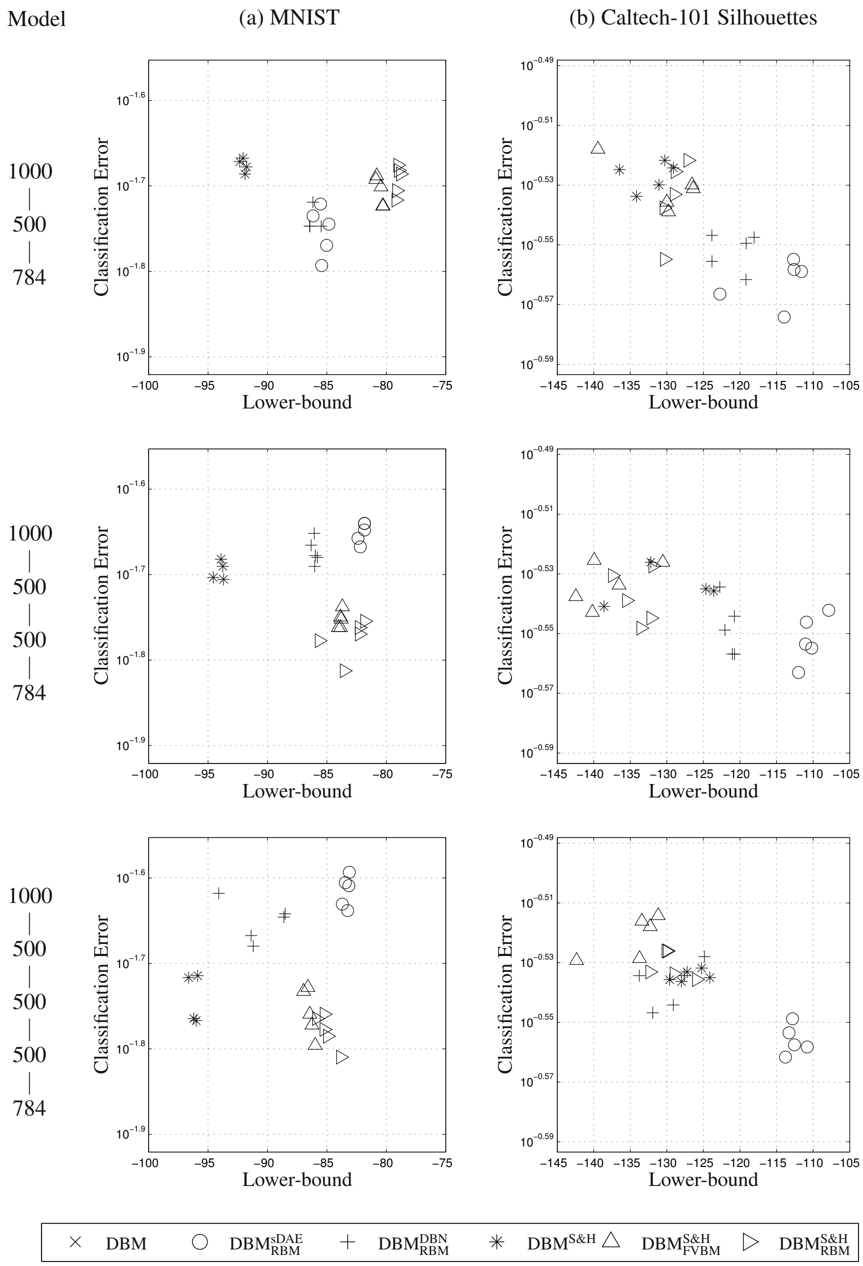


Fig. 6 Performance of the trained DBMs. Best performing models are in the bottom right corners.

duplicate copies of visible or hidden units, but only double the corresponding weight parameters [37].

We trained each model for 200 epochs in the case of MNIST and 2000 epochs in the case of Caltech-101 Silhouettes with a learning rate scheduled by $\frac{\eta_0}{1+\frac{n}{5000}}$ where n is the number of updates. η_0 was set to 0.01 and 0.0005 for pretraining and finetuning, respectively. When we did not pretrain a DBM, we trained each DBM for twice more epochs and set η_0 to 0.001. In all cases, we used a minibatch of size 128.

When training RBMs with CAST, we used equally spaced 21 tempered fast chains from 0.9 to 1 with a single sampling step each update. When DAEs were trained, at every update we dropped off a randomly chosen set of hidden units with probability 0.1 [19]. During the finetuning stage, we used CAST with the equally spaced 21 tempered chains from 0.9 to 1.

We evaluated the resulting models with the variational lower-bound of log-probabilities and the classification errors of test samples. The variational lower-bounds reflect the generative performance of the model. We trained a linear SVM [17] for each hidden layer l using μ_l as its features to compute the classification errors. This is expected to show how much discriminative information about input samples is captured by each hidden layer of the model.

The intractable normalization constant ($\log Z(\theta)$ in Eq. (8)) required when computing the variational lower-bound was approximated using annealed importance sampling (AIS) [40]. For each model, we used 20001 equally spaced tempered chains from 0 to 1 with 128 independent runs.

All models were trained five times starting from different random initializations. We report the medians over these random trials.

6.1 Result and Analysis

Figure 6 presents the result using both the lower-bound of log-probabilities and the classification error of the test samples. As has already been expected, none of the models trained *without* pretraining have been able to perform well enough to be presented inside the boundaries of the boxes in Figure 6.

It is clear from the figures that the proposed two-stage pretraining algorithm outperforms, in all cases, the conventional pretraining algorithm ($\text{DBM}^{\text{S\&H}}$). On MNIST, the DBMs pretrained with the proposed algorithm using the conventional pretraining algorithm in the first stage achieved the best performance. In the case of Caltech-101 Silhouettes, $\text{DBM}^{\text{sDAE}}_{\text{RBM}}$ was able to achieve superior performance in both generative and discriminative modeling. It is notable that without any pretraining (DBM) we were not able to achieve any reasonable performance.

Figure 7 presents layer-wise classification errors. It is clear from the significantly lower accuracies in the higher hidden layers of the DBMs trained without pretraining that pretraining is essential to allow upper layers to capture discriminative

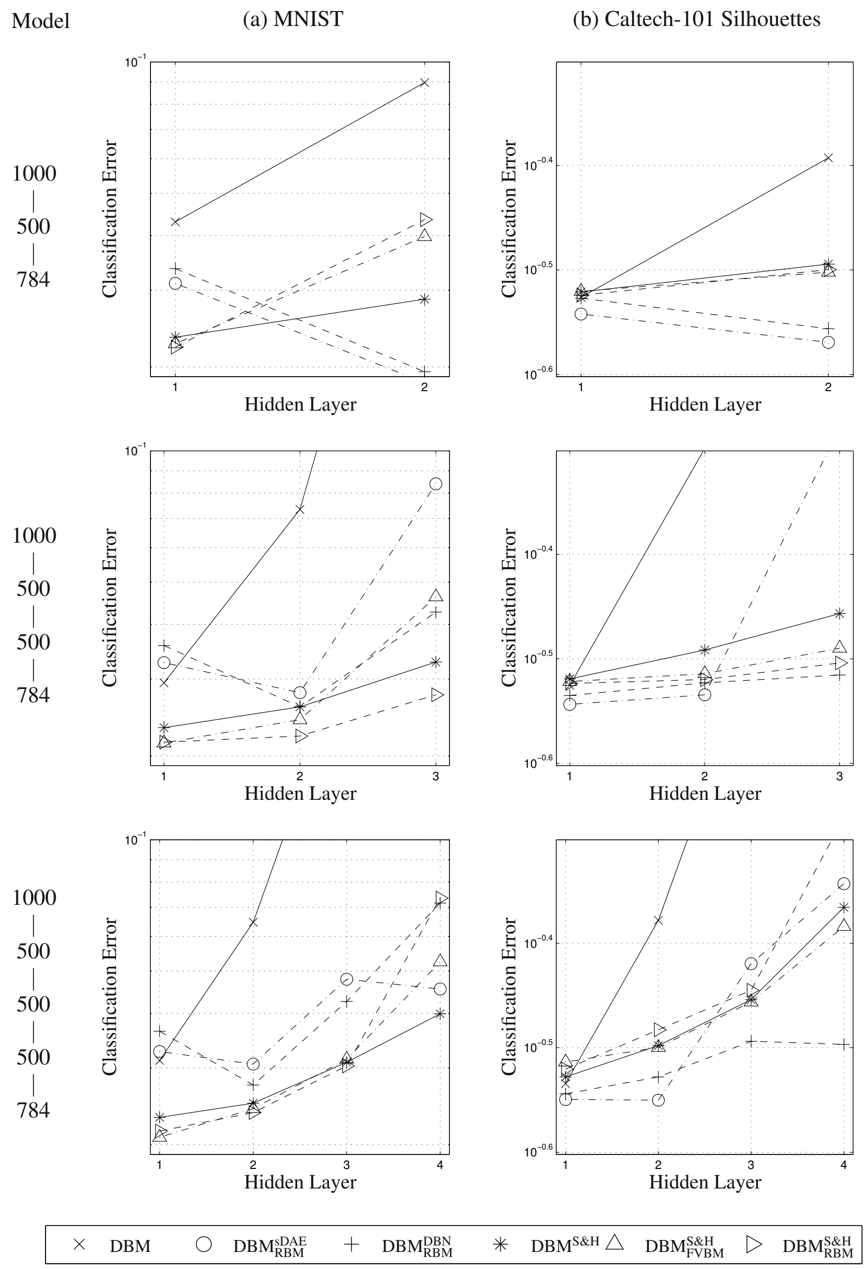


Fig. 7 Layer-wise Discriminative Performance. Lower is better.

structures of data. $\mathrm{DBM}^{\mathrm{DBN}}_{\mathrm{RBM}}$ and $\mathrm{DBM}^{\mathrm{S\&H}}_{\mathrm{RBM}}$ were most effective in ensuring the upper hidden layers to have better discriminative property.[8]

7 Conclusions

The experimental success of the proposed two-stage pretraining algorithm in training DBMs suggests that the difficulty of DBM learning might be due to the fact that the estimated variational lower-bound at the initial stage of learning is too crude, or too loose. Once the variational parameters are initialized well with another deep hierarchical model, the parameters of a DBM may be fitted to give a tighter variational lower-bound which facilitates jointly estimating all parameters.

The proposed two-stage pretraining algorithm provides a general framework in which many hierarchical deep learning models can be used. It even makes possible to include the conventional pretraining algorithm as a part of the proposed algorithm and improve upon it. This is a significant step in developing and improving a training algorithm for DBMs, as it allows us to fully utilize other learning algorithms that have been extensively studied previously.

7.1 Future Work

Recently, two additional algorithms for training DBMs have been proposed. In [30] the authors proposed to center the activations of the neurons in a DBM, which is closely related to the previously proposed method of the enhanced gradient for RBMs [9, 10]. They showed that this simple method allows training a DBM without any pretraining, however, without any direct comparison to the method of pretraining.

The authors of [18] and [6] proposed an alternative learning criterion based on the idea of generalized pseudo-likelihood [24]. The alternative criterion does *not* maximize the log-likelihood but maximizes the predictive (approximate) conditional probabilities among all the visible variables of a DBM.

It is important in the future to compare these recently proposed algorithms together with the two-stage pretraining algorithm as well as the conventional pretraining algorithm against each other. In the case of RBMs, the authors in [28] compared various learning criteria such as maximum-likelihood, contrastive divergence, ratio matching [25] and maximum pseudo-likelihood [4], and they found that each algorithm resulted in solutions that are different in multiple aspects. A similar approach of comparing different learning criteria for the DBM will reveal their inductive biases and allow us to choose an appropriate learning algorithm.

[8] It should be noted that these accuracies were computed purely to illustrate the effect of generative training of DBMs , and the reported accuracies are lower than other state-of-the-art accuracies (see, [18] for state-of-the-art accuracies for MNIST and [9] for Caltech-101 Silhouettes).

Acknowledgements. This work was supported by "the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170)".

References

1. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(8), 1798–1828 (2013)
2. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Schölkopf, B., Platt, J., Hoffman, T. (eds.) Advances in Neural Information Processing Systems 19, pp. 153–160. MIT Press, Cambridge (2007)
3. Bengio, Y., Yao, L., Alain, G., Vincent, P.: Generalized denoising auto-encoders as generative models. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 26, pp. 899–907 (2013)
4. Besag, J.: Statistical Analysis of Non-Lattice Data. Journal of the Royal Statistical Society. Series D (The Statistician) 24(3), 179–195 (1975)
5. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus (2006)
6. Brakel, P., Stroobandt, D., Schrauwen, B.: Training energy-based models for time-series imputation. Journal of Machine Learning Research 14, 2771–2797 (2013)
7. Cho, K.: Improved Learning Algorithms for Restricted Boltzmann Machines. Master's thesis, Aalto University School of Science (2011)
8. Cho, K., Raiko, T., Ilin, A.: Parallel tempering is efficient for learning restricted Boltzmann machines. In: Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN 2010), pp. 1–8 (July 2010)
9. Cho, K., Raiko, T., Ilin, A.: Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011), pp. 105–112. ACM, New York (2011)
10. Cho, K., Raiko, T., Ilin, A.: Enhanced gradient for training restricted Boltzmann machines. Neural Computation 25(3), 805–831 (2013)
11. Cho, K., Raiko, T., Ilin, A.: Gaussian-Bernoulli deep Boltzmann machines. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2013), Texas, USA (August 2013)
12. Cho, K., Raiko, T., Ilin, A., Karhunen, J.: A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines. In: NIPS 2012 Workshop on Deep Learning and Unsupervised Feature Learning, Lake Tahoe (December 2012)
13. Cho, K., Raiko, T., Ilin, A., Karhunen, J.: A two-stage pretraining algorithm for deep Boltzmann machines. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 106–113. Springer, Heidelberg (2013)
14. Courville, A., Bergstra, J., Bengio, Y.: A spike and slab restricted Boltzmann machine. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011) (2011)
15. Desjardins, G., Courville, A., Bengio, Y.: On training deep Boltzmann machines. arXiv:1203.4416 (cs.NE) (March 2012)

16. Desjardins, G., Courville, A., Bengio, Y., Vincent, P., Delalleau, O.: Parallel tempering for training of restricted Boltzmann machines. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010). JMLR Workshop and Conference Proceedings, vol. 9, pp. 145–152. JMLR W&CP (2010)
17. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
18. Goodfellow, I., Miraz, M., Courville, A., Bengio, Y.: Multi-prediction deep Boltzmann machines. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 26, pp. 548–556 (December 2013)
19. Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 (cs.NE) (July 2012)
20. Hinton, G.: Training products of experts by minimizing contrastive divergence. Neural Computation 14, 1771–1800 (2002)
21. Hinton, G., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7), 1527–1554 (2006)
22. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006)
23. Honkela, A., Harmeling, S., Lundqvist, L., Valpola, H.: Using kernel PCA for initialisation of variational Bayesian nonlinear blind source separation method. In: Puntonet, C.G., Prieto, A.G. (eds.) ICA 2004. LNCS, vol. 3195, pp. 790–797. Springer, Heidelberg (2004)
24. Huang, F., Ogata, Y.: Generalized pseudo-likelihood estimates for Markov random fields on lattice. Annals of the Institute of Statistical Mathematics 54(1), 1–18 (2002)
25. Hyvärinen, A.: Some extensions of score matching. Computational Statistics & Data Analysis 51(5), 2499–2512 (2007)
26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324 (1998)
27. Luo, H., Carrier, P., Courville, A., Bengio, Y.: Texture modeling with convolutional spike-and-slab RBMs and deep extensions. In: Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2013). JMLR Workshop and Conference Proceedings, vol. 31, pp. 415–423. JMLR W&CP (April 2013)
28. Marlin, B.M., Swersky, K., Chen, B., de Freitas, N.: Inductive principles for restricted Boltzmann machine learning. In: Proceedings of the Thirteenth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2010). JMLR Workshop and Conference Proceedings, vol. 9, pp. 509–516. JMLR W&CP (2010)
29. Martens, J.: Deep learning via Hessian-free optimization. In: Fürnkranz, J., Joachims, T. (eds.) Proceedings of the 27th Internation Conference on Machine Learning (ICML 2010), Haifa, Israel, pp. 735–742 (June 2010)
30. Montavon, G., Müller, K.R.: Deep Boltzmann machines and the centering trick. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) NN: Tricks of the Trade, 2nd edn. LNCS, vol. 7700, pp. 621–637. Springer, Heidelberg (2012)
31. Pascanu, R., Bengio, Y.: Revisiting natural gradient for deep networks. arXiv:1003.0358 (cs.NE) (2013)
32. Raiko, T., Valpola, H., LeCun, Y.: Deep learning made easier by linear transformations in perceptrons. In: Proceedings of the Fifteenth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2012). JMLR Workshop and Conference Proceedings, vol. 22, pp. 924–932. JMLR W&CP (April 2012)

33. Rumelhart, D.E., Hinton, G., Williams, R.J.: Learning representations by back-propagating errors. Nature 323, 533–536 (1986)
34. Salakhutdinov, R.: Learning in Markov random fields using tempered transitions. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I., Culotta, A. (eds.) Advances in Neural Information Processing Systems 22, pp. 1598–1606 (2009)
35. Salakhutdinov, R.: Learning deep Boltzmann machines using adaptive MCMC. In: Fürnkranz, J., Joachims, T. (eds.) Proceedings of the 27th International Conference on Machine Learning (ICML 2010), Haifa, Israel, pp. 943–950 (June 2010)
36. Salakhutdinov, R., Hinton, G.: Deep Boltzmann machines. In: Proceedings of the Twelfth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2009). JMLR Workshop and Conference Proceedings, vol. 5, pp. 448–455. JMLR W&CP (2009)
37. Salakhutdinov, R., Hinton, G.: A better way to pretrain deep Boltzmann machines. In: Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems 25, pp. 2456–2464 (2012)
38. Salakhutdinov, R., Hinton, G.: An effcient learning procedure for deep Boltzmann machines. Neural Computation 24, 1967–2006 (2012)
39. Salakhutdinov, R., Larochelle, H.: Efficient learning of deep Boltzmann machines. In: Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (2011)
40. Salakhutdinov, R., Murray, I.: On the quantatitive analysis of deep belief networks. In: Proceedings of the 25th International Conference on Machine learning (ICML 2008), pp. 872–879. ACM, New York (2008)
41. Schulz, H., Behnke, S.: Learning two-layer contractive encodings. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part I. LNCS, vol. 7552, pp. 620–628. Springer, Heidelberg (2012)
42. Smolensky, P.: Information processing in dynamical systems: foundations of harmony theory. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. foundations, vol. 1, pp. 194–281. MIT Press, Cambridge (1986)
43. Tieleman, T.: Training restricted Boltzmann machines using approximations to the likelihood gradient. In: Proceedings of the 25th Internation Conference on Machine Learning (ICML 2008), pp. 1064–1071. ACM, New York (2008)
44. Tieleman, T., Hinton, G.: Using fast weights to improve persistent contrastive divergence. In: Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009), pp. 1033–1040. ACM, New York (2009)
45. Vincent, P.: A connection between score matching and denoising autoencoders. Neural Computation 23(7), 1661–1674 (2011)
46. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research 11, 3371–3408 (2010)

Training Dynamic Neural Networks Using the Extended Kalman Filter for Multi-Step-Ahead Predictions

Artem Chernodub

Abstract. This paper is dedicated to single-step-ahead and multi-step-ahead time series prediction problems. We consider feedforward and recurrent neural network architectures, different derivatives calculation and optimization methods and analyze their advantages and disadvantages. We propose a novel method for training feedforward neural networks with tapped delay lines for better multi-step-ahead predictions. Special mini-batch calculations of derivatives called Forecasted Propagation Through Time for the Extended Kalman Filter training method are introduced. Experiments on well-known benchmark time series are presented.

Keywords: multi-step-ahead prediction, mini-batch Extended Kalman Filter, Forecasted Propagation Through Time, Backpropagation Through Time.

1 Introduction

Dynamics are everywhere around us. Economic and financial processes, ecological systems, industrial systems, weather fluctuations, and any kind of control system has internal dynamics governing its behavior. Even life is a dynamic system that consists of processing the information input from our senses and producing reactions to the outer environment. Surely, dealing with dynamics is required to perform many real-life tasks. Dynamic and static systems have an important difference: whereas in static systems all events are independent, dynamical systems have a memory of past events. This frequently makes solving dynamic problems much more difficult because errors are independent in the static case. For example, a single misclassification in an automated product sorting facility likely will

Artem Chernodub
Institute of Mathematical Machines and Systems NASU, Neurotechnologies Dept., Glushkova 42 ave., 03187 Kyiv, Ukraine

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_11

not significantly change the overall outcome of the task. In the dynamic case if the plant is unstable, errors can snowball: a single incorrect control action could result in a catastrophic accident. Good prediction quality is essential for successful control in any dynamic plant.

Neural networks are an effective and friendly tool for black-box modeling of a plant's unknown dynamics [1-3]. The neural network may be trained on examples of the dynamic plant's recorded output and then be exploited for prediction of new values. Usually, neural networks are trained to perform single-step-ahead (SS) predictions, where the predictor uses some available input and output observations to estimate the variable of interest for the time step immediately following the latest observation [4-6]. Knowing only the next nearest value is enough for many problems. However, recently there has been growing interest in multi-step-ahead (MS) predictions, where the values of interest must be predicted for some horizon in the future. Knowing the sequence of future values allows for estimation of projected amplitudes, frequencies, and variability, which are important for Model Predictive Control [7], modeling flood forecasts [8] and fault diagnostics [9]. Generally speaking, the ability to perform MS predictions is frequently treated as the "true" test for the quality of a developed empirical dynamic model. In particular, well-known echo state machine neural networks (ESNs) became popular because of their ability to perform good long-horizon $(H = 84)$ multistep predictions [10].

This chapter is dedicated to time series prediction methods using dynamic neural networks. In Section 1 we consider a system identification problem that uses statistical methods to build empirical models of dynamic plants from the measured data. We formulate single-step-ahead (SS) and multi-step-ahead (MS) prediction problems using the developed empirical models. In Section 2 we consider different perceptron-like models of dynamic neural networks including feedforward (FFN) and recurrent (RNN) neural networks and provide an analysis of their strengths and weaknesses. We discuss methods for calculating static and dynamic (BPTT) derivatives for these architectures. In Section 3 we briefly consider the vanishing gradient effect and its impact on neural networks' long-term learning capabilities. In Section 4 we consider optimization algorithms for training these neural networks. We describe online and mini-batch implementations of the second-order Extended Kalman Filter (EKF) algorithm for training neural networks. We compare the EKF algorithm with the standard first-order gradient descent-based backpropagation algorithm. In Section 5 we propose a new method for training feedforward neural models to perform MS prediction, called Forecasted Propagation Through Time (FPTT). FPTT allows calculation of batch-like dynamic derivatives while minimizing the negative effect of vanishing gradients. We use mini-batch modification of the EKF algorithm that naturally deals with these batch-like dynamic derivatives for training the neural network. We present experiments on SS and MS predictions using well-known time series benchmarks.

2 Modeling the Dynamic Systems

Consider a dynamic plant:

$$\mathbf{S}(k+1) = \Phi(\mathbf{S}(k), \mathbf{u}(k)), \tag{1}$$

$$\mathbf{y}(k+1) = \Psi(\mathbf{S}(k)), \tag{2}$$

where $\mathbf{S}(k)$ is a state vector, $\mathbf{u}(k)$ is a plant's input, $\mathbf{y}(k+1)$ is an observable plant's output, and $\mathbf{\Phi}(\cdot)$ i $\mathbf{\Psi}(\cdot)$ are some nonlinear functions. The identification problem in the general case may be stated as follows: to identify *a priori* unknown functions $\Phi(\cdot)$ and $\Psi(\cdot)$ using the known history of inputs $\{\mathbf{u}(k)\}_{k=1}^{T}$ and the measured outputs $\{\mathbf{y}(k)\}_{k=1}^{T}$.

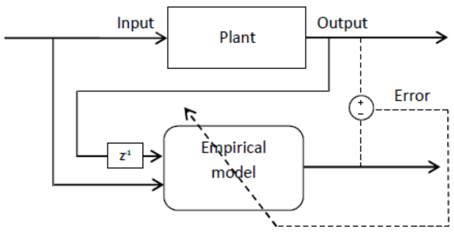


Fig. 1 General scheme of plant's identification

In this chapter we consider identification of the dynamic plants in the sense that we build empirical models of the dynamic plants, predicting time series using the concept of nonlinear autoregression (NAR). To use NAR we must make two additional important assumptions: 1) the plant's inputs $\{\mathbf{u}(k)\}_{k=1}^{T}$ are not observable and 2) the plant's states $\mathbf{S}(k)$ may be expressed as a function of the last N observable outputs of the plant:

$$\mathbf{S}(k) = \Omega(\mathbf{y}(k), \ldots., \mathbf{y}(k-N+1)). \tag{3}$$

where k is the time step variable and $\Omega(\cdot)$ is an unknown function that defines the underlying dynamic process. Now it is possible to capture the plant's behavior using the following equation:

$$\mathbf{y}(k+1) = F(\mathbf{y}(k), \mathbf{y}(k-1), \ldots, \mathbf{y}(k-N+1)). \tag{4}$$

Empirical models such as neural networks may be trained on known inputs and outputs of dynamic processes and then be used for estimation of new outputs using the unknown inputs (Fig. 1). By training the neural network we mean tuning the neural network's free parameters (weights). The goal of training is to develop the empirical model $\tilde{F}(\cdot)$ of function $F(\cdot)$ as closely as possible.

If such an empirical model $\tilde{F}(\cdot)$ is available, one can perform single-step-ahead (SS) predictions $\tilde{\mathbf{y}}(\cdot)$ using the measured outputs $\mathbf{y}(\cdot)$:

$$\tilde{\mathbf{y}}(k+1) = \tilde{F}(\mathbf{y}(k), \mathbf{y}(k-1), ..., \mathbf{y}(k-N+1)). \tag{5}$$

Meanwhile, since we are working with autoregression models, we can perform iterated multi-step-ahead (MS) predictions $\hat{\mathbf{y}}(\cdot)$ using both the measured outputs $\mathbf{y}(\cdot)$ and the neural network's SS $\hat{\mathbf{y}}(\cdot)$ and MS estimations $\tilde{\mathbf{y}}(\cdot)$ if real outputs are not available:

$$\tilde{\mathbf{y}}(k+2) = \tilde{F}(\hat{\mathbf{y}}(k+1), \mathbf{y}(k), ..., \mathbf{y}(k-N)), \tag{6}$$

$$\dots$$

$$\tilde{\mathbf{y}}(k+H) = \tilde{F}(\tilde{\mathbf{y}}(k+H-1), \tilde{\mathbf{y}}(k+H-2), ..., \tilde{\mathbf{y}}(k+H-N)), \tag{7}$$

$$\tilde{\mathbf{y}}(k+H+1) = \tilde{F}(\tilde{\mathbf{y}}(k+H), \tilde{\mathbf{y}}(k+H-1), ..., \tilde{\mathbf{y}}(k+H-N+1)), \tag{8}$$

where H is the horizon of prediction. Here and later we emphasize the differences between SS $\tilde{\mathbf{y}}(\cdot)$ and MS $\hat{\mathbf{y}}(\cdot)$ predictions, $\hat{\mathbf{y}}(k+H) \equiv \tilde{\mathbf{y}}(k+H)$ for $H=1$. We can also now define the Normalized Mean Squared Error, NMSE:

$$NMSE = \sum_k \frac{(\mathbf{t}(k) - \tilde{\mathbf{y}}(k))^2}{(\mathbf{t}(k) - \bar{\mathbf{t}}(k))^2}, \tag{9}$$

where $\mathbf{t}(\cdot)$ are target values, $\bar{\mathbf{t}}(k)$ is mean target value, $\tilde{\mathbf{y}}(\cdot)$ are predicted values.

3 Dynamic Neural Networks

Early neural network architectures (Adaline [11], Rosenblatt's Perceptron [1, p. 78], Multilayer Perceptron [1, p. 152]) were developed for solving static pattern recognition problems only. The dynamic neural networks for time series discussed in this chapter are adaptations of feedforward neural networks for pattern recognition. There are two main approaches for performing such "dynamization" of the static neural networks: 1) adding a tapped delay line to the network's inputs and 2) adding recurrent connections to the network's topology.

In the first case (Dynamic Linear Neural Network, Dynamic Multilayer Perceptron) the neural network receives the previous inputs, delayed in time, together with the current input. Training the neural network usually is performed using the well-known backpropagation method for calculating the derivatives and various gradient-based optimization methods. Advantages of this scheme are its simplicity and convenience. Our estimation is that this scheme is used in more than 90% of cases. On the other hand, the number of delays in the tapped delay lines must be set *a priori*. If these properties do not correlate with the order of the underlying dynamic process, the neural network achieves poor results. Another disadvantage of this scheme is bad quality in the iterated multi-step-ahead predictions.

Another basic way to implement dynamics inside a feedforward neural network is adding internal recurrent connections to the input, hidden or output neurons (Recurrent Multilayer Perceptron, Elman Neural Network, etc.) A special methodology for calculating dynamic derivatives must be provided to take into account the influence of the previous time steps on the current state. Dynamic derivatives can be calculated using two techniques: Real-Time Recurrent Learning (RTRL) or Backpropagation Through Time (BPTT). Then, any gradient-based optimization algorithm for tuning weights may be used. Recurrent neural networks are more suitable for modeling the dynamic process's structure. However, training RNNs is a much more complex problem than training the FFNNs because of the additional degrees of freedom and more complicated error surface. Moreover, during calculation of the dynamic derivatives for neural networks with sigmoidal activation functions, the vanishing gradient effect occurs. This effect additionally complicates the detection of correlations between the neural network's previous inputs and current target outputs.

In this section we consider the most popular models of perceptron-like dynamic neural networks. Note that all neural network schemes described here were adopted for the autoregression case. Also, for simplicity we always assume that dimensionality of the modeled plant's output is 1, i.e. the neural network has only a single output neuron.

3.1 Dynamic Linear Neural Network

Dynamic Linear Neural Network, DLNN [2, p. 644] is the simplest model of a dynamic neural network. It consists of a single-layer perceptron and a time delay line (TDL) of order N (Fig. 2). Dynamic Linear Neural Network may be effectively used for the identification of linear dynamic plants only.

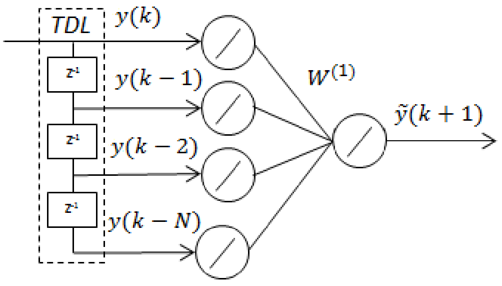


Fig. 2 Dynamic Linear Neural Network

The Neural Network receives the input

$$\mathbf{x}(k) = [\mathbf{y}(k) \quad \mathbf{y}(k-1) \quad \ldots \quad \mathbf{y}(k-N)]^T, \tag{10}$$

and calculates the output

$$\tilde{\mathbf{y}}(k+1) = \sum_{i=1}^{N_w} x_i(k) w_i^{(1)}(k), \tag{11}$$

where N_w is number of NN's weights. Error gradients $\frac{\partial E(k)}{\partial \mathbf{w}^{(1)}}$ for DLNN are calculated as:

$$\frac{\partial E(k)}{\partial w_i^{(1)}} = x_i(k) w_i^{(1)}. \tag{12}$$

3.2 Dynamic Multilayer Perceptron

The Dynamic Multilayer Perceptron [2, p. 665] (DMLP) is a much more powerful neural network architecture than the DLNN. It consists of a multilayer perceptron with non-linear units and a time delay line of order N (Fig. 3).

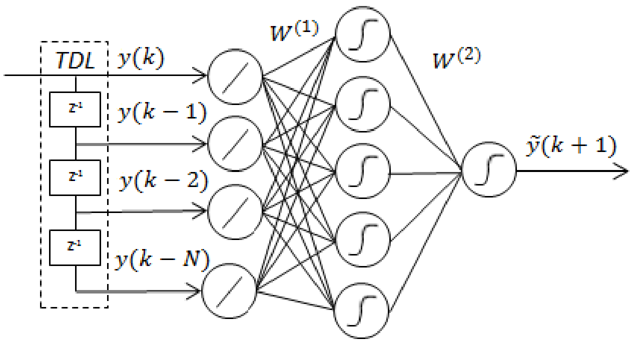


Fig. 3 Dynamic Multilayer Perceptron

DLNN receives the input vector $\mathbf{x}(k)$ (10) and calculates the output $\tilde{\mathbf{y}}(k+1)$ as:

$$z_j = f(\sum_i w_{ji}^{(1)} x_i), \tag{13}$$

$$\tilde{y}(k+1) = g(\sum_j w_j^{(2)} z_j), \tag{14}$$

where z_j is the postsynaptic value for the j-th hidden neuron, $\mathbf{w}^{(1)}$ are the hidden layer's weights, $f(\cdot)$ are the hidden layer's activation functions, $\mathbf{w}^{(2)}$ are the output layer's weights, and $g(\cdot)$ are the output layer's activation functions.

Error gradients $\frac{\partial E(k)}{\partial \mathbf{w}^{(1)}}$ and $\frac{\partial E(k)}{\partial \mathbf{w}^{(2)}}$ for the DMLP are calculated using the backpropagation technique. The procedure is the same as for the static case. Local gradients for the hidden layer $\boldsymbol{\delta}^{HID}$ are calculated and local gradients for input layer $\boldsymbol{\delta}^{IN}$ are calculated on demand as follows:

$$\delta^{OUT} = t(k+1) - \tilde{y}(k+1), \tag{15}$$

$$\delta_j^{HID} = f'(z_j) w_j^{(2)} \delta^{OUT}, \tag{16}$$

$$\delta_i^{IN} = \sum_{n=1}^{K} w_{ni}^{(1)} \delta_n^{HID}, \tag{17}$$

where $t(k+1)$ is the target value, and K is number of neurons in the hidden layer. Then the error gradients are calculated:

$$\frac{\partial E(k)}{\partial w_j^{(2)}} = \delta^{OUT} z_j, \tag{18}$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(1)}} = \delta_j^{IN} x_i. \tag{19}$$

3.3 Recurrent Multilayer Perceptron

The Recurrent Multilayer Perceptron (RMLP) is an adaptation of the multilayer perceptron. It is made dynamic by adding recurrent connections to the hidden layer, a so-called “context layer”.

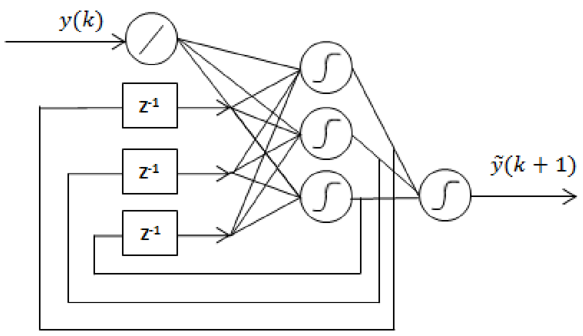


Fig. 4 Recurrent Multilayer Perceptron

The RMLP's input vector $\mathbf{x}(k)$ is:

$$\mathbf{x}(k) = [y(k) \quad z_1(k-1) \quad z_2(k-1) \quad \ldots \quad z_K(k-1)]^T, \tag{20}$$

where z_j is the postsynaptic value of the j-th neuron of the hidden layer, and K is the number of hidden neurons.

To consider the influence of the previous time steps to the current error residual, special dynamic Backpropagation Through Time derivatives [1, p. 836] are calculated, described in Fig. 4. An alternative for the BPTT method for calculating the dynamic derivatives is Real Time Recurrent Learning [1, p. 840]. RTRL is now rarely used because it requires more computational resources than BPTT, yet is less accurate.

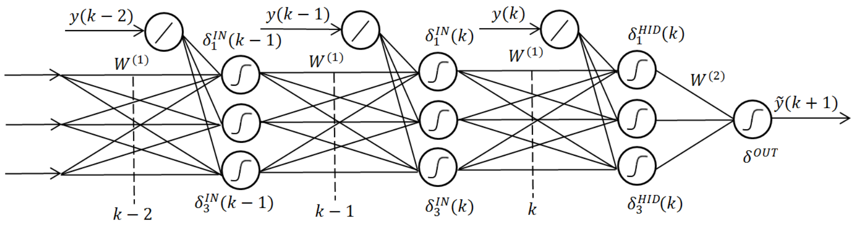


Fig. 5 Calculation of dynamic BPTT derivatives for RMLP, truncation depth h = 2

After calculating the output, the neural network is unfolded back through time (Fig. 5). The recurrent neural network is presented as a feedforward neural network with many layers, each corresponding to one of the retrospective time steps $k-1, k-2, \ \ldots, \ k-h$, where h is the BPTT truncation depth. The hyperparameter h corresponds to N, the order of the time delay line in the DMLP. Derivatives are calculated using the standard backpropagation method. Local gradients for backpropagation are calculated using the following equations:

$$\delta_j^{HID}(k) = f_j{}'(k) w_j^{(2)} \delta^{OUT}, \tag{21}$$

$$\delta_j^{IN}(k) = f_j{}'(k-1) \sum_{i=1}^{K} w_{ij}^{(1)} \delta_i^{HID}(k), \tag{22}$$

$$\delta_j^{IN}(k-n) = f_j{}'(k-n-1) \sum_{i=1}^{K} w_{ij}^{(1)} \delta_i^{IN}(k-n+1), \tag{23}$$

where $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ are weights of the hidden and output layer, $\boldsymbol{\delta}_j^{HID}$ is the local gradient for the j-th neuron of the hidden layer, $\boldsymbol{\delta}_j^{IN}(k-n)$ is the local gradient for the hidden layer at retrospective time step $k-n$, $1 \le n \le h$, and h is the truncation depth.

Error gradients for the output layer $\frac{\partial E_{BPTT}(k)}{\partial \mathbf{w}^{(2)}} = \frac{\partial E(k)}{\partial \mathbf{w}^{(2)}}$ are calculated using (16). For the hidden layer $\frac{\partial E_{BPTT}(k)}{\partial \mathbf{w}^{(1)}}$, the procedure is more complex and has two stages: First, gradients for each unfolded layer are calculated for each retrospective time step $k-n$. Second, the BPTT derivatives are calculated as averaged static gradients:

$$\frac{\partial E_{BPTT}(k)}{\partial w_{ji}^{(1)}} = \frac{1}{h}\sum_{n=0}^{h} \frac{\partial E(k-n)}{\partial w_{ji}^{(1)}}. \tag{24}$$

3.4 NARX Neural Network

The NARX neural network is a dynamic neural network with two main approaches for implementing its dynamics. It is a multilayer perceptron that is equipped with both a tapped delay line at the input and global recurrent output feedback connections (Fig. 6).

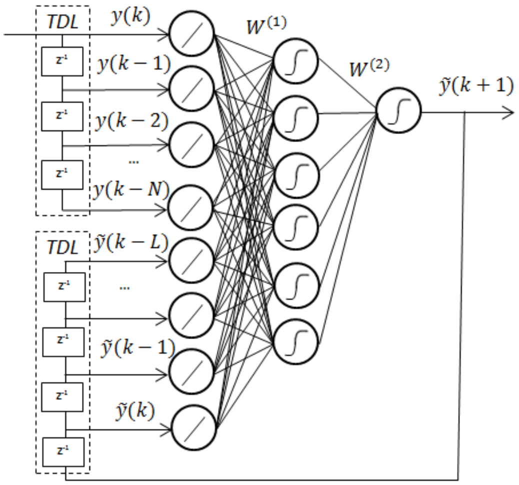


Fig. 6 NARX neural network

As it was shown in [12], NARX networks with sigmoidal activation functions are universal approximators of dynamic systems, i.e. in theory they can simulate any Turing machine. The input vector $\mathbf{x}(k)$ is:

$$\mathbf{x}(k) = [y(k) \quad ... \quad y(k-N) \quad \tilde{y}(k) \quad ... \quad \tilde{y}(k-L)]^T, \tag{25}$$

where N is an order of the time delay line for the input values, and L is the order of the time delay line for the recurrent feedback connections. The scheme for calculating the dynamic BPTT derivatives for NARX networks is shown in Fig. 7. Similarly to the RMLP, the NARX network must be unfolded back though time; it is presented as feedforward neural network with many layers, each layer corresponding to one of the previous time steps $k-1, k-2, \ldots, k-h$, and error is propagated back through this feedforward network.

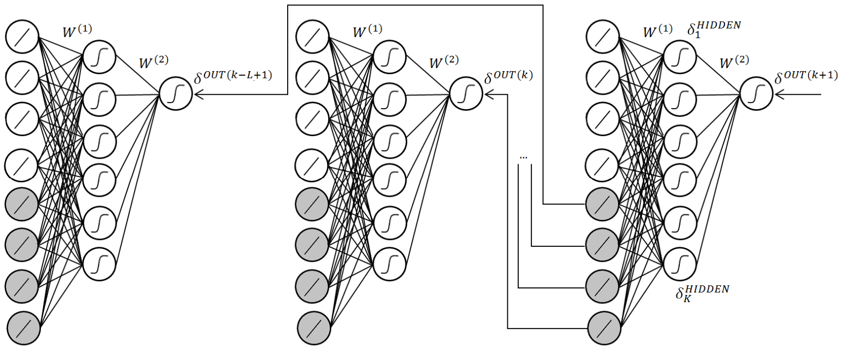


Fig. 7 Calculating the dynamic BPTT derivatives for the NARX neural network, truncation depth h = 1

During the backpropagation pass, local gradients are calculated as follows:

$$\delta_j^{HID} = f_j'(k) w_j^{(2)} \delta^{OUT(k+1)}, \tag{26}$$

$$\delta_j^{IN} = f_j'(k-1) \sum_{i=1}^{K} w_{ij}^{(1)} \delta_i^{HID}, \tag{27}$$

$$\delta^{OUT(k-l+1)} = \delta_{N+l}^{IN}, \tag{28}$$

where δ_j^{HID} is the local gradient for the j-th hidden neuron, δ_j^{IN} is the local gradient for the l -th input neuron, $1 \le l \le L$, L is the order of the time delay line for the recurrent connections, $\delta_j^{OUT(n)}$ is the output local gradient for the n-th step back through time. The final dynamic gradients for both layers $\frac{\partial E_{BPTT}(k)}{\partial \mathbf{w}^{(1)}}$ and $\frac{\partial E_{BPTT}(k)}{\partial \mathbf{w}^{(2)}}$ are averaged static gradients $\frac{\partial E(k)}{\partial \mathbf{w}^{(1)}}$ and $\frac{\partial E(k)}{\partial \mathbf{w}^{(2)}}$.

3.5 Experiment on SS Predictions with Different Network Types

In order to test the dynamic neural network architectures described above, we prepared 6 well-known time series: “MG17”, “ICE”, “SOLAR”, “CHICKENPOX”, “RIVER”,”LASER” (Fig. 8).

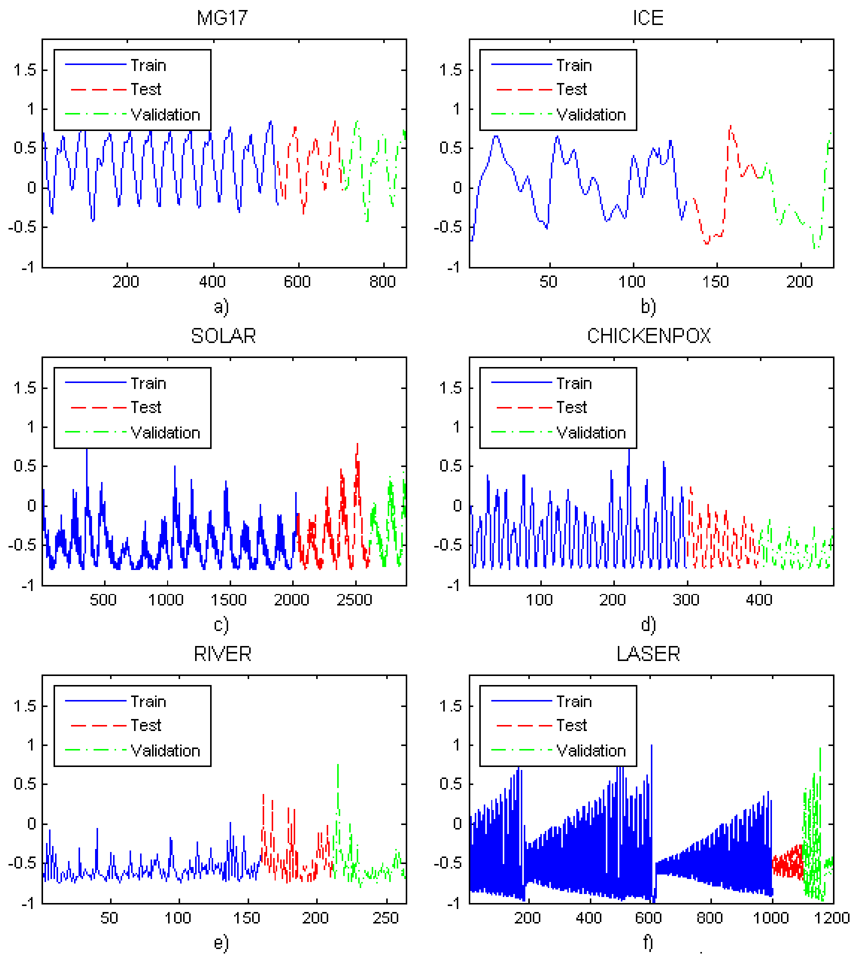


Fig. 8 Datasets for the SS-prediction experiment. a) Mackey-Glass chaotic series. b) Global ice volume dataset. c) River flow dataset. d) Monthly chickenpox instances dataset. e) River flow dataset. f) Santa-Fe Laser Dataset

The dataset "MG17" is the Mackey-Glass chaotic process. It is a famous benchmark for time series predictions. The discrete-time equation is given by the following difference equation (with delays):

$$x_{t+1} = (1-b)x_t + a\frac{x_{t-\tau}}{1+(x_{t-\tau})^{10}}, t = \tau, \tau+1, \dots \ , \tag{29}$$

where $\tau \geq 1$ is an integer. We used the following parameters: $a = 0.1$, $b = 0.2$, $\tau = 17$ as in [6]. The dataset "ICE" represents 219 measurements of global ice volume over the last 440,000 years [13]. The dataset "SOLAR" consists of recording 2899 months of mean solar sunspots. The dataset "CHECKPOX" represents 498 months of chickenpox cases in New York City for 1931-1972 [14]. The dataset "RIVER" consists of the monthly river flows of the Sacramento River [14]. The dataset "LASER" consists of far-infrared laser intensity over a period of chaotic activity, taken from the Santa Fe competition.

For each dataset, we trained 100 DLNN, DMLP, RMLP and NARX networks using the Extended Kalman Filter (EKF) method (see Section 4). The training parameters for the EKF were set to $\eta = 10^{-3}$ and $\mu = 10^{-8}$. The number of neurons in the hidden layer for MLP-based networks was varied from 3 to 8, the order of input tapped delay line was set to $N = 5$ for the DLNN and DMLP and $N = 5$, $L = 5$ for NARX. The initial weights were set to small random values. Each network was trained for 50 - 500 epochs depending on the dataset. The best performing network on the 'Test' sequence was then tested on the 'Validation' subset of the data. The final results are presented in Table 1. The DLNN is only performing linear regression and so is not a serious competitor to the nonlinear MLP-based architectures; it is given for comparison.

Table 1 Mean NMSE Single-Step-Ahead predictions for different neural network architectures

	MG17	ICE	SOLAR	CHICKENPOX	RIVER	LASER
DLNN	0.0042	0.0244	0.1316	0.8670	4.3638	0.7711
DMLP	0.0006	0.0376	0.1313	0.4694	0.9979	0.0576
RMLP	0.0050	0.0273	0.1493	0.7608	6.4790	0.2415
NARX	0.0010	0.1122	0.1921	1.4736	2.0685	0.1332

We can see that in most cases, feedforward DMLP networks outperform recurrent networks for SS predictions. However, the DLNN shows approximately the same quality as the DMLP for the SOLAR dataset, indirectly confirming the hypothesis that the dynamic process generating solar spots is actually linear in nature. For recurrent networks it is not possible to say which architecture is significantly better: the highest and lowest performing networks vary between datasets and prediction quality may differ by 2-5 times.

4 The Vanishing Gradient Effect

The vanishing gradient effect [1, p. 846], [15], [16] significantly complicates training of ordinary perceptron-like recurrent neural networks. As described in Section 2, recurrent neural networks are first considered feedforward neural networks with many layers, each layer corresponding to one retrospective time step, in order to calculate the dynamic derivatives. When the error is propagated back through the layers the absolute values of the target derivatives exponentially decrease. This makes the detection of long-term dependencies between events difficult. To understand the internal mechanics of the vanishing gradient effect, consider the provisional feedforward multilayer perceptron that has N layers (Fig. 9):

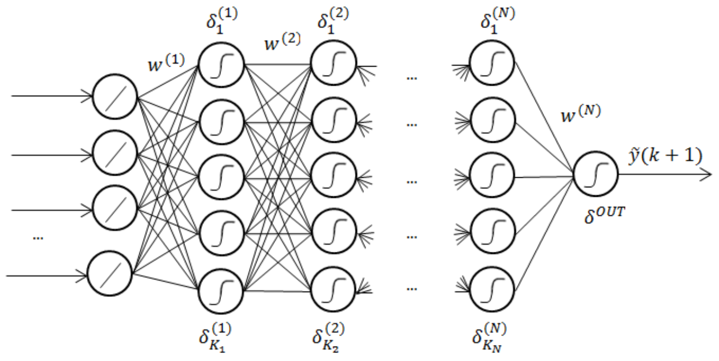


Fig. 9 Scheme of *N*-layer perceptron

Calculation of the error gradients $\frac{\partial E(k)}{\partial \mathbf{w}}$ or Jacobians $\frac{\partial \tilde{\mathbf{y}}(k+1)}{\partial \mathbf{w}}$ must be performed during the backward pass. Since we use backpropagation for calculating the derivatives for training, all relevant local gradients $\boldsymbol{\delta}^{OUT} = \mathbf{e}(k)$ or $\boldsymbol{\delta}^{OUT} = \mathbf{1}$ must be propagated back through N layers respectively. Error gradients are the product of the local gradients $\delta_j^{(m)}$ and the corresponding values $z_i^{(m-1)}$:

$$\frac{\partial E(k)}{\partial w_{ji}^{(m)}} = \delta_j^{(m)} z_i^{(m-1)}. \tag{30}$$

The Jacobians $\frac{\partial \tilde{\mathbf{y}}(k+1)}{\partial \mathbf{w}}$ for the neural network's training procedure are calculated using the standard backpropagation technique by propagating a constant value $\boldsymbol{\delta}^{OUT} = \mathbf{1}$ at each backward pass instead of propagating the residual error $\boldsymbol{\delta}^{OUT} = \mathbf{t}(k+1) - \tilde{\mathbf{y}}(k+1)$, calculating Jacobians instead of error gradients since $\frac{\partial E(k)}{\partial \mathbf{w}} = \frac{\partial [\mathbf{e}(k)^2]}{\partial \mathbf{w}} = 2\mathbf{e}(k)\frac{\partial \tilde{\mathbf{y}}(k+1)}{\partial \mathbf{w}}$.

Local gradients $\boldsymbol{\delta}^{(m)}$ are calculated for each neuron layer, starting from neuron layer $m = N$ and finishing with neuron layer $m = 1$:

$$\delta_j^{(m)} = f_j^{(m-1)\,\prime} \sum_i w_{ij}^{(m)} \delta_i^{(m+1)}. \tag{31}$$

In practice, it was found that for neural networks with $N > 2$ layers, the gradients (Jacobians) vanish, i.e. $\frac{\partial E(k)}{\partial w_{ji}^{(m)}} \to 0$ (or $\frac{\partial \tilde{y}(k+1)}{\partial w_{ji}^{(m)}} \to 0$) for $m \to 1$. Therefore, updates to the weights become very small and the neural network cannot be trained properly.

The origins of the vanishing gradient effect lie in the calculation of the local gradients (30). During backpropagation, the absolute values of each local gradient are frequently smaller than those of the previous local gradient because they are the product of values which are all less than or equal to 1. We know that the values will always be less than or equal to 1 since the initial local gradients $\boldsymbol{\delta}^{OUT}$ never contain values greater than 1, the neural network's weights $w_{ij}^{(m)}$ cannot be large to avoid overfitting, and the derivatives of the activation functions $f_j^{(m-1)}$ are always less than 1. Moreover, in [16] it was proven that if overfitting occurs, the case where $\left| f_j^{(m-1)\,\prime} w_{ij}^{(m)} \text{weights}\, \delta_i^{(m+1)} \right| < 1.0$, the vanishing gradient effect still occurs. This happens because the relevant derivative goes to zero faster than the absolute weight can grow and local gradients $\delta^{(m)}$ exponentially decrease as a function of the current layer's number m.

For small, static pattern recognition problems a single hidden layer is often enough, so the vanishing gradient effect does not have a significant impact on the training quality. However, if one works with training datasets that contain hundreds of millions of high-dimensional training samples, the impact of the vanishing gradient effect is substantial. To use such data, one might need neural networks with many layers and millions of weights. The vanishing gradient effect plays a key role in these "Deep Neural Networks" which usually have 6-9 layers; a special "pre-training" procedure was invented [17] to avoid this effect in static networks. Unrolled dynamic recurrent neural networks may have 20-30 layers [9], so the importance of the vanishing gradient effect cannot be underestimated.

5 Optimization Methods for Dynamic Neural Network Training

There are many methods for training neural networks. Their key properties are calculating the weights' derivatives, which strongly depends on the neural network's topology and selected optimization algorithm. Here we consider gradient-based optimization methods based on backpropagation. There are two understandings of term “backpropagation” – in a broad and in a narrow sense. In a broad sense, backpropagation is a method for calculating the derivatives of a neural network and a procedure for correcting the weights based on a gradient descent optimization technique. In a more narrow sense, which we will use here, backpropagation refers only to the technique for calculating derivatives.

5.1 Gradient Descent

Gradient descent is a first-order optimization algorithm. It is widely used for training neural networks due to its simplicity and low computational cost. Its disadvantages are low convergence speed and high risk of stopping in a local minimum. The idea of training is to move in the direction of the negative gradient in the weights space:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha\ \Delta\mathbf{w}(k), \tag{32}$$

$$\Delta\mathbf{w}(k) = -\frac{\partial E(k)}{\partial \mathbf{w}(k)}, \tag{33}$$

where α is training speed, $\mathbf{w}$ are weights coefficients, and $\frac{\partial E(k)}{\partial \mathbf{w}(k)}$ static or dynamic derivatives depending on the network’s topology.

5.2 Extended Kalman Filter Training

The extended Kalman Filter method was proposed in the late 1980s [18] as an effective and efficient tool for supervised training of neural networks. Kalman Filter training shows much better fitting accuracy and faster convergence in comparison to the gradient descent-based methods. It is based on second order derivative information about the error surface, which is accumulated in the covariance matrix. This makes the Kalman Filter a good alternative to the popular second-order batch training methods such as conjugate gradient, BFGS, Levenburg-Marquardt, etc. Meanwhile, Kalman Filter methods are online (sample-by-sample), offering additional benefits. First, it makes it possible to operate in real-time. Second, KF training is less likely to converge to a local minimum due to the stochastic component in the training process [1, p. 24]. Moreover, it is not necessary to implement the regularization procedure here to decrease overfitting, because it is already implicitly implemented inside the Kalman recursion [19]. Mentioned above, a pioneering paper by S. Singhal and L. Wu produced a family of training methods called “Bayesian filtering for parameter estimation” [1], [19]. Several methods were developed: decoupled EKF training for reducing computational resources required [1, p. 855], [20, p. 33], [21], multi-stream [20, p. 35] and mini-batch [22], [23] training for escaping local minima. New designs of Kalman

Filters were used: The Ensemble Kalman Filter, EnKF [24], Unscented Kalman Filter, UKF [20, p. 221], [25], Cubature Kalman Filter, CKF) [1, p. 787], [26] and their more numerically stable square-root implementations [1, p. 773], [20, p. 273], etc. Although KF methods are usually used for training RNN networks, they can be applied to any differentiable model of a neural network including Multilayer Perceptrons, RBF networks, belief networks and others.

The Kalman Filter deals with the dynamics of the training process, so the existence of recurrent connections in the network's topology is not a necessary condition. Training the neural network is considered a state estimation problem of some unknown "ideal" neural network that provides zero residual. In this case, the states are the neural network's weights $\mathbf{w}(k)$, and the residual is the current training error $\mathbf{e}(k)$,

$$e(k) = t(k+1) - \tilde{y}(k+1), \tag{34}$$

where $t(k+1)$ is a target and $\tilde{y}(k+1)$ is NN's output.

The dynamic training process can be described by equations (33) and (34) using the state space model. The state transition equation (33) describes the evolution in time of the neural network's weights $\mathbf{w}(k)$ under the influence of the random Gaussian process $\xi(n)$ with zero mean and diagonal covariance matrix $\mathbf{Q}$:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \xi(k). \tag{35}$$

Measurement equation (34) is considered a linearized NN model at the time step k. It is noised by random Gaussian noise $\zeta(k)$ with zero mean and known covariance matrix $\mathbf{R}$:

$$\mathbf{h}(k) = \frac{\partial \mathbf{y}(\mathbf{w}(k), \mathbf{z}(k), \mathbf{x}(k))}{\partial \mathbf{w}} + \zeta(k), \tag{36}$$

where $\mathbf{w}(k)$ are weights, $\mathbf{z}(k)$ are postsynaptic values , $\mathbf{x}(k)$ are input values of the neural network. The Jacobians $\frac{\partial \mathbf{y}}{\partial \mathbf{w}}$ for the neural network's training procedure are calculated using the standard backpropagation procedure by propagating the value $\delta^{OUT} = 1$ at each backward pass.

During the initialization step, the covariance matrices of measurement noise $\mathbf{R} = \eta \mathbf{I}$ and dynamic training noise $\mathbf{Q} = \mu \mathbf{I}$ are set. Matrix $\mathbf{R}$ has size $O \times O$ and matrix Q has size $N_w \times N_w$, where O is the number of output neurons (we assume $O = 1$) and N_w is the number of weight coefficients. Coefficient η is inverse to the training speed, usually $\eta \sim 10^{-2} ... 10^{-4}$, and coefficient μ defines the measurement noise, usually $\mu \sim 10^{-4} ... 10^{-8}$. Also, the identity covariance matrix $\mathbf{P}$ of size $N_w \times N_w$ and zero observation matrix $\mathbf{H}$ of size $O \times N_w$ are defined.

The following steps must be performed for all elements of the training dataset:
1) Forward pass: the neural network's output $\tilde{\mathbf{y}}(k+1)$ is calculated.

2) Backward pass: Jacobians $\frac{\partial \tilde{\mathbf{y}}}{\partial \mathbf{w}}$ are calculated using backpropagation. Observation matrix $\mathbf{H}(k)$ is filled:

$$\mathbf{H}(k) = \left[\frac{\partial \tilde{y}(k+1)}{\partial w_1} \quad \frac{\partial \tilde{y}(k+1)}{\partial w_2} \quad \dots \quad \frac{\partial \tilde{y}(k+1)}{\partial w_{N_w}} \right]. \tag{37}$$

3) Residual matrix $\mathbf{E}(k)$ is filled:

$$\mathbf{E}(k) = [t(k+1) - \tilde{y}(k+1))]. \tag{38}$$

4) New weights $\mathbf{w}(k)$ and correlation matrix $\mathbf{P}(k+1)$ are calculated:

$$\mathbf{K}(k) = \mathbf{P}(k)\mathbf{H}(k)^T [\mathbf{H}(k)\mathbf{P}(k)\mathbf{H}(k)^T + \mathbf{R}]^{-1}, \tag{39}$$

$$\mathbf{P}(k+1) = \mathbf{P}(k) - \mathbf{K}(k)\mathbf{H}(k)\mathbf{P}(k) + \mathbf{Q}, \tag{40}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{K}(k)\mathbf{E}(k). \tag{41}$$

5.3 *Experiment: Gradient Descent vs Extended Kalman Filter*

In order to compare the sequential optimization algorithms we trained 100 Dynamic Multilayer Perceptrons on the Mackey-Glass chaotic process (please, see Section 2.5 for details). The training speed for the Gradient Descent $\alpha = 0.01$, the training parameters for the EKF were set to $\eta = 10^{-3}$ and $\mu = 10^{-8}$. Initial weights of all neural networks were exactly the same for the both methods.

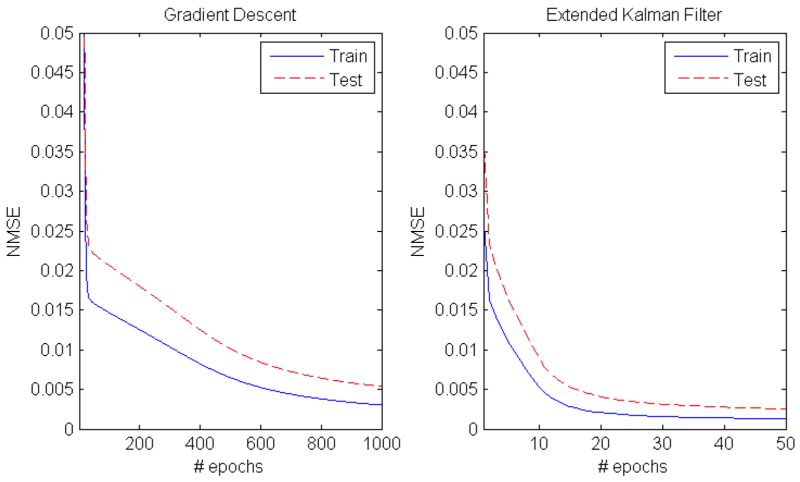


Fig. 10 Left: training 100 DMLPs with the Gradient Descent. Right: training 100 DMLPs with the Extended Kalman Filter.

The evolution of errors in time situation shown on Fig. 10 is typical for such comparison: we can see that EKF converges extremely faster than GD. Final test error was 0.0055 for GD vs 0.0026 for EKF (validation step was not performed in this experiment). At the same time, total training time of 100 DMLPs was 100.54 seconds for GD versus 7.18 seconds for EKF.

5.4 Mini-Batch Extended Kalman Filter Training

The EKF training algorithm also has a mini-batch form [23]. In this case, a batch size of B patterns and a neural network with O outputs is treated as training a single shared-weight network with $O \times B$ outputs, i.e. B data streams which feed B networks constrained to have identical weights are formed from the training set. A single weight update is calculated and applied equally to each stream's network. This weights update is sub-optimal for all samples in the mini-batch. If streams are taken from different places in the dataset, then this trick becomes equivalent to a Multistream EKF [21], a well-known technique for avoiding poor local minima. The mini-batch observation matrix $\mathbf{H}_{BATCH}(k)$ and residual matrix $\mathbf{E}_{BATCH}(k)$ now become:

$$\mathbf{H}_{BATCH}(k) = \begin{bmatrix} \frac{\partial \tilde{y}(k+1)}{\partial w_1} & \cdots & \frac{\partial \tilde{y}(k+1)}{\partial w_{N_w}} \\ \cdots & \cdots & \cdots \\ \frac{\partial \tilde{y}(k+B)}{\partial w_1} & \cdots & \frac{\partial \tilde{y}(k+B)}{\partial w_{N_w}} \end{bmatrix}, \tag{42}$$

$$\mathbf{E}_{BATCH}(k) = \left[t(k+1) - \tilde{y}(k+1) \quad \ldots \quad t(k+B) - \tilde{y}(k+B) \right]. \tag{43}$$

Note that outputs $\tilde{y}(\cdot)$ for calculating the training derivatives and residuals for mini-batch EKF are pure SS predictions. If derivatives $\frac{\partial \tilde{y}(\cdot)}{\partial w}$ are calculated dynamically (e.g., BPTT) they provide better MS prediction quality.

The size of matrix $\mathbf{R}$ is $(O \times B) \times (O \times B)$, the size of matrix $\mathbf{H}_{BATCH}(k)$ is $(O \times B) \times N_w$, and the size of matrix $\mathbf{E}_{BATCH}(k)$ is $(O \times H) \times 1$. The remainder is identical to regular EKF. Again, here without loss of generality we assume $O = 1$. However, the mini-batch method requires at least B more calculations at each time step in comparison to the original EKF method.

6 Training FFNNs for MS Predictions Using FPTT and Mini-Batch EKF

Recurrent neural networks trained using Backpropagation Through Time show better multi-step-ahead prediction quality than feedforward neural networks trained using backpropagation. The underlying idea of BPTT is to calculate derivatives by propagating the errors back across the RNN, which is unfolded through time. This penalizes the network for accumulating errors in time and therefore provides better MS predictions. Nonetheless, RNNs have some disadvantages. First, the implementation of RNNs is harder than feedforward neural networks (FFNNs) in industrial settings. Second, training the RNNs is a difficult problem because of their more complicated error surfaces and vanishing gradient effects. Third, the internal dynamics of RNNs are less amenable to stability analysis. All of the above reasons prevent RNNs from becoming widely popular in industry. Meanwhile, RNNs have inspired a new family of methods for training FFNNs to perform MS predictions called direct methods [6]. Accumulated error is backpropagated through an unfolded through time FFNN in BPTT style that causes minimization of the MS prediction error. Nevertheless, the vanishing gradient effect still occurs in all multilayer perceptron-based networks with sigmoidal activation functions.

We propose a new, effective method for training the feedforward neural models to perform MS prediction, called Forecasted Propagation Through Time (FPTT), for calculating the batch-like dynamic derivatives that minimize the negative effect of vanishing gradients. We use the mini-batch modification of the EKF algorithm which naturally deals with these batch-like dynamic derivatives for training the neural network.

The scheme for calculating the dynamic derivatives for the FFNNs, called Forecasted Propagation Through Time, is depicted in Fig. 11.

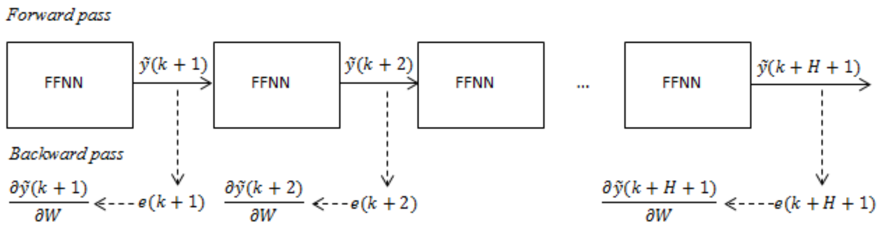

Forward pass
FFNN
$\tilde{y}(k+1)$
FFNN
$\tilde{y}(k+2)$
FFNN
...
FFNN
$\tilde{y}(k+H+1)$
Backward pass
$\frac{\partial \tilde{y}(k+1)}{\partial W}$
$e(k+1)$
$\frac{\partial \tilde{y}(k+2)}{\partial W}$
$e(k+2)$
$\frac{\partial \tilde{y}(k+H+1)}{\partial W}$
$e(k+H+1)$

Fig. 11 Calculation of dynamic FPTT derivatives for feedforward neural networks

1. At each time step the neural network is unfolded forward through time H times using Eqs. (5)-(8) in the same way as it is performed for regular multi-step-ahead prediction, where H is a horizon of prediction. Outputs $\tilde{y}(k+1),..., \tilde{y}(k+H+1)$ are calculated.

2. For each of the H forecasted time steps, prediction errors $e(k+h+1) = t(k+h+1) - \tilde{y}(k+h+1)$, $h = 1,...,H$ are calculated.
3. The set of independent derivatives $\left\{ \frac{\partial \tilde{y}(k+h)}{\partial w} \right\}$, $h = 1,...,H+1$, are calculated for each copy of the unfolded neural network using the standard backpropagation of independent errors $\{e(k+h)\}$.

The mini-batch observation matrix $\mathbf{H}_{FPTT}(k)$ and residual matrix $\mathbf{E}_{FPTT}(k)$ now become:

$$\mathbf{H}_{FPTT}(k) = \begin{bmatrix} \frac{\partial \tilde{y}(k+1)}{\partial w_1} & ... & \frac{\partial \tilde{y}(k+1)}{\partial w_{N_w}} \\ ... & ... & ... \\ \frac{\partial \tilde{y}(k+H+1)}{\partial w_1} & ... & \frac{\partial \tilde{y}(k+H+1)}{\partial w_{N_w}} \end{bmatrix}, \tag{44}$$

$$\mathbf{E}_{FPTT}(k) = [t(k+1) - \tilde{y}(k+1) \quad ... \quad t(k+B) - \tilde{y}(k+H+1)]. \tag{45}$$

Outputs $\tilde{y}(\cdot)$ for calculating the training derivatives and residuals here are the direct result of MS prediction at each time step. There are three main differences between the proposed FPTT and traditional BPTT. First, BPTT unfolds the neural network backward through time; FPTT unfolds the neural network recursively forward through time. This is useful from a technological point of view because this functionality must be implemented for MS predictions anyway. Second, FPTT does not backpropagate the accumulated error through the whole unfolded structure. Instead, it calculates BP for each copy of the neural network. Finally, FPTT does not average derivatives, it calculates a set of formally independent errors and a set of formally independent derivatives for future time steps. By doing this, we leave the question of contributions by each time step to the total MS error to the mini-batch EKF algorithm.

6.1 Multi-Step-Ahead on the Mackey-Glass Chaotic Process

In the first experiment on MS predictions we used the Mackey-Glass chaotic process (see details about data in Section 2.5). 500 values were used for training; the next 100 values were used for testing. First, we trained 100 DMLP networks with one hidden layer and hyperbolic tangent activation functions using traditional EKF and BP derivatives. The training parameters for EKF were set as $\eta = 10^{-3}$ and $\mu = 10^{-8}$. The number of neurons in the hidden layer was varied from 3 to 8, the order of input tapped delay line was set $N = 5$, and the initial weights were set to small random values. Each network was trained for 50 epochs. After each epoch, MS prediction on horizon $H = 14$ on the training data was performed to select the best network. This network was then evaluated on the test sequence to achieve the final MS quality result. Second, we trained 100 DMLP networks with the same initial weights using the proposed mini-batch EKF technique together with FPTT derivatives and evaluated their MS prediction accuracy. Third, we trained 100 NARX networks (orders of tapped delay lines: $N = 5, L = 5$) using EKF and BPTT derivatives to make comparisons. The results of these experiments are presented in Table 1. Normalized Mean Square Error (NMSE) was used for the quality estimations.

Table 2 Mackey-Glass dataset: mean NMSE errors for different prediction horizon values

	H=1	H=2	H=6	H=8	H=10	H=12	H=14
DMLP EKF BP	0.0006	0.0014	0.013	0.022	0.033	0.044	0.052
DMLP BEKF FPTT	0.0017	0.0022	0.012	0.018	0.022	0.027	0.030
NARX EKF	0.0010	0.0014	0.012	0.018	0.023	0.028	0.032

6.2 *Multi-Step-Ahead on Santa-Fe Laser Dataset*

In order to explore the capability of the global behavior of DMLP using the proposed training method, we tested it on the laser data from the Santa Fe competition. The dataset consisted of laser intensities collected from the real experiment. Data was divided to training (1000 values) and testing (100 values) subsequences. This time the goal for training was to perform long-term (H=100) MS prediction. The order of the time delay line was set to $N = 25$ as in [4], the rest was the same as in the previous experiment. The obtained average NMSE for 100 DMLP networks was 0.175 for DMLP EKF BP (classic method) versus 0.082 for DMLP BEKF FPTT (proposed method). NARX networks shows NMSE 0.131 in average.

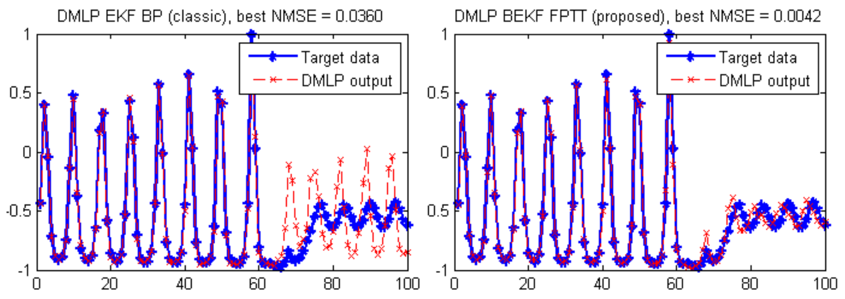


Fig. 12. The best results of the closed-loop long-term predictions (H=100) on testing data using DMLPs trained using different methods

Meanwhile, the best instance trained using mini-batch EKF+FPTT shows 10 times better accuracy than the best instance trained using the traditional approach.

7 Conclusions

We considered the multi-step-ahead prediction problem and discussed neural network based approaches as a tool for its solution. Feedforward and recurrent neural models were considered, and advantages and disadvantages of their usage were discussed. A novel direct method for training feedforward neural networks to perform multi-step-ahead predictions was proposed, based on the mini-batch Extended Kalman Filter. This method is considered to be useful from a technological point of view because it uses existing multi-step-ahead prediction functionality for calculating special FPTT dynamic derivatives that require a slight modification of the standard EKF algorithm. Our method demonstrates doubled long-term accuracy compared to standard training of the dynamic MLPs using the EKF due to direct minimization of the accumulated multi-step-ahead error.

References

1. Haykin, S.: Neural Networks and Learning Machines, 3rd edn., 936 p. Prentice Hall, New York (2009)
2. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn., p. 842. Prentice Hall, Englewood Cliffs (1999)
3. Bishop, C.M.: Pattern Recognition and Machine Learning, 738 p. Springer (2006)
4. Giles, L.L., Horne, B.G., Sun-Yan, Y.: A Delay Damage Model Selection Algorithm for NARX Neural Networks. IEEE Transactions on Signal Processing 45(11), 2719–2730 (1997)
5. Parlos, A.G., Raisa, O.T., Atiya, A.F.: Multi-step-ahead prediction using dynamic recurrent neural networks. Neural Networks 13(7), 765–786 (2000)
6. Bone, R., Cardot, H.: Advanced Methods for Time Series Prediction Using Recurrent Neural Networks. In: Recurrent Neural Networks for Temporal Data Processing, ch. 2, pp. 15–36. Intech, Croatia (2011)
7. Qina, S.J., Badgwellb, T.A.: A survey of industrial model predictive control technology. Control Engineering Practice 11(7), 733–764 (2003)
8. Toth, E., Brath, A.: Multistep ahead streamflow forecasting: Role of calibration data in conceptual and neural network modeling. Water Resources Research 43(11) (2007), doi:10.1029/2006WR005383
9. Prokhorov, D.V.: Toyota Prius HEV Neurocontrol and Diagnostics. Neural Networks (21), 458–465 (2008)
10. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Technical ReportGMDReport 148, German National Research Center for Information Technology (2001)
11. Anderson, J.A., Rosenfeld, E. (eds.): Talking nets: An oral history of neural networks, p. 54. MIT Press, Cambridge (1998)
12. Hava, T., Siegelmann, B.G., Horne, C.: Computational capabilities of recurrent NARX neural networks. IEEE Transactions on Systems, Man, and Cybernetics, Part B 27(2), 208–215 (1997)
13. Newton, H.J., North, G.R.: Forecasting global ice volume. J. Time Series Analysis 1991(12), 255–265 (1991)

14. Hyndman, R.J.: Time Series Data Library, `http://data.is/TSDLdemo`
15. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient scent is difficult. IEEE Trans. Neural Networks 5(2), 157–166 (1994)
16. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: A Field Guide to Dynamical Recurrent Neural Networks, 421 p. IEEE Press (2001)
17. Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., Kingsbury, B.: Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Processing Magazine 29(6), 82–97 (2012)
18. Singhal, S., Wu, L.: Training Multilayer Perceptrons with the Extended Kalman algorithm. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 1, pp. 133–140. Morgan Kaufmann, San Mateo (1989)
19. Arasaratnam, I., Haykin, S.: Nonlinear Bayesian Filters for Training Recurrent Neural Networks. In: Gelbukh, A., Morales, E.F. (eds.) MICAI 2008. LNCS (LNAI), vol. 5317, pp. 12–33. Springer, Heidelberg (2008)
20. Haykin, S.: Kalman Filtering and Neural Networks, 304 p. John Wiley & Sons (2001)
21. Puskorius, G.V., Feldkamp, L.A.: Decoupled Extended Kalman Filter Training of Feedforward Layered Networks. In: International Joint Conference on Neural Networks, Seattle, July 8-14, vol. 1, pp. 771–777 (1991)
22. Li, S.: Comparative Analysis of Backpropagation and Extended Kalman Filter in Pattern and Batch Forms for Training Neural Networks. In: Proceedings on International Joint Conference on Neural Networks (IJCNN 2001), Washington, DC, July 15-19, vol. 1, pp. 144–149 (2001)
23. Chernodub, A.: Direct Method for Training Feed-Forward Neural Networks Using Batch Extended Kalman Filter for Multi-Step-Ahead Predictions. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 138–145. Springer, Heidelberg (2013)
24. Mirikitani, D.T., Nikolaev, N.: Dynamic Modeling with Ensemble Kalman Filter Trained Recurrent Neural Networks. In: Seventh International Conference on Machine Learning and Applications (ICMLA 2008), San Diego, USA, December 11-13 (2008)
25. Wan, E.A., van der Merwe, R.: The Unscented Kalman Filter for Nonlinear Estimation. In: Proceedings of IEEE Symposium (AS-SPCC), Lake Louise, Alberta, Canada, pp. 153–158 (October 2000)
26. Arasaratnam, I., Haykin, S.: Cubature Kalman Filters. IEEE Transactions on Automatic Control 56(6), 1254–1269

Learning as Constraint Reactions[⋆]

Giorgio Gnecco, Marco Gori, Stefano Melacci, and Marcello Sanguineti

Abstract. A theory of learning is proposed, which extends naturally the classic regularization framework of kernel machines to the case in which the agent interacts with a richer environment, compactly described by the notion of constraint. Variational calculus is exploited to derive general representer theorems that give a description of the structure of the solution to the learning problem. It is shown that such solution can be represented in terms of *constraint reactions*, which remind the corresponding notion in analytic mechanics. In particular, the derived representer theorems clearly show the extension of the classic kernel expansion on support vectors to the expansion on *support constraints*. As an application of the proposed theory three examples are given, which illustrate the dimensional collapse to a finite-dimensional space of parameters. The constraint reactions are calculated for the classic collection of supervised examples, for the case of box constraints, and for the case of hard holonomic linear constraints mixed with supervised examples. Interestingly, this leads to representer theorems for which we can re-use the kernel machine mathematical and algorithmic apparatus.

Giorgio Gnecco
IMT, Piazza S. Ponziano 6, 55100 Lucca, Italy
e-mail: giorgio.gnecco@imtlucca.it

Marco Gori · Stefano Melacci
DIISM – University of Siena, Via Roma 56, 53100 Siena, Italy
e-mail: {marco,mela}@diism.unisi.it

Marcello Sanguineti
DIBRIS – University of Genoa, Via Opera Pia 13, 16145 Genova, Italy
e-mail: marcello.sanguineti@unige.it

⋆ Part of this chapter is an excerpt of the paper "Foundations on Support Constraint Machines," by the same authors, that will appear in Neural Computation, which contains a significantly broader view on the subject along with a more general framework, algorithmic issues, and references to applications.

1 Introduction

Examples of constraints in machine learning come out naturally in various situations: constraints may represent, for instance, prior knowledge provided by an expert (e.g., a physician in the case of a medical application: in such a case constraints may be expressed in the form of rules which help in the detection of a disease [14, 16]). The expressive power of constraints becomes particularly significant when dealing with a specific problem, like vision, control, text classification, ranking in hypertextual environment, and prediction of the stock market.

Table 1 provides some examples of constraints that are often encountered in practical problems arising in different domains. The first example (*i*) describes the simplest case in which we handle several classic pairs (x_κ, y_κ) provided for supervised learning in classification, where x_κ is the κ-th supervised example and $y_\kappa \in \{-1, 1\}$ is its label. If $f(\cdot)$ is the function that the artificial agent is expected to compute, then the corresponding real-valued representation of the constraint is just the translation of the classic "robust" sign agreement between the target and the function to be learned. Example *ii* is the normalization of a probability density function, whereas example *iii* (which refers to a binary classification problem) imposes the coherence between the decisions taken on S_1x and S_2x for the object x, where S_1 and S_2 are matrices used to select two different views of the same object (see [17]). In the example *iv* we report a constraint from computer vision coming from the classic problem of determining the optical flow. It consists of finding the smoothest solution for the velocity field under the constraint that the brightness of any point in the movement pattern is constant. If $u(x,y,t)$ and $v(x,y,t)$ denote the components of the velocity field and $E(x,y,t)$ the brightness of any pixel (x,y) at time t, then the velocity field satisfies the linear constraint indicated in Table 1 *iv*. Finally, example *v* in the table refers to a document classification problem, and states the rule that all papers dealing with numerical analysis and neural networks are machine-learning papers. Notice that, whereas the first row of example *v* expresses the rule by a first-order logic description, in the second row there is a related constraint expressed by real-valued functions that is constructed using the classic product T-norm [4, 5, 13].

The aim of this chapter is to show how the framework of kernel machines can be extended to support constraint machines by including prior knowledge modeled by several kinds of constraints. In particular, we propose a framework in which the ambient space is described in terms of Reproducing Kernel Hilbert Spaces (RKHSs) of Sobolev type, which has the advantage, over generic RKHSs, of providing optimality conditions expressed as partial differential equations (see Theorems 1 and 2 in Section 2). The general learning paradigm of support constraints machines, its mathematical foundations, representer theorems, and algorithmic issues is presented in [11].

Unlike the classic framework of learning from examples, the beauty and the elegance of the simplicity behind the parsimony principle - for which simple explanations are preferred to complex ones - has not been profitably used yet for the formulation of systematic theories of learning in general constrained environments, although there are some works on learning in specific constrained contexts [2, 6, 12, 15, 21–23]. We propose the study of parsimonious agents interacting

Table 1 Examples of constraints from different environments. For each entry, both a linguistic description of the constraint and its real-valued representation are provided.

i	κ-th supervised pair for classification
	$y_\kappa \cdot f(x_\kappa) - 1 \geq 0$
ii	normalization of a probability density function
	$\int_{\mathscr{X}} f(x)dx = 1$, and $\forall x \in \mathscr{X} : \ f(x) \geq 0$
iii	coherence constraint (two classes)
	$\forall x \in \mathscr{X} : \ f_1(S_1 x) \cdot f_2(S_2 x) > 0$
iv	brightness invariance - optical flow
	$\frac{\partial E}{\partial x} u + \frac{\partial E}{\partial y} v + \frac{\partial E}{\partial t} = 0$
v	document classification: $\forall x : \ na(x) \wedge nn(x) \Rightarrow ml(x)$
	$\forall x \in \mathscr{X} : \ f_{na}(x) f_{nn}(x)(1 - f_{ml}(x)) = 0$

simultaneously with examples and constraints in a multi-task environment with the purpose of developing the simplest (smoothest) vectorial function in a set of feasible solutions. More precisely, we think of an intelligent agent acting on a subset $\mathscr{X}$ of the perceptual space $\mathbb{R}^d$ as one implementing a vectorial function $f := [f_1, \ldots, f_n]' \in \mathscr{F}$, where $\mathscr{F}$ is a space of functions from $\mathscr{X}$ to $\mathbb{R}^n$. Each function f_j is referred to as a *task of the agent*. We assume that additional prior knowledge is available, defined by the fulfillment of constraints modeled as

$$\forall x \in \mathscr{X}_i \subseteq \mathscr{X} : \ \phi_i(x, f(x)) = 0, \ i = 1, \ldots, m, \tag{1}$$

or as

$$\forall x \in \mathscr{X}_i \subseteq \mathscr{X} : \ \check{\phi}_i(x, f(x)) \geq 0, \ i = 1, \ldots, m, \tag{2}$$

where $\phi_i, \check{\phi}_i$ are scalar-valued functions. Following the terminology in variational calculus, when the sets $\mathscr{X}_i$ are open we call (1) *bilateral holonomic constraints* and (2) *unilateral holonomic constraints*. When the sets $\mathscr{X}_i$ are made by finite numbers of points, we replace the term "holonomic" by *point-wise*. The constraints above are called *hard* if they cannot be violated; constraints that can be violated (at the cost of some penalization) play the role of *soft* constraints (this is usually the case for supervised pairs of the learning set). In this way, any cross-dependence amongst the functions f_j is expressed directly by the constraints.

In this chapter, which is an improved and extended version of [8], we investigate theoretically and by means of case studies the problem of learning in a constraint-based environment, taking into account both hard and soft constraints of holonomic and point-wise types. The focus on holonomic constraints is motivated by the fact that they model very general prior knowledge, expressed by universal quantifiers. Examples of learning problems with holonomic constraints are given, e.g, in [5], where the constraints arise by a suitable representation of prior knowledge expressed in terms of first-order-logic clauses. We also consider point-wise constraints, which arise, e.g., in the case of interpolation and approximation problems, given a finite set of examples. However, the proposed framework can be extended to several other kinds of constraints and their combinations (e.g., isoperimetric ones, box constraints [9,16], and boundary conditions [10]).

The chapter is organized as follows. In Section 2 we formalize the problems of learning from soft and hard constraints and we present the corresponding representer theorems, which provide information on the form of their solutions. Section 3 is devoted to the concepts of reactions of the constraints and support constraint machines. Section 4 analyzes some practical instances of the proposed framework. Section 5 is a discussion. Finally, the most technical results are detailed in three appendices.

2 Learning from Constraints and Its Representer Theorems

In this chapter, we assume $\mathscr{X}$ to be either the whole $\mathbb{R}^d$, or an open, bounded and connected subset of $\mathbb{R}^d$, with strongly local Lipschitz continuous boundary [1]. In particular, we consider the case in which, $\forall j \in \mathbb{N}_n := \{1,\ldots,n\}$ and some positive integer k, the function $f_j : \mathscr{X} \to \mathbb{R}$ belongs to the Sobolev space $\mathscr{W}^{k,2}(\mathscr{X})$, i.e., the subset of $\mathscr{L}^2(\mathscr{X})$ whose elements f_j have weak partial derivatives up to the order k with finite $\mathscr{L}^2(\mathscr{X})$-norms. So, the ambient space of the class of proposed problems of learning from constraints is

$$\mathscr{F} := \underbrace{\mathscr{W}^{k,2}(\mathscr{X}) \times \ldots \times \mathscr{W}^{k,2}(\mathscr{X})}_{n\,\text{times}} .$$

We take $k > \frac{d}{2}$ since, by the Sobolev Embedding Theorem (see, e.g., Chapter 4 in [1]), for $k > \frac{d}{2}$ each element of $\mathscr{W}^{k,2}(\mathscr{X})$ has a continuous representative, and under such an assumption $\mathscr{F}$ is a RKHS.

We can introduce a seminorm $\| f \|_{P,\gamma}$ on $\mathscr{F}$ via the pair (P,γ), where $P := [P_0,\ldots,P_{l-1}]'$ is a suitable (vectorial) finite-order[1] differential operator of order k with l components and $\gamma \in \mathbb{R}^n$ is a fixed vector with positive components. Let us consider the functional

$$\begin{aligned} \mathscr{E}(f) := \| f \|^2_{P,\gamma} &= \sum_{j=1}^{n} \gamma_j < Pf_j, Pf_j > \\ &= \sum_{j=1}^{n} \gamma_j \left(\sum_{r=0}^{l-1} \int_{\mathscr{X}} (P_r f_j(x) P_r f_j(x)) dx \right) . \end{aligned} \tag{3}$$

Note that we overload the notation and use the symbol P for both the (matrix) differential operator acting on f and the (vector) one acting on its components. If we choose for P the form used in Tikhonov's stabilizing functionals [24], for $n = 1$ and $l = k+1$ we get

$$\| f \|^2_{P,\gamma} = \gamma \int_{\mathscr{X}} \sum_{r=0}^{k} \rho_r(x) \left(D_r f(x)\right)^2 dx ,$$

[1] The results can be extended to infinite-order differential operators (see the example in Section 4.3).

where the function $\rho_r(x)$ is nonnegative, $P_r := \sqrt{\rho_r(x)}D_r$, and D_r denotes a differential operator with constant coefficients containing only partial derivatives of order r. In this work, we focus on the case in which the operator P is invariant under spatial shift and has constant coefficients. For a function u and a multiindex α with d nonnegative components α_j, we write $D^\alpha u$ to denote $\frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}} u$, where $|\alpha| := \sum_{j=1}^d \alpha_j$. So, the generic component P_i of P has the expression $P_i = \sum_{|\alpha| \leq k} b_{i,\alpha} D^\alpha$, where the $b_{i,\alpha}$'s are suitable real coefficients. Then, the formal adjoint of P is defined as the operator $P^\star = [P_0^\star, \ldots, P_{l-1}^\star]'$ whose i-th component $P_i^\star$ is given by $P_i^\star = \sum_{|\alpha| \leq k} (-1)^{|\alpha|} b_{i,\alpha} D^\alpha$. Finally, we define the operators $L := (P^\star)'P$ and, using again an overloaded notation, $\gamma L := [\gamma_1 L, \ldots, \gamma_n L]'$.

2.1 Soft Constraints

We start considering the case of learning from soft constraints, whose representer theorem has a simpler formulation than in the case of hard ones. The problem of learning from soft constraints is based on a direct soft re-formulation of (3). We can associate any holonomic or point-wise unilateral constraint $\check{\phi}_i(x, f(x)) \geq 0$ with $\phi_i^{\geq}(x, f(x)) = 0$, where $\phi_i^{\geq}(\cdot, \cdot)$ is a non-negative function. Similarly, each holonomic or point-wise bilateral constraint can be expressed via a pair of unilateral constraints. Hence, regardless of bilateral or unilateral constraints, the problem of learning from soft holonomic or point-wise constraints can be formulated as the minimization of the functional

$$\mathscr{L}_s(f) := \frac{1}{2}\mathscr{E}(f) + \sum_{i=1}^{m} \int_{\mathscr{X}} p(x)\, 1_{\mathscr{X}_i}(x) \phi_i^{\geq}(x, f(x)) dx, \qquad (4)$$

where $p(x)$ is a (nonnegative) weight function, e.g., a probability density function (this setting can be extended to the case of a generalized probability density function). We use $1_{\mathscr{X}_i}(\cdot)$ to denote the characteristic function of the set $\mathscr{X}_i$ when $\mathscr{X}_i$ is open. In order to keep the notation uniform, we let $1_{\mathscr{X}_i}(\cdot) := \delta(\cdot - x_i)$, where δ denotes the Dirac delta, for a set $\mathscr{X}_i = \{x_i\}$ made up of a single element. Finally, for two vector-valued functions $u^{(1)}$ and $u^{(2)}$ of the same dimensions, $u^{(1)} \otimes u^{(2)}$ represents the vector-valued function v whose first component is the convolution of the first components of $u^{(1)}$ and $u^{(2)}$, the second component is the convolution of the second components of $u^{(1)}$ and $u^{(2)}$, and so on, i.e., $v_i := (u^{(1)} \otimes u^{(2)})_i := u_i^{(1)} \otimes u_i^{(2)}$ for each index i.

Theorem 1. (REPRESENTER THEOREM FOR SOFT HOLONOMIC AND SOFT POINT-WISE CONSTRAINTS). *Let $p(\cdot)$ be continuous, nonnegative and in $\mathscr{L}^1(\mathscr{X})$, and let f^o be a local minimizer of the functional (4) over $\mathscr{F}$.*

(i) Let also the following hold: $\forall i \in \mathbb{N}_m$, $\mathscr{X}_i \subseteq \mathscr{X}$ is open and $\forall x \in \mathscr{X}_i$, there is an open neighborhood $\mathscr{N}$ of $(x, f^o(x))$ for which $\phi_i^{\geq} \in \mathscr{C}^1(\mathscr{N})$. Then, f^o satisfies on $\mathscr{X}$

$$\gamma L f^o(x) + \sum_{i=1}^{m} p(x) 1_{\mathscr{X}_i}(x) \cdot \nabla_f \phi_i^{\geq}(x, f^o(x)) = 0, \tag{5}$$

where $\gamma L := [\gamma_1 L, \ldots, \gamma_n L]'$ is a spatially-invariant operator[2]*, and $\nabla_f \phi_i^{\geq}$ is the gradient w.r.t. the second vector argument f of the function $\phi_i^{\geq}$.*

(ii) Suppose now that the sets $\mathscr{X}_i$ are disjoint and each set $\mathscr{X}_i$ is made up of a single point x_i, and that $\phi_i^{\geq}$ has the form

$$\phi_i^{\geq}(x, f(x)) = \sum_{j \in \mathbb{N}_n} \phi_{i,j}^{\geq}(x, f_j(x)),$$

where $\phi_{i,j}^{\geq}(x, f_j(x)) := (1 - y_{i,j} \cdot f_j(x))_+$, and the $y_{i,j}$'s belong to the set $\{-1, 1\}$. Then, f^o satisfies on $\mathscr{X}$ (for $j = 1, \ldots, n$)

$$\gamma_j L f_j^o(x) + \sum_{i=1}^{m} p(x) 1_{\mathscr{X}_i}(x) \cdot \overline{\partial_{f_j}} \phi_{i,j}^{\geq}(x, f_j^o(x)) = 0, \tag{6}$$

where $\overline{\partial_{f_j}} \phi_{i,j}^{\geq}(x, f_j^o(x))$ is a suitable element of the subdifferential[3] *$\partial_{f_j} \phi_{i,j}^{\geq}(x, f_j^o(x))$.*

(iii) Let the assumptions of either item (i) or item (ii) hold. If, moreover, $\mathscr{X} = \mathbb{R}^d$, L is invertible on $\mathscr{W}^{k,2}(\mathscr{X})$, and there exists a free-space Green's function g of L that belongs to $\mathscr{W}^{k,2}(\mathscr{X})$, then f^o can be represented as

$$f^o(\cdot) = \sum_{i=1}^{m} \gamma^{-1} g(\cdot) \vec{\otimes} \phi_i^{\geq}(\cdot, f^o(\cdot)), \tag{7}$$

where $g \vec{\otimes} \phi_i^{\geq} := g \otimes \omega_i^{\geq}$ and

$$\omega_i^{\geq} := \uparrow \phi_i^{\geq}(\cdot, f^o(\cdot)) := -p(\cdot) 1_{\mathscr{X}_i}(\cdot) \nabla_f \phi_i^{\geq}(\cdot, f^o(\cdot))$$

under the assumptions of (i), while the n components of $\omega_i^{\geq}$ are defined as

$$\omega_{i,j}^{\geq} := \uparrow \phi_i^{\geq}(\cdot, f^o(\cdot)) := -p(\cdot) 1_{\mathscr{X}_i}(\cdot) \overline{\partial_{f_j}} \phi_{i,j}^{\geq}(\cdot, f^o(\cdot))$$

under the assumptions of (ii).

Proof. (i) is proved by fixing arbitrarily $\eta \in \mathscr{C}_0^k(\mathscr{X}, \mathbb{R}^n)$ (the set of functions from $\mathscr{X}$ to $\mathbb{R}^n$ that are continuously differentiable up to order k, and have compact support), then computing

[2] Here we use again an overloaded notation, as made for the operator P.

[3] Let $\Omega \subseteq \mathbb{R}^d$ be a convex set. We recall that the subdifferential of a convex function $u : \Omega \to \mathbb{R}$ at a point $x_0 \in \Omega$ is the set of all the subgradients of u at x_0, that is the set of all vectors $v \in \mathbb{R}^d$ such that $f(x) - f(x_0) \geq v'(x - x_0)$.

$$0 = \lim_{\varepsilon \to 0} \frac{\mathscr{L}_s(f^o + \varepsilon\eta) - \mathscr{L}_s(f^o)}{\varepsilon}$$
$$= \int_{\mathscr{X}} \left(\gamma L f^o(x) + \sum_{i=1}^{m} p(x) 1_{\mathscr{X}_i}(x) \cdot \nabla_f \phi_i^{\geq}(x, f^o(x)) \right)' \eta(x) dx \,.$$

The first equality has been derived by the local optimality of f^o, whereas the second one has been derived by exploiting the assumption that $\forall x \in \mathscr{X}_i$ there is an open neighborhood $\mathscr{N}$ of $(x, f^o(x))$ for which $\phi_i^{\geq} \in \mathscr{C}^1(\mathscr{N})$. Finally, the proof is completed applying the fundamental lemma of the calculus of variations (for which we refer, e.g., to Section 2.2 in [7]).

(ii) Let us fix arbitrarily $\eta \in \mathscr{C}_0^k(\mathscr{X}, \mathbb{R}^n)$, with the additional condition that $\eta(x) = 0$ for all $x \in \cup_{i=1}^m \mathscr{X}_i$. Proceeding likewise in the proof of item (i), one obtains

$$\lim_{\varepsilon \to 0} \frac{\mathscr{L}_s(f^o + \varepsilon\eta) - \mathscr{L}_s(f^o)}{\varepsilon} = \int_{\mathscr{X}} (\gamma L f^o(x))' \eta(x) dx = 0 \,. \tag{8}$$

Since, for each index $j = 1, \ldots, n$, $\gamma_j L f_j^o$ is a distribution, formula (8) implies that the support of $\gamma_j L f_j^o$ is a subset of $\{x_1, \ldots, x_m\}$, which is a set of finite cardinality. By Theorem XXXV in Chapter 3 of [19], $\gamma_j L f_j^o$ is made up of a finite linear combination of Dirac delta's and their partial derivatives up to some finite order, centered on $x_1, \ldots, x_m$. Now, all the coefficients associated with the partial derivatives of any order of the Dirac delta's are 0, as it can be checked by choosing a function $\eta \in \mathscr{C}_0^\infty(\mathscr{X}, \mathbb{R}^n)$ such that only its j-th component η_j is different from 0, and $\eta_j(x) = 0$ for all $x \in \cup_{i=1}^m \mathscr{X}_i$ (even though some partial derivatives of some order of η_j may be different from 0 for some $x \in \cup_{i=1}^m \mathscr{X}_i$). Concluding, $\gamma_j L f_j^o$ satisfies on $\mathscr{X}$

$$\gamma_j L f_j^o(x) = \sum_{i=1}^{m} B_i \delta(x - x_i) \,, \tag{9}$$

where the B_i's are constants. Notice that (9) is of the same form as (6).

Now, we look for lower and upper bounds on the B_i's. For simplicity of exposition, in the following we suppose $m = 1$, so there is only one constant B_1 (however, the next arguments hold also for the case $m > 1$). We denote by η^{j+} any function in $\mathscr{C}_0^k(\mathscr{X}, \mathbb{R}^n)$ such that only its j-th component η_j^{j+} is different from 0, and $\eta_j^{j+}(x_1) > 0$. Once η^{j+} has been fixed, we denote by η^{j-} the function $-\eta^{j+}$. The following possible cases show up.

Case (a): $(1 - y_{1,j} \cdot f_j^o(x_1))_+ < 0$. In this case, one obtains

$$\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon\eta^{j+}) - \mathscr{L}_s(f^o)}{\varepsilon}$$
$$= \int_{\mathscr{X}} \gamma_j L f_j^o(x) \eta_j^{j+}(x) dx = B_1 \eta_j^{j+}(x_1) \geq 0 \,, \tag{10}$$

and

$$\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j-}) - \mathscr{L}_s(f^o)}{\varepsilon} = \int_{\mathscr{X}} \gamma L_j f_j^o(x) \eta_j^{j-}(x) dx = -B_1 \eta_j^{j+}(x_1) \geq 0, \tag{11}$$

then $B_1 = 0$ (since $\eta_j^{j+}(x_1) > 0$). Notice that the "$\geq$" in formulas (10) and (11) follow by the local optimality of f^o, whereas the first equalities by the left/right differentiability[4] of the function $(\cdot)_+$.

Case (b): $(1 - y_{1,j} \cdot f_j^o(x_1))_+ > 0$. Similarly, in this case, one obtains

$$\begin{aligned}\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j+}) - \mathscr{L}_s(f^o)}{\varepsilon} &= \int_{\mathscr{X}} \left(\gamma_j L f_j^o(x) - y_{1,j} p(x) 1_{\mathscr{X}_i}(x)\right) \eta_j^{j+}(x) dx \\ &= (B_1 - y_{1,j} p(x_1)) \eta_j^{j+}(x_1) \geq 0,\end{aligned} \tag{12}$$

and

$$\begin{aligned}\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j-}) - \mathscr{L}_s(f^o)}{\varepsilon} &= \int_{\mathscr{X}} \left(\gamma_j L f_j^o(x) - y_{1,j} p(x) 1_{\mathscr{X}_i}(x)\right) \eta_j^{j-}(x) dx \\ &= -(B_1 - y_{1,j} p(x_1)) \eta_j^{j+}(x_1) \geq 0,\end{aligned} \tag{13}$$

then $B_1 = y_{1,j} p(x_1)$.

Case (c): $(1 - y_{1,j} \cdot f_j^o(x_1))_+ = 0$ and $y_{1,j} = -1$. In this case, one obtains

$$\begin{aligned}\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j+}) - \mathscr{L}_s(f^o)}{\varepsilon} &= \int_{\mathscr{X}} \left(\gamma_j L f_j^o(x) - y_{1,j} p(x) 1_{\mathscr{X}_i}(x)\right) \eta_j^{j+}(x) dx \\ &= (B_1 - y_{1,j} p(x_1)) \eta_j^{j+}(x_1) \geq 0,\end{aligned} \tag{14}$$

and

$$\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j-}) - \mathscr{L}_s(f^o)}{\varepsilon} = \int_{\mathscr{X}} \gamma L_j f_j^o(x) \eta_j^{j-}(x) dx = -B_1 \eta_j^{j+}(x_1) \geq 0, \tag{15}$$

then $B_1 \in [y_{1,j} p(x_1), 0] = [-p(x_1), 0]$.

[4] Depending on the sign of $y_{i,j}$.

Case (d): $(1 - y_{1,j} \cdot f_j^o(x_1))_+ = 0$ and $y_{1,j} = 1$. Finally, in this case, one obtains

$$\lim_{\varepsilon \to 0^+} \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j+}) - \mathscr{L}_s(f^o)}{\varepsilon} = \int_{\mathscr{X}} \gamma_j L f_j^o(x) \eta_j^{j+}(x) dx = B_1 \eta_j^{j+}(x_1) \geq 0 , \tag{16}$$

and

$$\begin{aligned} \lim_{\varepsilon \to 0^+} & \frac{\mathscr{L}_s(f^o + \varepsilon \eta^{j-}) - \mathscr{L}_s(f^o)}{\varepsilon} \\ &= \int_{\mathscr{X}} \left(\gamma_j L f_j^o(x) - y_{1,j} p(x) 1_{\mathscr{X}_i}(x) \right) \eta_j^{j-}(x) dx \\ &= -(B_1 - y_{1,j} p(x_1)) \eta_j^{j+}(x_1) \geq 0 , \end{aligned} \tag{17}$$

then $B_1 \in [0, y_{1,j} p(x_1)] = [0, p(x_1)]$.

Concluding, one obtains (6) summarizing the results of the analysis of cases (a)-(d), and applying the definition of subdifferentiability to the function $(\cdot)_+$.

(iii) follows by the Euler-Lagrange equations (5) (resp., (6)) of item (i) (resp., (ii)), the definition of the free-space Green's function g of L as the solution of $Lg = \delta$ (where δ denotes the Dirac delta, centered in 0), and the stated assumptions on L and g. □

Item (i) of Theorem 1 applies, e.g., to the case of a function $\phi_i^{\geq}$ that is continuously differentiable everywhere (or at least a function $\phi_i^{\geq}$ that is "seen as" a continuously differentiable function at local optimality, in the sense that $(x, f^o(x))$ is not a point of discontinuity of any partial derivative of $\phi_i^{\geq}$). However, a function $\phi_i^{\geq}$ deriving from a unilateral constraint may not be continuously differentiable everywhere. In such a case, one may approximate such a function by a continuously differentiable approximation, or (for certain $\phi_i^{\geq}$'s) deal directly with the nondifferentiable case, as shown in Theorem 1 (ii) for a particular choice of such functions. We remark that the classic supervised learning is a degenerate case of Theorem 1 (i), in which one sets $p(x)\, 1_{\mathscr{X}_i}(x) = p(x)\delta(x - x_i)$. Such a degenerate case is considered in Theorem 1 (ii) for the case of a particular nondifferentiable function $\phi_i^{\geq}$, but such a result can also be extended to other differentiable or nondifferentiable functions $\phi_i^{\geq}$. Finally, in Theorem 1 (iii) one can recognize both the ingredients of a parsimonious knowledge-based solution, i.e., the free-space Green's function g and the functions $\omega_i^{\geq}$, mixed by convolution. Indeed, note that, by defining $\omega^{\geq} := \sum_{i=1}^m \omega_i^{\geq}$, formula (7) can be re-written as $f^o = \gamma^{-1} g \otimes \omega^{\geq}$ and that, under the assumptions of Theorem 1 (iii), it follows by such an expression that the Fourier transform $\hat{f}^o$ of f^o (by Fourier transform of a vector-valued function we mean the vector of Fourier transforms of each component) is $\hat{f}^o = \gamma^{-1} \hat{g} \cdot \hat{\omega}^{\geq}$. Since under the assumptions of Theorem 1 (iii) the operator L is invertible and has $g\otimes$ as its inverse, from $f^o = \gamma^{-1} g \otimes \omega^{\geq}$ we get also $\omega^{\geq} = \gamma L f^o$, which is just a compact expression of the solution (7) to the Euler-Lagrange equations (5) or (6).

2.2 Hard Constraints

We now consider hard holonomic constraints. The following theorem prescribes the representation of the solution of the associated learning problem. Given a set of m holonomic constraints (defined, in general, on possibly different open subsets $\mathscr{X}_i$), we denote by $m(x)$ the number of constraints that are actually defined in the same point x of the domain. We denote by $\hat{\mathscr{X}}$ any open subset of $\mathscr{X}$, where the same subset of constraints is defined in all its points, in such a way that $m(x)$ is constant on the same $\hat{\mathscr{X}}$. By "cl" we denote the closure in the Euclidean topology. Finally, recall that a constraint $\check{\phi}_i(x, f(x)) \geq 0$ is said to be *active* in $x_0 \in \hat{\mathscr{X}}$ at local optimality iff $\check{\phi}_i(x_0, f^o(x_0)) = 0$, otherwise it is *inactive* in x_0 at local optimality.

Theorem 2. (REPRESENTER THEOREM FOR HARD HOLONOMIC CONSTRAINTS, CASE OF FUNCTIONAL LAGRANGE MULTIPLIERS). *Let us consider the minimization of the functional (3) in the case of $m < n$ hard bilateral constraints of holonomic type, which define the subset*

$$\mathscr{F}_\phi := \{f \in \mathscr{F} : \forall i \in \mathbb{N}_m, \forall x \in \mathscr{X}_i \subseteq \mathscr{X} : \phi_i(x, f(x)) = 0\}$$

of the function space $\mathscr{F}$, where $\forall i \in \mathbb{N}_m : \phi_i \in \mathscr{C}^{k+1}(\mathrm{cl}(\mathscr{X}_i) \times \mathbb{R}^m)$. Let f^o be any constrained local minimizer of class $\mathscr{C}^{2k}(\mathscr{X}, \mathbb{R}^n)$ of the functional (3). Let us assume that for any $\hat{\mathscr{X}}$ and for every x_0 in the same $\hat{\mathscr{X}}$ we can find two permutations σ_f and σ_ϕ of the indexes of the n functions f_j and of the m constraints ϕ_i, such that $\phi_{\sigma_\phi(1)}, \ldots, \phi_{\sigma_\phi(m(x_0))}$ refer to the constraints actually defined in x_0, and the Jacobian matrix

$$\frac{\partial(\phi_{\sigma_\phi(1)}, \ldots, \phi_{\sigma_\phi(m(x_0))})}{\partial(f^o_{\sigma_f(1)}, \ldots, f^o_{\sigma_f(m(x_0))})}, \tag{18}$$

evaluated in x_0, is not singular. Then, the following hold.

(i) There exists a set of functions $\lambda_i : \hat{\mathscr{X}} \to \mathbb{R}$, $i \in \mathbb{N}_m$, such that, in addition to the above constraints, f^o satisfies on $\hat{\mathscr{X}}$ the Euler-Lagrange equations

$$\gamma L f^o(x) + \sum_{i=1}^{m} \lambda_i(x) 1_{\mathscr{X}_i}(x) \cdot \nabla_f \phi_i(x, f^o(x)) = 0, \tag{19}$$

where $\gamma L := [\gamma_1 L, \ldots, \gamma_n L]'$ is a spatial-invariant operator, and $\nabla_f \phi_i$ is the gradient w.r.t. the second vector argument f of the function ϕ_i.

(ii) Let $\gamma^{-1} g := [\gamma_1^{-1} g, \ldots, \gamma_n^{-1} g]'$. If for all i one has $\mathscr{X}_i = \mathscr{X} = \mathbb{R}^d$, L is invertible on $\mathscr{W}^{k,2}(\mathscr{X})$, and there exists a free-space Green's function g of L that belongs to $\mathscr{W}^{k,2}(\mathscr{X})$, then f^o has the representation

$$f^o(\cdot) = \sum_{i=1}^{m} \gamma^{-1} g(\cdot) \vec{\otimes} \phi_i(\cdot, f^o(\cdot)), \tag{20}$$

where $g \vec{\otimes} \phi_i := g \otimes \omega_i$ and $\omega_i(\cdot) :=\uparrow \phi_i(\cdot, f^o(\cdot)) := -\lambda_i(\cdot) 1_{\mathscr{X}_i}(\cdot) \nabla_f \phi_i(\cdot, f^o(\cdot))$.

(iii) For the case of $m < n$ unilateral constraints of holonomic type, which define the subset $\mathscr{F}_{\check{\phi}} := \left\{ f \in \mathscr{F} : \forall i \in \mathbb{N}_m, \forall x \in \mathscr{X}_i \subseteq \mathscr{X} : \check{\phi}_i(x, f(x)) \geq 0 \right\}$ of the function space $\mathscr{F}$, (i) and (ii) still hold (with every occurrence of ϕ_i replaced by $\check{\phi}_i$) if one requires the nonsingularity of the Jacobian matrix (see (18)) to hold when restricting the constraints defined in x_0 to the ones that are active in x_0 at local optimality. Moreover, each Lagrange multiplier $\lambda_i(x)$ is nonpositive and equal to 0 *when the correspondent constraint is inactive in x at local optimality.*

Proof. The proof adapts to the case of hard holonomic constraints the one of Theorem 1 above. For completeness, it is detailed in Appendix 2. □

Notice that, due to the definition of ω_i, without loss of generality, one can define $\lambda_i(x) := 0$ for all $x \in \mathscr{X} \setminus \mathscr{X}_i$. Likewise in Theorem 1, by defining $\omega := \sum_{i=1}^{m} \omega_i$, formula (20) can be re-written as $f^o = \gamma^{-1} g \otimes \omega$, and, under the assumptions of Theorem 2 (ii),(iii), one can write $\hat{f}^o = \gamma^{-1} \hat{g} \cdot \hat{\omega}$. We also mention that a similar result can be proven for the case of hard point-wise constraints (in which one has discrete sets $\mathscr{X}_i$ composed of the $|\mathscr{X}_i|$ elements $x_{(i,1)}, x_{(i,2)}, \ldots, x_{(i,|\mathscr{X}_i|)}$), and for combinations of soft and hard constraints (e.g., soft point-wise constraints on supervised examples mixed with hard holonomic constraints, in which case the Lagrange multipliers are distributions instead of functions, as detailed in Appendix 2).

3 Support Constraint Machines

The next definition formalizes a concept that plays a basic role in the proposed learning paradigm.

Definition 1. The function $\omega_i^{\geq}$ in Theorem 1 (iii) (resp., the function ω_i in Theorem 2 (ii)) is called the *reaction of the i-th constraint* and $\omega^{\geq} := \sum_{i=1}^{m} \omega_i^{\geq}$ (resp. $\omega := \sum_{i=1}^{m} \omega_i$) is the overall reaction of the given constraints.

We emphasize the fact that the reaction of a constraint is a concept associated with the constrained local minimizer f^o. In particular, two different constrained local minimizers may be associated with different constraint reactions. A similar remark holds for the overall reaction of the constraints. Loosely speaking, under the assumptions of Theorem 1 (iii) or Theorem 2 (ii),(iii), the reaction of the i-th constraint provides the way under which such a constraint contributes to the expansion of f^o. So, in this case solving the learning problem is reduced to finding the reactions of the constraints.

Proposition 1. *Under the assumptions of the respective representer theorems (Theorem 1 (iii) and Theorem 2 (ii),(iii)), the reactions of the constraints are uniquely determined by the constrained local minimizer f^o.*

Proof. For the case of soft constraints (Theorem 1 (iii)), the statement follows directly by the definition of the reactions of the constraints $\omega_i^{\geq}(\cdot)$. For the case of hard constraints (Theorem 2 (ii),(iii)), the proof can be given by contradiction. Let us assume that there exist two different sets of Lagrange multipliers associated with the same constrained local minimizer f^o: $\{\lambda_i,\ i=1\dots,m\}$ and $\left\{\overline{\lambda}_i,\ i=1\dots,m\right\}$, with at least one $\lambda_i \neq \overline{\lambda}_i$. According to Theorem 2 (i), f^o satisfies the Euler-Lagrange equations (19). Without loss of generality, for each $x \in \hat{\mathscr{X}}$, one can re-order the constraints and the associated Lagrange multipliers in such a way that the first $m(x)$ constraints are the ones actually defined in $x \in \hat{\mathscr{X}}$, and assume that $\lambda_i(x) = \overline{\lambda}_i(x) = 0$ for all indexes $i > m(x)$, as the corresponding constraint reactions are equal to 0 in x due to the definition of ω_i. Subtracting the two expressions of f^o in terms of the two sets of Lagrange multipliers, one obtains $\sum_{i=1}^m (\lambda_i - \overline{\lambda}_i)\nabla_f \phi_i = (\lambda_{(D)} - \overline{\lambda}_{(D)})' \frac{\partial(\phi_1,\dots,\phi_{m(x)})}{\partial(f_1,\dots,f_{m(x)})} = 0$, where $\lambda_{(D)} := [\lambda_1,\dots,\lambda_m(x)]'$ and $\overline{\lambda}_{(D)} := [\overline{\lambda}_1,\dots,\overline{\lambda}_{m(x)}]'$. Now, distinct multipliers are only compatible with the singularity of the Jacobian matrix, which contradicts the assumption on the invertibility of (18). $\square$

A remarkable difference between the case of soft and hard constraints is the following. For soft constraints, the solution provided by Theorem 1 (iii) is based on the assumption of knowing the probability density of the data $p(\cdot)$. For hard constraints, instead (see Theorem 2 (ii),(iii)), one needs to compute the Lagrange multipliers $\lambda_i(\cdot)$ associated with the constraints, and also to check the (hard) satisfaction of the constraints.

Summing up, and removing the superscript "$\geq$" in $\phi_i^{\geq}$ and $\omega_i^{\geq}$ when the meaning is clear from the context, the solution of the learning problem is fully representable by the reactions ω_i of the constraints ϕ_i, as it is depicted in Fig. 1, where ∇_f denotes the gradient with respect to f, $\lambda_i(x)$ is the Lagrange multiplier, and $p(x)$ is the probability density. Interestingly, in the two cases, each constraint reaction has exactly the same dependency on the gradient of the constraint with respect to its second vector-valued argument but, while in the case of hard constraints the Lagrange multipliers need to be determined so as to impose the hard fulfillment of the constraints, in the case of soft constraints one exploits the probability density of

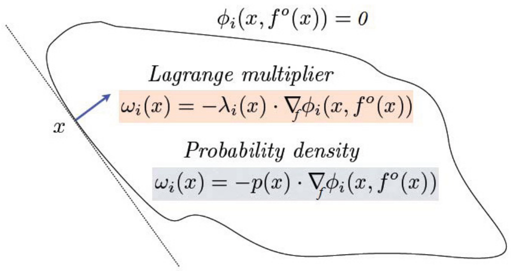

$\phi_i(x, f^o(x)) = 0$
Lagrange multiplier
$\omega_i(x) = -\lambda_i(x) \cdot \nabla_f \phi_i(x, f^o(x))$
x
Probability density
$\omega_i(x) = -p(x) \cdot \nabla_f \phi_i(x, f^o(x))$

Fig. 1 Constraint reactions in the generic point $x \in \mathscr{X}$ corresponding to the cases of hard and soft constraints, where one can see, resp., the roles of the Lagrange multiplier associated with each constraint and of the probability density in x. For illustrative purposes, the case $n = 2$ is considered here. The reaction of the constraint ϕ_i in x is a vector orthogonal to the level lines of $\phi_i(x, f^o(x))$, interpreted as a function of its second vector-valued argument only.

the data - which comes from the problem formulation - in the representation of the constraint reactions. Both $\lambda_i(x)$ and $p(x)$ play a crucial role in determining the constraint reactions. For a given point x, the weights $\lambda_i(x)$ need to be computed in such a way not to violate the constraints at x, whereas in case of soft-fulfillment, $p(x)$ - which is the typical weight that is high (low) in regions of high (low) data density - is used to compute the constraint reactions. Now, we introduce the following two concepts.

Definition 2. A *support constraint* is a constraint associated with a reaction that is different from 0 at least in one point of the domain $\mathscr{X}$. A *support constraint machine* is any learning machine capable of finding a (local or global) solution to either of the problems of learning from constraints formulated in Theorems 1 or 2, when such a solution is expressed by either the representation (7) or the one (20).

So, under the assumptions of Theorem 1 (iii) or Theorem 2 (ii),(iii), the solution f^o can be obtained by the knowledge of the reactions associated merely with the support constraints. This motivates the use of the terminology "support constraints" as an extension of the classical concept of "support vectors" used in kernel methods [20]. Interestingly, support vectors are particular cases of support constraints. Indeed, the connection with kernel methods arises because, under quite general conditions, the free-space Green's function g associated with the operator L is a kernel of a RKHS (see, e.g., [10]). Finally, we mention that for convex problems of learning from constraints, convex optimization algorithms can be used to find the reactions of the constraints (hence, to determine the support constraints).

4 Case Studies

4.1 Supervised Learning from Examples

The classic formulation is based on soft constraints and consists in finding $f^\star \in \operatorname{argmin}_{f\in\mathscr{F}} \mathscr{L}_s(f)$, where $\mathscr{L}_s(f)$ is given in formula (4) with $p(x)1_{\mathscr{X}_i}(x) = p(x)\delta(x-x_i)$ and each set $\mathscr{X}_i$ is a singleton. By $y_{i,j}$ we denote a real number for regression and an element of the set $\{-1,1\}$ for classification. As before, we denote by f^o a local minimizer (of which a global one $f^\star$ is a particular case). There are different possible choices for $\phi_i^{\geq}(x, f(x))$, which typically depend mostly on whether one faces regression or classification problems. The *quadratic loss* $V_Q(u) := \frac{1}{2}u^2$ is associated with the bilateral constraints $\phi_{i,j}(f_j(x)) = (y_{i,j} - f_j(x))$, which originate[5] - in the case of soft constraints - the term $\phi_i^{\geq}(f(x)) = \sum_{j\in\mathbb{N}_n} V_Q \circ \phi_{i,j}(f_j(x)) = \frac{1}{2}\sum_{j\in\mathbb{N}_n}(y_{i,j} - f_j(x))^2$. For every $j \in \mathbb{N}_n$ and $x \in \mathscr{X}$, by Theorem 1 the j-th component of the reaction of the i-th constraint is

[5] In the following, we write $\phi_i^{\geq}(f(x))$ instead of $\phi_i^{\geq}(x, f(x))$ since there is no explicit dependence on x.

$$\begin{aligned}\omega_{i,j}^{\overset{\geq}{=}}(x) &= (\uparrow \phi_i^{\overset{\geq}{=}}(f^o(x)))_j = -p(x)1_{\mathscr{X}_i}(x)\frac{\partial}{\partial f_j}\phi_i^{\overset{\geq}{=}}(f^o(x))\\ &= -p(x)\delta(x-x_i)\frac{\partial}{\partial f_j}\left(\frac{1}{2}\sum_{h\in\mathbb{N}_n}(y_{i,h}-f_h^o(x))^2\right)\\ &= p(x)(y_{i,j}-f_j^o(x))\delta(x-x_i)\,.\end{aligned}$$

The *hinge loss* $V_H(u)=(u)_+$ is associated with the unilateral constraint $\check{\phi}_{i,j}(f(x)) = 1-y_{i,j}\cdot f_j(x)$, which gives rise to $\phi_i^{\overset{\geq}{=}}(f(x)) = \sum_{j\in\mathbb{N}_n} V_H\circ\check{\phi}_{i,j}(f_j(x)) = \sum_{j\in\mathbb{N}_n}(1-y_{i,j}\cdot f_j(x))_+$. In this case, the reaction of the i-th constraint is given by

$$\begin{aligned}\omega_{i,j}^{\overset{\geq}{=}}(x) &= (\uparrow \phi_i^{\overset{\geq}{=}}(f^o(x)))_j = -p(x)1_{\mathscr{X}_i}(x)\overline{\partial_{f_j}}\phi_i^{\overset{\geq}{=}}(f^o(x))\\ &= -p(x)\delta(x-x_i)\overline{\partial_{f_j}}\left((1-y_{i,j}\cdot f_j^o(x))_+\right)\,,\end{aligned}$$

where $-\overline{\partial_{f_j}}\left((1-y_{i,j}\cdot f_j^o(x))_+\right)$ is equal to 0 if $(1-y_{i,j}\cdot f_j^o(x))<0$ and to $y_{i,j}$ if $(1-y_{i,j}\cdot f_j^o(x))>0$, whereas if $(1-y_{i,j}\cdot f_j^o(x))=0$, $-\overline{\partial_{f_j}}\left((1-y_{i,j}\cdot f_j^o(x))_+\right)$ denotes an element (to be found) either of the set $[0,1]$, when $y_{i,j}=1$, or of $[-1,0]$, when $y_{i,j}=-1$.

In both cases, due to the presence of the Dirac delta, we end up with

$$f_j^o(x) = \frac{1}{\gamma_j}\sum_{i=1}^m g\otimes\omega_{i,j}^{\overset{\geq}{=}}(x) = \sum_{i=1}^m \alpha_{i,j}g(x-x_i), \tag{21}$$

where the $\alpha_{i,j}$'s are suitable scalar coefficients (different in the case of hinge or quadratic loss). The classical solution schemes of Ridge Regression and Support Vector Machines can be applied to find the $\alpha_{i,j}$'s.

We conclude with a remark on the notion of constraint reaction in the classic case of supervised learning from examples. In the case of the quadratic loss, it is clear that there is a non-null reaction whenever the associated hard constraint is not satisfied. This happens iff $y_{i,j}\neq f_j^o(x_i)$ for at least one index j. This corresponds to the well-known fact that usually all the examples are support vectors (apart from the case of an interpolating solution). On the opposite side, a set of support vectors that is a proper subset of all the examples usually arises in the hinge loss case.

4.2 Linear Constraints with Supervised Examples Available

Let $\mathscr{X}=\mathbb{R}^d$ and $\forall i\in\mathbb{N}_m$, $\forall x\in\mathscr{X}$ let $\phi_i(f(x)) := a_i'f(x)-b_i(x)=0$, where $a_i\in\mathbb{R}^n$ and $b_i(x)$ is a real-valued function. We consider hard holonomic bilateral constraints that can be written as $Af(x)=b(x)$, where $A\in\mathbb{R}^{m,n}$, and $b\in\mathscr{C}_0^{2k}(\mathscr{X},\mathbb{R}^m)$ is a smooth vector-valued function with compact support. We assume $n>m$ and rank$(A)=m$. We discuss the solution for the class of so-called *rotationally-symmetric differential operators P*, as defined by [10]. These are operators of the form $P := [\sqrt{\rho_0}D_0,\sqrt{\rho_1}D_1,\ldots,\sqrt{\rho_\kappa}D_\kappa,\ldots,\sqrt{\rho_k}D_k]'$, where the operators D_κ satisfy $D_{2r} = \Delta^r = \nabla^{2r}$ and $D_{2r+1} = \nabla\nabla^{2r}$ (Δ denotes the Laplacian operator and ∇ the gradient, with the additional condition $D_0 f = f$, see also [18, 25]), $\rho_0, \rho_1, \dots, \rho_\kappa, \dots, \rho_k \geq 0$, and $\rho_0, \rho_k > 0$. Such operators correspond via $L = (P^\star)'P$ to $L = \sum_{\kappa=0}^{k}(-1)^\kappa \rho_\kappa \nabla^{2\kappa}$, which is an invertible operator on $\mathscr{W}^{k,2}(\mathbb{R}^d)$ (see, e.g., Lemma 5.1 in [10]). In addition, we assume that $\gamma = \bar{\gamma} > 0$, where $\bar{\gamma}$ has constant and equal components, and again, an overloaded notation is used. We also assume that m_d additional supervised pairs (x_κ, y_κ) ($\kappa = 1, \dots, m_d$) induce soft constraints expressed in terms of the quadratic loss. A slight variation of Theorem 2 (see Theorem 3, reported for completeness in Appendix 2, and applied here with $\mu = 1$), implies that a constrained local minimizer f^o of the associated functional satisfies the Euler-Lagrange equations

$$\bar{\gamma} L f^o + A'\lambda + \frac{1}{m_d}\sum_{\kappa=1}^{m_d}(f^o(\cdot) - y_\kappa)\delta(\cdot - x_\kappa) = 0. \tag{22}$$

After some straightforward computations (see Appendix 3 for details), one obtains for the overall constraint reaction (of both hard and soft constraints) the expression

$$\omega(x) = c(x) + \bar{\gamma}\sum_{\kappa=1}^{m_d} Q\alpha_\kappa^{(ql)}\delta(x - x_\kappa),$$

where the $\alpha_\kappa^{(ql)}$'s ($\kappa = 1, \dots, m_d$) are suitable coefficients to be determined (and "ql" stands for "quadratic loss"), whereas $c(x) := \bar{\gamma}A'(AA')^{-1}Lb(x)$ and $Q := I_n - A'[AA']^{-1}A$, where I_n is the identity matrix of size n. Finally, as a unilateral variation of this example, we mention the remarkable case of a unilateral constraint $f(x) \geq 0$ (componentwise), which makes sense when the components of f represent, e.g., mass or probability densities.

4.3 Box Constraints

To fix ideas, let us consider, as a simple sketch, the case of the constraint defined by the rule $\forall x \in \mathscr{B} \subset \mathscr{X} : f(x) - 1 = 0$ [16], with $\mathscr{B} = [a, b] \subset \mathbb{R}$. As depicted in Fig. 2(b), after softening the constraint in the way illustrated by [16], the reaction of the constraint becomes a rectangular impulse, instead of a Dirac distribution as in the case of supervised learning from point-wise examples. However, the latter can be still thought as a degenerate case of the rectangular impulse. For the case in which l in (3) is not finite and the infinite-order differential regularization operator that corresponds to the Gaussian kernel with width σ is used, Theorem 1 (iii) for a single box provides for the solution the representation (up to a positive constant)

$$(g \otimes 1_{\mathscr{B}})(x) \propto erf((x-a)/\sigma) - erf((x-b)/\sigma),$$

where $1_{\mathscr{B}}(\cdot)$ is the characteristic function of $\mathscr{B}$. This clearly indicates that the solution can be thought of as the response of a system with a certain free-space Green's function g, which we call *plain kernel*, to a Dirac delta (supervised pair) or to a rectangular impulse (box constraint). The latter case is just an example to show that the representation of the solution is not based on the plain kernel anymore, but on a function that arises from its marriage with the reaction of the constraint. Basically, the emergence of plain kernels is just a consequence of the degeneration of the reaction of the constraint to a Dirac distribution.

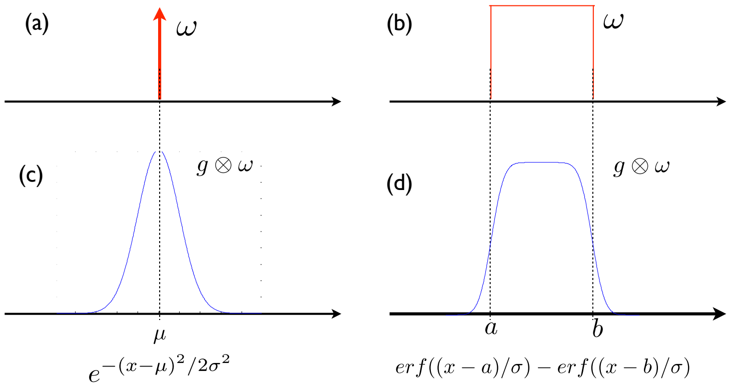


Fig. 2 (a) Constraint reactions corresponding to a classic supervised pair (b) and to the (softened) constraint $\forall x \in [a,b]: \ f(x) = 1$ (box constraint). In (c) and (d) we can see the emergence, resp., of the plain and box kernel. Here, the infinite-order differential regularization operator in (3) is such that the free-space Green's function of L yields the classic Gaussian kernel.

5 Discussion

We have introduced a general framework of learning that involves agents acting in a constraint-based environment, for hard and soft constraints of holonomic type and for soft point-wise constraints. The application of the theory to the chosen case studies illustrates the generality of the approach, which can be fully grasped as we acquire the notion of constraint reaction. The theory presented in this chapter extends the framework of kernel machines to more general hard and soft constraints, and opens the doors to an in-depth re-thinking of the notion of plain kernel that, under some assumptions, was proved to be the Green's function of the differential operator [10] used in the formulation of the learning problem. Interestingly, the notion of constraint reaction and the corresponding representer theorems show that the solution to the learning problem is given in terms of new kinds of kernels that, unlike the plain kernels, also involve the structure of the corresponding constraints: indeed, they originate from the marriage of the plain kernels with the reactions of the constraints. Finally, when the probability density of the data is unknown, the theory suggests to explore the numerical solution of the Euler-Lagrange equations by using unsupervised data, e.g., to learn the probability density itself.

Acknowledgements. G. Gnecco and M. Sanguineti were partially supported by the project "Methodologies for the Approximate Solution of Team Optimization and Strategic Interaction Problems" granted by INDAM-GNAMPA (National Institute of High Mathematics - National Group for Mathematical Analysis, Probability, and Their Application) and the Progetto di Ricerca di Ateneo 2012 "Models and Computational Methods for High-Dimensional Data", granted by the University of Genoa.

Appendix 1

The next lemma, which is a consequence of the Implicit Function Theorem, is exploited in the proof of Theorem 2 in Section 2.2. For a scalar-valued function u of various vector arguments, we denote by $\nabla_i u$ the column vector of partial derivatives of u with respect to all the components of the i-th vector argument. Instead, for a vector-valued function u of various vector arguments, $\nabla_i u$ denotes the matrix whose h-th row is the transpose of the column vector $\nabla_i u_h$.

Lemma 1. *Let $\Omega \subseteq \mathbb{R}^d$, $\mathscr{Y} \subseteq \mathbb{R}^{n_1}$, $\mathscr{Z} \subseteq \mathbb{R}^{n_2}$ be open subsets, and $\phi : \Omega \times \mathscr{Y} \times \mathscr{Z} \to \mathbb{R}^{n_2}$ a given function. Let also $y : \Omega \to \mathscr{Y}$ and $z : \Omega \to \mathscr{Z}$ be other given functions, which satisfy the (vector-valued) holonomic and bilateral constraint*

$$\phi(x, y(x), z(x)) = 0\,, \forall x \in \Omega\,.$$

Suppose also that $\phi \in \mathscr{C}^{k+1}(\Omega \times \mathscr{Y} \times \mathscr{Z}, \mathbb{R}^{n_2})$ for some positive integer $k \geq 1$ and that, for each $x \in \Omega$, the Jacobian matrix

$$\nabla_3 \phi(x, y(x), z(x)) := \begin{pmatrix} \frac{\partial \phi_1(x,y(x),z(x))}{\partial z_1} & \cdots & \frac{\partial \phi_1(x,y(x),z(x))}{\partial z_{n_2}} \\ \cdots & \cdots & \cdots \\ \frac{\partial \phi_{n_2}(x,y(x),z(x))}{\partial z_1} & \cdots & \frac{\partial \phi_{n_2}(x,y(x),z(x))}{\partial z_{n_2}} \end{pmatrix} \tag{23}$$

is nonsingular (possibly after interchanging locally some components of $y(x)$ by an equal number of components of $z(x)$, and redefining the function ϕ and the vectors $y(x)$ and $z(x)$ according to such a replacement). Now, let η_y be an arbitrary function in $\mathscr{C}_0^k(\Omega, \mathbb{R}^{n_1})$ with compact support Ω_C contained in an open ball of sufficiently small radius, and consider a perturbation $\Delta y(x) := \varepsilon \eta_y(x)$ of the function $y(x)$, where $\varepsilon \in \mathbb{R}$ is sufficiently small. Then, there exists a unique function $\eta_z \in \mathscr{C}_0^k(\Omega, \mathbb{R}^{n_2})$ with compact support Ω_C such that the perturbed holonomic and bilateral constraint

$$\phi(x, y(x) + \Delta y(x), z(x) + \Delta z(x)) = 0\,, \forall x \in \Omega$$

is satisfied for $\Delta z(x)$ of the form

$$\Delta z(x) = \varepsilon \eta_z(x) + O(\varepsilon^2)\,, \tag{24}$$

where the "hidden constant" inside the "big O" notation above does not depend[6] *on x, and $\eta_z(x)$ has the expression*

$$\eta_z(x) = -(\nabla_3\phi(x,y(x),z(x)))^{-1}(\nabla_2\phi(x,y(x),z(x)))\eta_y(x)\,. \tag{25}$$

Moreover, for each $h \in \{1,\dots,k\}$ and $i \in \{1,\dots,n_2\}$, one has, for the i-th component Δz_i of Δz,

$$\frac{\partial^h}{\partial x_{j_1}\dots\partial x_{j_h}}\Delta z_i(x) = \varepsilon\frac{\partial^h}{\partial x_{j_1}\dots\partial x_{j_h}}\eta_{z_i}(x) + O(\varepsilon^2)\,, \tag{26}$$

where, again, the "hidden constants" inside the "big O" notations above do not depend on x.

Proof. Fix $x = x_0 \in \Omega$. Since $\phi \in \mathscr{C}^{k+1}(\Omega \times \mathscr{Y} \times \mathscr{Z}, \mathbb{R}^{n_2})$ for $k \geq 1$ and the Jacobian matrix (23) is nonsingular, one can apply the Implicit Function Theorem, according to which, on a suitable open ball $\mathscr{B}$ of $(0,0)$ of sufficiently small radius $\varepsilon > 0$, there exists a unique function $u \in \mathscr{C}^{k+1}(\mathscr{B}, \mathbb{R}^{n_2})$ such that $u(0,0) = 0$ and

$$\phi(x+\Delta x, y(x)+\Delta y, z(x)+u(\Delta x,\Delta y)) = 0\,, \quad \forall(\Delta x,\Delta y) \in \mathscr{B}\,. \tag{27}$$

Moreover, since[7] $k+1 \geq 2$, each component $u_i(\Delta x,\Delta y)$ of the function $u(\Delta x,\Delta y)$ has the multivariate Taylor expansion

$$u_i(\Delta x,\Delta y) = \sum_{|\alpha|=1} D^\alpha u_i(0,0)(\Delta x,\Delta y)^\alpha + O(\|(\Delta x,\Delta y)\|^2)\,, \tag{28}$$

where $(\Delta x,\Delta y)^\alpha := \prod_{j=1}^d(\Delta x_j)^{\alpha_j}\prod_{j=1}^{n_1}(\Delta y_j)^{\alpha_{d+j}}$, and the term $O(\|(\Delta x,\Delta y)\|^2)$ denotes a function of class $\mathscr{C}^{k+1}(\mathscr{B})$, infinitesimal at $(0,0)$ with order at least 2, where the "hidden" constant inside the "big O" notation above depends only on the local behavior of ϕ on a neighborhood of $(x,y(x),z(x))$, and is independent from x itself, provided that, after the initial choice x_0 for x, x varies inside a

[6] In this formula and in the next one (26) there is, instead, a dependence of the hidden constants on the specific choice of η_y, which may be removed by further assuming $\|\eta_y\|_{\mathscr{C}_0^k(\Omega,\mathbb{R}^{n_1})} \leq M_y$ for some given positive constant M_y.

[7] In the lemma, we have made the assumption $\phi \in \mathscr{C}^{k+1}(\Omega \times \mathscr{Y} \times \mathscr{Z}, \mathbb{R}^{n_2})$ instead of the looser one $\phi \in \mathscr{C}^{k}(\Omega \times \mathscr{Y} \times \mathscr{Z}, \mathbb{R}^{n_2})$ in order to be able to express the remainder in Taylor's polynomial (28) by the integral Lagrange's form, instead, e.g., of the Peano's form (however, for simplicity of notation, in formula (28) we have not reported the explicit expression of the remainder in the integral Lagrange's form). Considering for simplicity the case of a scalar-valued function $u(x)$ of class $\mathscr{C}^2$ depending on a scalar argument x, we recall that one has the expression

$$f(x+\Delta x) = f(x) + f'(x)\Delta x - \int_0^{\Delta x}(t-\Delta x)f''(x+t)dt\,,$$

where the last term is the remainder expressed in the integral Lagrange's form. This formula can be generalized to the multivariate case, and such an extension is used to be able to obtain terms of order $O(\varepsilon^2)$ in (26).

compact subset Ω_C of the projection of the set[8] $\mathscr{B} + (x_0, y(x_0))$ on Ω. Now, let $\eta_y \in \mathscr{C}_0^k(\Omega_C, \mathbb{R}^{n_1}) \subseteq \mathscr{C}_0^k(\Omega, \mathbb{R}^{n_1})$ and set $\Delta x = 0$ and $\Delta y = \Delta y(x) := \varepsilon \eta_y(x)$. Then, we define each component $\Delta z_i(x)$ of the function $\Delta z(x)$ as

$$\begin{aligned} \Delta z_i(x) &:= u_i(0, \varepsilon \eta_y(x)) \\ &= \sum_{|\alpha|=1} D^\alpha u_i(0,0)(0, \varepsilon \eta_y(x))^\alpha + O(\|(0, \varepsilon \eta_y(x))\|^2) \\ &= \varepsilon \sum_{|\alpha|=1} D^\alpha u_i(0,0)(0, \eta_y(x))^\alpha + O(\varepsilon^2), \end{aligned} \tag{29}$$

where the replacement of the term $O(\|(0, \varepsilon \eta_y(x))\|^2)$ by the one $O(\varepsilon^2)$ follows by the fact that $\eta_y(x)$ is fixed and uniformly bounded. Then, (24) follows by setting

$$\eta_{z,i}(x) := \sum_{|\alpha|=1} D^\alpha u_i(0,0)(0, \eta_y(x))^\alpha ,$$

which shows that the function $\eta_{z,i}$ is in $\mathscr{C}_0^k(\Omega_C, \mathbb{R}) \subseteq \mathscr{C}_0^k(\Omega, \mathbb{R})$, likewise η_y is in $\mathscr{C}_0^k(\Omega_C, \mathbb{R}^{n_1}) \subseteq \mathscr{C}_0^k(\Omega, \mathbb{R}^{n_1})$. Finally, the application of the Implicit Function Theorem shows also that the vector $\eta_z(x)$ with components $\eta_{z,i}(x)$ has the expression

$$\eta_z(x) = -(\nabla_3 \phi(x, y(x), z(x)))^{-1} (\nabla_2 \phi(x, y(x), z(x))) \eta_y(x) .$$

Finally, (26) is derived directly by (24), by computing its partial derivatives of order h (i.e., exploiting the expression of the remainder of Taylor's polynomial (28) in Lagrange's integral form, the rule of differentiation under the integral's sign, the chain rule, and the fact that each component of the function η_y is bounded on Ω_C, together with its partial derivatives - up to the order k - with respect to the components of x). □

The meaning of Lemma 1 is the following: in order to be still able to satisfy the holonomic and bilateral constraint, a perturbation $\Delta y(x) := \varepsilon \eta_y(x)$ of the function $y(x)$ implies a perturbation $\Delta z(x) := \varepsilon \eta_z(x)$ (apart from an infinitesimal of order greater than ε) of the function $z(x)$, where η_z depends only on η_y and suitable partial derivatives of ϕ evaluated at the current solution $(x, y(x), z(x))$, but does not depend on ε. The formula (26) shows that also the partial derivatives of $\Delta z(x)$ up to the order k have similar expressions.

Appendix 2

This appendix reports the complete proof of Theorem 2 in Section 2.2.

Proof. (i) Let f^o be a constrained local minimizer over $\mathscr{F}$ of the functional $\mathscr{E}(f) = \| f \|_{P,\gamma}^2$ defined in formula (3). Fix $x_0 \in \hat{\mathscr{X}}$ and a compact subset $\mathscr{X}_C \subset \hat{\mathscr{X}}$ contained in an open ball of sufficiently small radius, and containing x_0, and, after

[8] Here, we denote by $\mathscr{B} + (x_0, y(x_0))$ the translation of the set $\mathscr{B}$ by $(x_0, y(x_0))$.

performing the permutations σ_ϕ and σ_f, re-order the constraints (and the components of f, resp.) in such a way that the ones with indexes $\sigma_\phi(1),\dots,\sigma_\phi(m(x_0))$ $(\sigma_f(1),\dots,\sigma_f(m(x_0))$, resp.) are the first $m(x_0)$ ones. Due to an application of Lemma 1 in Appendix 1, if one fixes arbitrarily the functions $\eta_i \in \mathscr{C}_0^k(\mathscr{X}_C)$ for $i = m(x_0)+1, m(x_0)+2,\dots,n$, then, for every sufficiently small $|\varepsilon| > 0$, the bilateral holonomic constraints are met for a function f whose components f_j have the following expressions:

$$\begin{aligned}
f_1 &= f_1^o + \varepsilon\eta_1 + O(\varepsilon^2)\,,\\
f_2 &= f_2^o + \varepsilon\eta_2 + O(\varepsilon^2)\,,\\
&\dots\\
f_{m(x_0)} &= f_{m(x_0)}^o + \varepsilon\eta_{m(x_0)} + O(\varepsilon^2)\,,\\
f_{m(x_0)+1} &= f_{m(x_0)+1}^o + \varepsilon\eta_{m(x_0)+1}\,,\\
f_{m(x_0)+2} &= f_{m(x_0)+2}^o + \varepsilon\eta_{m(x_0)+2}\,,\\
&\dots\\
f_n &= f_n^o + \varepsilon\eta_n\,,
\end{aligned} \tag{30}$$

where the functions $\eta_i \in \mathscr{C}_0^k(\mathscr{X}_C)$, for $i = 1,\dots,m(x_0)$, are still determined by Lemma 1. In particular, by setting $y(x) = [f_{m(x_0)+1}^o(x), f_{m(x_0)+2}^o(x),\dots,f_n^o(x)]'$, $z(x) = [f_1^o(x),\dots,f_{m(x_0)}^o(x)]'$, $\phi = [\phi_1,\dots,\phi_{m(x_0)}]'$, $\eta_y = [\eta_{m(x_0)+1},\eta_{m(x_0)+2},\dots,\eta_n]'$, and $\eta_z = [\eta_1,\dots,\eta_{m(x_0)}]'$, one has

$$\eta_z(x) = -(\nabla_3\phi(x,y(x),z(x)))^{-1}(\nabla_2\phi(x,y(x),z(x)))\eta_y(x)\,. \tag{31}$$

Moreover, due to (26), the partial derivatives, up to the order k, of the first $m(x_0)$ components of f, have expressions similar to (30), and contain terms of order $O(\varepsilon^2)$. This implies that $\mathscr{E}(f)$ can be written as

$$\begin{aligned}
\mathscr{E}(f) &= \sum_{j=1}^n \gamma_j < P(f^o+\varepsilon\eta)_j, P(f^o+\varepsilon\eta)_j > + O(\varepsilon^2)\\
&= \sum_{j=1}^n \gamma_j < Pf_j^o, Pf_j^o > + 2\varepsilon\sum_{j=1}^n \gamma_j < Pf_j^o, P\eta_j >\\
&\quad + \varepsilon^2\sum_{j=1}^n \gamma_j < P\eta_j, P\eta_j > + O(\varepsilon^2)\\
&= \sum_{j=1}^n \gamma_j < Pf_j^o, Pf_j^o > + 2\varepsilon\sum_{j=1}^n \gamma_j < Pf_j^o, P\eta_j > + O(\varepsilon^2)\,.
\end{aligned}$$

Moreover, by an application of Green's formula (see, e.g., Proposition 5.6.2 in [3]), we have

$$< Pf_j^o, P\eta_j > = < (P^\star)'Pf_j^o, \eta_j > = < Lf_j^o, \eta_j >\,,$$

where $P^{\star}$ is the formal adjoint of the operator P. Now, we define locally the row vector function $\lambda(x)$ as follows:

$$\lambda(x) := -[\gamma_1 (Lf^o)_1(x), \dots, \gamma_{m(x_0)} (Lf^o)_{m(x_0)}(x)] (\nabla_3 \phi(x, y(x), z(x)))^{-1} . \quad (32)$$

Then, with such a definition, and exploiting formula (31), one obtains

$$\begin{aligned}\sum_{j=1}^{m(x_0)} \gamma_j < Pf_j^o, P\eta_j > &= \sum_{j=1}^{m(x_0)} \gamma_j < Lf_j^o, \eta_j > \\ &= \int_{\mathscr{X}} \lambda(x) (\nabla_2 \phi(x, y(x), z(x))) \eta_y(x) dx .\end{aligned}$$

Summing up, one has

$$\begin{aligned}\mathscr{E}(f) - \mathscr{E}(f^o) = 2\varepsilon \int_{\mathscr{X}} \Big(&[\gamma_{m(x_0)+1} (Lf^o)_{m(x_0)+1}(x), \dots, \gamma_n (Lf^o)_n(x)] \\ &+ \lambda(x) (\nabla_2 \phi(x, y(x), z(x))) \Big) \eta_y(x) dx + O(\varepsilon^2) .\end{aligned}$$

Now, since $\mathscr{E}(f) - \mathscr{E}(f^o) \geq 0$ for $|\varepsilon| > 0$ sufficiently small due to the local optimality of f^o, and $\eta_y \in \mathscr{C}_0^k(\mathscr{X}_C, \mathbb{R}^{n-m(x_0)})$ is arbitrary, by applying the fundamental lemma of the calculus of variations (see, e.g., Section 2.2 in [7]) we conclude that

$$[\gamma_{m(x_0)+1} (Lf^o)_{m(x_0)+1}(x), \dots, \gamma_n (Lf^o)_n(x)] + \lambda(x) (\nabla_2 \phi(x, y(x), z(x))) = 0$$

on $\mathscr{X}_C$. This, together with the definition (32) of $\lambda(x)$, shows that (19) holds on $\mathscr{X}_C$. Finally, by varying the point x_0, one obtains (19) on the whole $\hat{\mathscr{X}}$.

(ii) follows by (19), the definition of the free-space Green's function g of L as the solution of $Lg = \delta$ (where δ denotes the Dirac delta, centered in 0), and the stated assumptions on L and g.

(iii) For the case of unilateral constraints, of course the constraints that are inactive in x_0 at local optimality are not taken into account locally, so the condition about the nonsingularity of the Jacobian matrix has to be referred only to the constraints that are active in x_0 at local optimality. Moreover, all the arguments used to derive (i) and (ii) still hold (of course, restricting the analysis to the active constraints in x_0 at local optimality, and replacing the ϕ_i's by the $\check{\phi}_i$'s), since, for every sufficiently small $|\varepsilon| > 0$, a function f constructed as in the proof of (i) still satisfies with equality the active constraints in x_0 at local optimality.

Finally, we show that each Lagrange multiplier function $\lambda_i(x)$ is nonpositive. Without loss of generality, we can restrict the analysis to the points of continuity of $\lambda_i(x)$. Suppose by contradiction that there exists one such point $\hat{x}_0 \in \hat{\mathscr{X}}$ such that $\lambda_i(\hat{x}_0) > 0$. Then, by continuity $\lambda_i(x) > 0$ on a sufficiently small open ball centered on $\hat{x}_0$. For simplicity of notation, we also suppose that all the constraints defined on $\hat{x}_0$ are active in $\hat{x}_0$ at local optimality. Then, due to the condition about the nonsingularity of the Jacobian matrix, there is a vector $u = [u_1, \dots, u_{m(\hat{x}_0)}]'$ such that $\nabla_3 \check{\phi}(\hat{x}_0, y(\hat{x}_0), z(\hat{x}_0))u = e_i$, where e_i is a column vector of all 0's, with the exception of the i-th component, which is 1. Then, by an application of the Implicit Function Theorem (likewise in the proof of Lemma 1), for every sufficiently small $\varepsilon > 0$ (but in this case, not for every sufficiently small $\varepsilon < 0$) one can construct a feasible smooth perturbation $f(x)$ of $f^o(x)$ such that its components f_j satisfy

$$\begin{aligned} f_1(x) &= f_1^o(x) + \varepsilon\eta_1(x) + O(\varepsilon^2), \\ f_2(x) &= f_2^o(x) + \varepsilon\eta_2(x) + O(\varepsilon^2), \\ &\dots \\ f_{m(\hat{x}_0)}(x) &= f_{m(\hat{x}_0)}^o(x) + \varepsilon\eta_{m(\hat{x}_0)}(x) + O(\varepsilon^2), \\ f_{m(\hat{x}_0)+1}(x) &= f_{m(\hat{x}_0)+1}^o(x), \\ f_{m(\hat{x}_0)+2}(x) &= f_{m(\hat{x}_0)+2}^o(x), \\ &\dots \\ f_n(x) &= f_n^o(x), \end{aligned} \tag{33}$$

for suitable functions $\eta_1, \dots, \eta_{m(\hat{x}_0)} \in \mathscr{C}_0^k(\mathscr{X}_C)$ such that $\eta_1(\hat{x}_0) = u_1$, $\eta_2(\hat{x}_0) = u_2$, ..., $\eta_{m(\hat{x}_0)}(\hat{x}_0) = u_{m(\hat{x}_0)}$, and such that $\mathscr{E}(f) - \mathscr{E}(f^o)$, apart from an infinitesimal of order $O(\varepsilon^2)$, is directly proportional to

$$\varepsilon[\gamma_1(Lf^o)_1(\hat{x}_0), \dots, \gamma_{m(x_0)}(Lf^o)_{m(\hat{x}_0)}(\hat{x}_0)]u = -\varepsilon\lambda(\hat{x}_0)e_i = -\varepsilon\lambda_i(\hat{x}_0) < 0,$$

which contradicts the local optimality of f^o. Then, one has $\lambda_i(\hat{x}_0) \leq 0$. □

The following theorem is a slight variation of Theorem 2, and is exploited in the example in Section 4.2.

Theorem 3. *(*REPRESENTER THEOREM FOR HARD HOLONOMIC CONSTRAINTS MIXED WITH SOFT QUADRATIC POINT-WISE CONSTRAINTS, CASE OF DISTRIBUTIONAL LAGRANGE MULTIPLIERS*). Let us consider the minimization of the functional*

$$\begin{aligned} \mathscr{L}_s'(f) &:= \frac{1}{2} \parallel f \parallel_{P,\gamma}^2 + \frac{\mu}{m_d} \sum_{\kappa=1}^{m_d} \sum_{j=1}^{n} V_Q(y_{\kappa,j} - f_j(x_\kappa)) \\ &= \frac{1}{2} \sum_{j=1}^{n} \gamma_j < Pf_j, Pf_j > + \frac{\mu}{2m_d} \sum_{\kappa=1}^{m_d} \sum_{j=1}^{n} (y_{\kappa,j} - f_j(x_\kappa))^2 \end{aligned} \tag{34}$$

(a particular case of the functional (4)), where $\mu \geq 0$ and m_d is the number of supervised examples, in the case of $m < n$ hard bilateral constraints of holonomic type, which define the subset

$$\mathscr{F}_\phi := \{f \in \mathscr{F} : \ \forall i \in \mathbb{N}_m, \ \forall x \in \mathscr{X}_i \subseteq \mathscr{X} : \ \phi_i(x, f(x)) = 0\}$$

of the function space $\mathscr{F}$, where $\forall i \in \mathbb{N}_m : \phi_i \in \mathscr{C}^\infty(\mathrm{cl}(\mathscr{X}_i) \times \mathbb{R}^m)$. Let $f^o \in \mathscr{F}_C$ be any constrained local minimizer of (34), and let the holonomic constraints be defined in such a way that either $Lf^o \in \mathscr{C}^0(\mathscr{X}, \mathbb{R}^n)$ or they are of the form $Af(x) = b(x)$, where $A \in \mathbb{R}^{m,n}$ with $m < n$ and $\mathrm{rank}(A) = m$, *and $b \in \mathscr{C}_0^{2k}(\mathscr{X}, \mathbb{R}^m)$. Let us assume that for any $\hat{\mathscr{X}}$ and for every x_0 in the same $\hat{\mathscr{X}}$ we can find two permutations σ_f and σ_ϕ of the indexes of the n functions f_j and of the m constraints ϕ_i, such that $\phi_{\sigma_\phi(1)}, \ldots, \phi_{\sigma_\phi(m(x_0))}$ refer to the constraints actually defined in x_0, and the Jacobian matrix*

$$\frac{\partial(\phi_{\sigma_\phi(1)}, \ldots, \phi_{\sigma_\phi(m(x_0))})}{\partial(f^o_{\sigma_f(1)}, \ldots, f^o_{\sigma_f(m(x_0))})}, \tag{35}$$

evaluated in x_0, is not singular. Suppose also that (35) is of class $\mathscr{C}^\infty(\hat{\mathscr{X}}, \mathbb{R}^n)$. Then, the following hold.

(i) There exists a set of distributions λ_i defined on $\hat{\mathscr{X}}$, $i \in \mathbb{N}_m$, such that, in addition to the above constraints, f^o satisfies on $\hat{\mathscr{X}}$ the Euler-Lagrange equations

$$\gamma L f^o + \sum_{i=1}^{m} \lambda_i 1_{\mathscr{X}_i}(\cdot) \cdot \nabla_f \phi_i(\cdot, f^o(\cdot)) + \frac{\mu}{m_d} \sum_{\kappa=1}^{m_d} (f^o(\cdot) - y_\kappa)\delta(\cdot - x_\kappa) = 0, \tag{36}$$

where $\gamma L := [\gamma_1 L, \ldots, \gamma_n L]'$ is a spatially-invariant operator, and $\nabla_f \phi_i$ is the gradient w.r.t. the second vector argument f of the function ϕ_i.

(ii) Let $\gamma^{-1} g := [\gamma_1^{-1} g, \ldots, \gamma_n^{-1} g]'$. If for all i one has $\mathscr{X}_i = \mathscr{X} = \mathbb{R}^d$, L is invertible on $\mathscr{W}^{k,2}(\mathscr{X})$, and there exists a free-space Green's function g of L that belongs to $\mathscr{W}^{k,2}(\mathscr{X})$, then f^o has the representation

$$f^o(\cdot) = \sum_{i=1}^{m} \gamma^{-1} g(\cdot) \vec{\otimes} \phi_i(\cdot, f^o(\cdot)) - \frac{\mu}{m_d} \sum_{\kappa=1}^{m_d} (f^o(\cdot) - y_\kappa)\gamma^{-1} g(\cdot - x_\kappa), \tag{37}$$

where $g \vec{\otimes} \phi_i := g \otimes \omega_i$ and $\omega_i(\cdot) := \uparrow \phi_i(\cdot, f^o(\cdot)) := -\lambda_i(\cdot) 1_{\mathscr{X}_i}(\cdot) \nabla_f \phi_i(\cdot, f^o(\cdot))$.

(iii) For the case of $m < n$ unilateral constraints of holonomic type, which define the subset $\mathscr{F}_{\check{\phi}} := \{f \in \mathscr{F} : \forall i \in \mathbb{N}_m, \forall x \in \mathscr{X}_i \subseteq \mathscr{X},\ \check{\phi}_i(x, f(x)) \geq 0\}$ of the function space $\mathscr{F}$, (i) and (ii) still hold (with every occurrence of ϕ_i replaced by $\check{\phi}_i$) if one requires the nonsingularity of the Jacobian matrix (see (35)) to hold when restricting the constraints defined in x_0 to the ones that are active in x_0 at local optimality. Moreover, each Lagrange multiplier λ_i is nonpositive and locally equal to 0 *when the correspondent constraint is locally inactive at local optimality.*

Proof. For $\mu = 0$ (or equivalently, when no supervised examples are available) and an additional smoothness assumption on f^o, the theorem reduces to Theorem 2. For the general case $\mu \geq 0$, one can show that the differences with respect to the proof of Theorem 2 are the following:

- there is an additional term $\frac{\mu}{m_d} \sum_{\kappa=1}^{m_d} (f^o(x) - y_\kappa)\delta(x - x_\kappa)$ in the Euler-Lagrange equations, due to the presence of the supervised examples;

- in general, the Lagrange multipliers $\lambda_i(\cdot)$ are not functions, likewise in Theorem 2, but distributions, obtained by a variation of formula (32), which is well-defined in a distributional sense since the Jacobian matrix (35) is locally invertible and infinitely smooth, and since either $Lf^o \in \mathscr{C}^0(\mathscr{X}, \mathbb{R}^n)$ or $Af(x) = b(x)$ hold (with the stated assumptions on A and b). More precisely, formula (32) is replaced by

$$\begin{aligned}\lambda := & -[\gamma_1(Lf^o)_1, \ldots, \gamma_{m(x_0)}(Lf^o)_{m(x_0)}](\nabla_3\phi(\cdot, y(\cdot), z(\cdot)))^{-1} \\ & + \left(\frac{\mu}{m_d}\sum_{\kappa=1}^{m_d}[(y_{\kappa,1} - f_1^o), \ldots, (y_{\kappa,m(x_0)} - f^o_{m(x_0)})]\right. \\ & \left.\delta(\cdot - x_\kappa)\right)(\nabla_3\phi(\cdot, y(\cdot), z(\cdot)))^{-1}, \end{aligned} \tag{38}$$

  where now λ is a row vector distribution;
- differently from Theorem 2, additional smoothness of f^o is not required, since only (35) is required to be infinitely smooth. □

Appendix 3

This appendix reports the complete derivations for the determination of the constraint reactions in the example of Section 4.2.

Let us determine the vector of distributional Lagrange multipliers λ. We start noting that

$$\begin{aligned} ALf(x) &= A\sum_{\kappa=0}^{k}(-1)^\kappa \rho_\kappa \nabla^{2\kappa} f(x) \\ &= \sum_{\kappa=0}^{k}(-1)^\kappa \rho_\kappa A\nabla^{2\kappa} f(x) \\ &= \sum_{\kappa=0}^{k}(-1)^\kappa \rho_\kappa \nabla^{2\kappa} Af(x) \\ &= \sum_{\kappa=0}^{k}(-1)^\kappa \rho_\kappa \nabla^{2\kappa} b(x) \\ &= Lb(x), \end{aligned}$$

where $Lb \in \mathscr{C}_0^0(\mathscr{X}, \mathbb{R}^m)$ has compact support. Hence, from (22) we get

$$\bar{\gamma}Lb(x) + A\left[A'\lambda(x) + \sum_{\kappa=1}^{m_d}\frac{(f^o(x) - y_\kappa)}{m_d}\delta(x - x_\kappa)\right] = 0.$$

So, the Lagrange multiplier distribution λ is given by

$$\lambda = -[AA']^{-1}\left(\bar{\gamma}Lb + \frac{1}{m_d}\sum_{\kappa=1}^{m_d}A(f^o(\cdot) - y_\kappa)\delta(\cdot - x_\kappa)\right).$$

Now, if we plug this expression for λ into the Euler-Lagrange equations (22), we get

$$\bar{\gamma} L f^o(x) = c(x) + \frac{1}{m_d} \sum_{\kappa=1}^{m_d} Q(y_\kappa - f^o(x)) \delta(x - x_\kappa),$$

where $c(x) := \bar{\gamma} A'(AA')^{-1} Lb(x)$ and $Q := I_n - A'[AA']^{-1}A$. Let $\alpha_\kappa^{(ql)} := \frac{1}{m_d} \bar{\gamma}^{-1}(y_\kappa - f^o(x_\kappa))$. By inverting the operator L, we get

$$f^o(x) = \bar{\gamma}^{-1} \int_{\mathscr{X}} g(\zeta) c(x - \zeta) d\zeta + \sum_{\kappa=1}^{m_d} Q \alpha_\kappa^{(ql)} g(x - x_\kappa). \tag{39}$$

So, the overall constraint reaction (of both hard and soft constraints) is

$$\omega(x) = c(x) + \bar{\gamma} \sum_{\kappa=1}^{m_d} Q \alpha_\kappa^{(ql)} \delta(x - x_\kappa).$$

The coefficients $\alpha_\kappa^{(ql)}$ can be determined by the following scheme. Denote by $y := [y_1, \dots, y_{m_d}] \in \mathbb{R}^{n,m_d}$ the matrix of targets, where the κ-th column is associated with the corresponding example x_κ, and $\alpha^{(ql)} := [\alpha_1^{(ql)}, \dots, \alpha_{m_d}^{(ql)}] \in \mathbb{R}^{n,m_d}$. By the definition of $\alpha^{(ql)}$ we get

$$\bar{\gamma} m_d \alpha^{(ql)} + Q \alpha^{(ql)} G = y - \bar{\gamma}^{-1} \int_{\mathscr{X}} g(\zeta) H(\zeta) d\zeta,$$

where G is the Gram matrix of the input data and the kernel g, and $H : \mathscr{X} \to \mathbb{R}^{n,m_d}$ is the matrix-valued function whose κ-th column is given by the function $c(x_\kappa - \cdot)$. The existence of a solution $\alpha^{(ql)}$ to the linear system above follows by a slight modification of Theorem 1 in [10] (since for $\rho_0 > 0$, $\|\cdot\|_{P,\bar{\gamma}}$ is a Hilbert-space norm on $\mathscr{W}^{k,2}(\mathbb{R}^d)$ by Proposition 3 in [10], and the square loss is convex and continuous) and the nonsingularity of the Jacobian matrix (35) associated with the set of hard constraints $Af(x) = b(x)$.

We conclude discussing the admissibility of the obtained solution (39). By an application of Theorem 3 in [10] about the smoothness properties of free-space Green's functions, it follows that, for this problem, $g \in \mathscr{W}^{k,2}(\mathbb{R}^d)$. This implies that $f^o \in \mathscr{F}$, $\mathscr{L}_s'(f^o)$ is finite, and f^o is a constrained global minimizer, too (thanks to the convexity of the problem).

References

1. Adams, R.A., Fournier, J.F.: Sobolev Spaces, 2nd edn. Academic Press (2003)
2. Argyriou, A., Micchelli, C.A., Pontil, M.: When is there a representer theorem? Vector versus matrix regularizers. Journal of Machine Learning Research 10, 2507–2529 (2009)
3. Attouch, H., Buttazzo, G., Michaille, G.: Variational Analysis in Sobolev and BV Spaces. Applications to PDEs and Optimization. SIAM, Philadelphia (2006)

4. Diligenti, M., Gori, M., Maggini, M., Rigutini, L.: Multitask kernel-based learning with logic constraints. In: Proc. 19th European Conf. on Artificial Intelligence, pp. 433–438 (2010)
5. Diligenti, M., Gori, M., Maggini, M., Rigutini, L.: Bridging logic and kernel machines. Machine Learning 86, 57–88 (2012)
6. Dinuzzo, F., Schoelkopf, B.: The representer theorem for Hilbert spaces: A necessary and sufficient condition. In: Proc. Neural Information Processing Systems (NIPS) Conference, pp. 189–196 (2012)
7. Giaquinta, M., Hildebrand, S.: Calculus of Variations I, vol. 1. Springer (1996)
8. Gnecco, G., Gori, M., Melacci, S., Sanguineti, M.: Learning with hard constraints. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 146–153. Springer, Heidelberg (2013)
9. Gnecco, G., Gori, M., Melacci, S., Sanguineti, M.: A theoretical framework for supervised learning from regions. Neurocomputing 129, 25–32 (2014)
10. Gnecco, G., Gori, M., Sanguineti, M.: Learning with boundary conditions. Neural Computation 25, 1029–1106 (2013)
11. Gnecco, G., Gori, M., Melacci, S., Sanguineti, M.: Foundations of support constraints machines. Neural Computation (to appear)
12. Gori, M., Melacci, S.: Constraint verification with kernel machines. IEEE Transactions on Neural Networks and Learning Systems 24, 825–831 (2013)
13. Klement, E.P., Mesiar, R., Pap, E.: Triangular Norms. Kluwer (2000)
14. Kunapuli, G., Bennett, K.P., Shabbeer, A., Maclin, R., Shavlik, J.: Online knowledge-based support vector machines. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part II. LNCS (LNAI), vol. 6322, pp. 145–161. Springer, Heidelberg (2010)
15. Mangasarian, O.L., Wild, E.W.: Nonlinear knowledge-based classification. IEEE Transactions on Neural Networks 19, 1826–1832 (2008)
16. Melacci, S., Gori, M.: Learning with box kernels. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(11), 2680–2692 (2013)
17. Melacci, S., Maggini, M., Gori, M.: Semi–supervised learning with constraints for multi–view object recognition. In: Alippi, C., Polycarpou, M., Panayiotou, C., Ellinas, G. (eds.) ICANN 2009, Part II. LNCS, vol. 5769, pp. 653–662. Springer, Heidelberg (2009)
18. Poggio, T., Girosi, F.: A theory of networks for approximation and learning. Technical report. MIT (1989)
19. Schwartz, L.: Théorie des distributions. Hermann, Paris (1978)
20. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
21. Sun, Z., Zhang, Z., Wang, H., Jiang, M.: Cutting plane method for continuously constrained kernel-based regression. IEEE Transactions on Neural Networks 21, 238–247 (2010)
22. Suykens, J.A.K., Alzate, C., Pelckmans, K.: Primal and dual model representations in kernel-based learning. Statistics Surveys 4, 148–183 (2010)
23. Theodoridis, S., Slavakis, K., Yamada, I.: Adaptive learning in a world of projections. IEEE Signal Processing Magazine 28, 97–123 (2011)
24. Tikhonov, A.N., Arsenin, V.Y.: Solution of ill-posed problems. W.H. Winston, Washington, DC (1977)
25. Yuille, A.L., Grzywacz, N.M.: A mathematical analysis of the motion coherence theory. International Journal of Computer Vision 3, 155–175 (1989)

Baseline-Free Sampling in Parameter Exploring Policy Gradients: Super Symmetric PGPE

Frank Sehnke and Tingting Zhao

Abstract. Policy Gradient methods that explore directly in parameter space are among the most effective and robust direct policy search methods and have drawn a lot of attention lately. The basic method from this field, Policy Gradients with Parameter-based Exploration, uses two samples that are symmetric around the current hypothesis to circumvent misleading reward in *asymmetrical* reward distributed problems gathered with the usual baseline approach. The exploration parameters are still updated by a baseline approach – leaving the exploration prone to asymmetric reward distributions. In this paper we will show how the exploration parameters can be sampled quasi-symmetrically despite having limited instead of free parameters for exploration. We give a transformation approximation to get quasi symmetric samples with respect to the exploration without changing the overall sampling distribution. Finally we will demonstrate that sampling symmetrically for the exploration parameters as well is superior to the original sampling approach, in terms of samples needed and robustness.

Subtracting an optimal baseline minimizes the variance of gradient estimates with their unbiasedness being maintained, which can provide more stable gradient estimates and thus also lead to faster convergence. However, the optimal baseline technique cannot avoid misleading reward in asymmetrical reward distributed problems. The superiority of the proposed symmetrical sampling over optimal baseline will be

Frank Sehnke
Center for Solar Energy and Hydrogen Research
(Zentrum für Sonnenenergie- und Wasserstoff-Forschung),
Industriestr. 6, Stuttgart, BW 70565 Germany
e-mail: frank.sehnke@zsw-bw.de

Tingting Zhao
Tokyo Institute of Technology,
2-12-1 O-okayama, Meguro-ku, Tokyo 152-8552, Japan
e-mail: tingting@sg.cs.titech.ac.jp

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_13

demonstrated through experiments. It will also be shown that combining symmetrical sampling, where misleading rewards are present, with an optimal baseline, where they are not, is virtually indistinguishable in performance to the full symmetrical approach, strengthening the assumption that misleading rewards are the main source of *confusion*.

We will stress that the new approach is not only more efficient in complex search spaces with respect to samples needed and requires no baseline whatsoever, but that it also shows an increased robustness in more erratic search spaces. This is demonstrated for a problem domain with constraints introduced in the reward function by penalty terms. Here the full symmetrical sampling produces significantly fewer samples that violate the constraints.

1 Introduction

1.1 State of the Art and Problem Definition

Policy Gradient (PG) methods that explore directly in parameter space have some major advantages over standard PG methods, as described in [1, 2, 3, 4, 5, 6] and [7] and have therefore drawn a lot of attention in the last years. The basic method from the field of Parameter Exploring Policy Gradients (PEPG) [8], Policy Gradients with Parameter-based Exploration (PGPE) [1], uses two samples that are symmetric around the current hypothesis to circumvent misleading reward in *asymmetrical* reward distributed problems, gathered with the usual baseline approach. In [4] it is shown that Symmetric Sampling (SyS) is superior even to the optimal baseline, despite the fact that the variance of the gradient estimate is lower for the optimal baseline approach. The exploration parameters, however, are still updated by a baseline approach – leaving the exploration prone to asymmetric reward distributions.

While the optimal baseline improved this issue substantially [4], it is likely that removing the baseline altogether by symmetric samples with respect to the exploration parameters will again be superior. The exploration parameters are standard deviations that are bounded between zero and infinity, hence for them there exist no correct symmetric samples. We will, however, show how the exploration parameters can be sampled quasi-symmetrically. We give a transformation approximation to get quasi symmetric samples without changing the overall sampling distribution, so that the PGPE assumptions based on normal distributed samples still hold. We also will implement the optimal baseline approach for a fair comparison between baseline-free and baseline sampling. The resulting method of sampling symmetrically with respect to the problem parameters and the exploration parameters is called Super Symmetric Sampling (SupSyS) and the PGPE variant that utilizes SupSyS is named Super Symmetric PGPE (SupSymPGPE).

We will stress that SupSyS is not only more efficient in complex search spaces with respect to samples needed and requires no baseline whatsoever, but that it also shows an increased robustness in more erratic search spaces. This is demonstrated for a problem domain with constraints introduced in the reward function by penalty terms. Here SupSyS produces significantly fewer samples that violate the constraints.

1.2 Motivation

While performance and robustness is the main focus in this paper for suggesting SupSyS, the motivation for this work stems from the wish to avoid a baseline and hence avoid collecting a history over past samples. There are several reasons why this is an advantage or even mandatory in some cases:

- Lazy Evaluation: Lazy evaluation is a technique to reduce computational or time effort by evaluating only a subset of the problem. Lazy evaluation is frequently used in evolutionary algorithms [9, 10, 11]. The reward/fitness scope can change drastically while changing the degree of lazy evaluation, and even between different evaluation subsets. A baseline that averages over a substantial amount of past samples while using high degrees of lazy evaluation becomes useless. A good example from the field of robotics is the walking task. If a robot's task is to cover some distance by locomotion, it makes sense to first evaluate the behavior in short time spans, in order to distinguish behaviors that don't move at all from ones that manage to cover some distance at least. As learning progresses, the evaluation time span has to be increased further and further to distinguish behaviors that let the robot move in a straight line for a very long time from behaviors that move the robot in a slight curve or let it fall over after some time (in the humanoid case). One can divide the distance covered by the evaluation time to get some form of normalized reward, but still effects like gaining momentum at the beginning will change the reward for different evaluation times.
- Moving target and Artificial Curiosity: In some cases there is not one fixed goal of learning, but a continually evolving entity. One extreme example of such a moving target is Artifical Curiosity [12]. What all moving target problems – including artificial curiosity – have in common is that the goal changes over time, and with it the reward gained for certain behaviors changes also. In such a setting a baseline is useless.
- No cost function (only comparison operator): There are also cases where no real reward function is available since only a comparison operator is defined for the problem. One example is learning to play board games by playing against variations of the same player program, like for example the training of an artificial Go player shown in [13]. In this case, the natural definition of reward is just *better than the other solution*. Though a reward function can be constructed [13], a direct comparison is more straight forward.
- One less meta parameter: Since the damping factor for a decaying moving average baseline or the sample batch size for an optimal baseline are much less sensitive than the two step size parameters, PGPE is considered to have only 2 meta parameters. Changing the baseline parameter with increasing sampling complexity of the problem can, however, make a substantial difference in

convergence speed. With SupSymPGPE this parameter vanishes altogether and we actually arrive at a method with only 2 meta parameters.

1.3 Outline

In Section 2 we will summarize the derivation of PGPE with standard and optimal baselines. We will proceed with deriving SupSymPGPE. The section is finalized by demonstrating how SupSyS can be applied to Multimodal PGPE. Section 3 will demonstrate via experiments that sampling symmetrically also for the exploration parameters is superior in terms of samples needed and robustness compared to the original sampling approach, if confronted with search spaces with significant amounts of local optima or with regions of high slopes due to penalty terms introduced by constraints. We also show via experiments that combining SupSyS where misleading rewards are present with an optimal baseline where they are not, is virtually indistinguishable in performance to the full symmetrical approach, strengthening the assumption that misleading rewards are the main source of *confusion*. In Section 4 we will conclude the paper with some open questions for future work and a summary of the presented findings.

2 Method

In this section we derive the super-symmetric sampling method. We show how the method relates to SyS by sampling with the standard and the optimal baseline, thereby summarizing the derivations from [1] and [4] for SyS and optimal baseline sampling PGPE.

2.1 Parameter-Based Exploration

To stay concordant with the nomenclature of [1], [14] and [4], we assume a Markovian environment that produces a cumulative reward r for a fixed length *episode*, *history*, *trajectory* or *roll-out* h. In this setting, the goal of reinforcement learning is to find the optimal policy parameters $\boldsymbol{\theta}$ that maximize the agent's expected reward

$$J(\boldsymbol{\theta}) = \int_H p(h|\boldsymbol{\theta})r(h)dh. \tag{1}$$

An obvious way to maximize $J(\boldsymbol{\theta})$ is to estimate $\nabla_{\boldsymbol{\theta}} J$ and use it to carry out gradient ascent optimization. The probabilistic policy used in standard PG is replaced with a probability distribution over the parameters $\boldsymbol{\theta}$ for PGPE. The advantage of this approach is that the actions are deterministic, and an entire history can therefore be generated from a single parameter sample. This reduction in samples-per-history is what reduces the variance in the gradient estimate (see [1] for details).

We name the distribution over parameters $\boldsymbol{\rho}$, in accordance with [1]. The expected reward with a given $\boldsymbol{\rho}$ is

$$J(\boldsymbol{\rho}) = \int_{\boldsymbol{\Theta}} \int_{H} p(h, \boldsymbol{\theta}|\boldsymbol{\rho}) r(h) dh d\boldsymbol{\theta}. \tag{2}$$

Differentiating this form of the expected return with respect to $\boldsymbol{\rho}$ and applying sampling methods (first choosing $\boldsymbol{\theta}$ from $p(\boldsymbol{\theta}|\boldsymbol{\rho})$, then running the agent to generate h from $p(h|\boldsymbol{\theta})$) yields the following gradient estimator:

$$\nabla_{\boldsymbol{\rho}} J(\boldsymbol{\rho}) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\rho}} \log p(\boldsymbol{\theta}|\boldsymbol{\rho}) r(h^n). \tag{3}$$

Assuming that $\boldsymbol{\rho}$ consists of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter θ_i in $\boldsymbol{\theta}$, gives the following forms for the derivative of the characteristic eligibility $\log p(\boldsymbol{\theta}|\boldsymbol{\rho})$ with respect to μ_i and σ_i

$$\nabla_{\mu_i} \log p(\boldsymbol{\theta}|\boldsymbol{\rho}) = \frac{(\theta_i - \mu_i)}{\sigma_i^2}, \tag{4}$$

$$\nabla_{\sigma_i} \log p(\boldsymbol{\theta}|\boldsymbol{\rho}) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{5}$$

which can be substituted into Eq. (3) to approximate the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ gradients.

2.2 Sampling with a Baseline

Given enough samples, Eq. (3) will determine the reward gradient to arbitrary accuracy. However each sample requires rolling out an entire state-action history, which is expensive. Following [15], we obtain a cheaper gradient estimate by drawing a single sample $\boldsymbol{\theta}$ and comparing its reward r to a baseline reward b given e.g. by a moving average over previous samples. Intuitively, if $r > b$ we adjust $\boldsymbol{\rho}$ so as to increase the probability of $\boldsymbol{\theta}$, and $r < b$ we do the opposite. If, as in [15], we use a step size $\alpha_i = \alpha\sigma_i^2$ in the direction of positive gradient (where α is a constant) we get the following parameter update equations:

$$\begin{aligned} \Delta\mu_i &= \alpha(r - b)(\theta_i - \mu_i), \\ \Delta\sigma_i &= \alpha(r - b)\frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i}. \end{aligned} \tag{6}$$

Usually the baseline is realized as decaying or moving average baseline of the form:

$$\begin{aligned} b(n) &= \gamma r(h^{n-1}) + (1 - \gamma) b(n-1), \\ &\text{or} \\ b(n) &= \sum_{n=N-m}^{N} r(h^n)/m. \end{aligned} \tag{7}$$

In [4] it was shown recently that an optimal baseline can be found for PGPE and the algorithm converges significantly faster with an optimal baseline.

2.3 Sampling with an Optimal Baseline

Like explained above, it is known that the variance of gradient estimates can be reduced by subtracting a *baseline b*: for PGPE, a modified gradient estimate is given by

$$\nabla_{\rho}\widehat{J}^{b}(\rho) = \frac{1}{N}\sum_{n=1}^{N}(r(h^n) - b)\nabla_{\rho}\log p(\theta^n|\rho).$$

The above moving-average baseline contributes to reducing the variance of gradient estimates. However, it was shown [16] that the moving-average baseline is not optimal; the optimal baseline is, by definition, given as the minimizer of the variance of gradient estimates with respect to a baseline. Following this line, the optimal baseline for PGPE and its theoretical property are given as follows:

Let b^* be the optimal baseline for PGPE that minimizes the variance:

$$b^* := \arg\min_b \mathbf{Var}[\nabla_{\boldsymbol{\rho}}\widehat{J}^{b}(\boldsymbol{\rho})].$$

Then the following theorem gives the optimal baseline for PGPE:

Theorem 1. *The optimal baseline for PGPE is given by*

$$b^* = \frac{\mathbb{E}[r(h)\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2]}{\mathbb{E}[\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2]},$$

and the excess variance for a baseline b is given by

$$\mathbf{Var}[\nabla_{\boldsymbol{\rho}}\widehat{J}^{b}(\boldsymbol{\rho})] - \mathbf{Var}[\nabla_{\boldsymbol{\rho}}\widehat{J}^{b^*}(\boldsymbol{\rho})] = \frac{(b-b^*)^2}{N}\mathbb{E}[\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2].$$

The above theorem gives an analytic form expression of the optimal baseline for PGPE. When expected return $r(h)$ and the squared norm of characteristic eligibility $\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2$ are independent of each other, the optimal baseline is reduced to average expected return $\mathbb{E}[r(h)]$. However, the optimal baseline is generally different from the average expected return. Thus, using the average expected return as the baseline will lead to suboptimal variance reduction. The above theorem also shows that the excess variance is proportional to the squared difference of baselines $(b-b^*)^2$ and the expected squared norm of characteristic eligibility $\mathbb{E}[\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2]$, and is inverse-proportional to sample size N.

Variance reduction due to an optimal baseline is significant whenever the optimal baseline is estimated accurately. An accurately estimated optimal baseline requires a large number of samples that are statistically independent of samples used for the estimation of policy gradients. Yet such large numbers of independent samples are usually not available in practice, because gathering samples is often costly. Thus, we want to keep the number of samples as small as possible. However, when the number of samples is small, the estimated optimal baseline is not reliable enough. In order to sovle this problem, we resort to reusing the previously collected samples.

The optimal baseline given above is categorized as an *on-policy* [17] method, where data drawn from the current target policy is used to estimate the optimal baseline. On the other hand, *off-policy* methods are more flexible in the sense that a data collecting policy and the current target policy can be different. Below, we extend the on-policy optimal baseline in PGPE to an *off-policy* scenario, which allows us to reuse previously collected data in a consistent manner.

Let us consider an off-policy scenario where a data collecting policy and the current target policy are different in general. In the context of estimating the optimal baseline in PGPE, we consider two hyper-parameters, $\boldsymbol{\rho}$ for the target policy to learn and $\boldsymbol{\rho}'$ for data collection. Let us denote samples collected with hyper-parameter $\boldsymbol{\rho}'$ by D':

$$D' = \{(\boldsymbol{\theta}'_n, h'_n)\}_{n=1}^{N'} \overset{i.i.d}{\sim} p(h, \boldsymbol{\theta}|\boldsymbol{\rho}') = p(h|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\rho}').$$

If we naively use data D' to estimate optimal baseline in Theorem 1, we have an inconsistency problem.

Importance sampling [18] is a technique to systematically resolve this distribution mismatch problem. The basic idea of importance sampling is to weight samples drawn from a sampling distribution to match the target distribution, which gives a consistent gradient estimator:

$$\frac{1}{N'}\sum_{n=1}^{N'} w(\boldsymbol{\theta}'_n)r(h'_n) \overset{N'\to\infty}{\longrightarrow} \mathbb{E}[r(h)],$$

where

$$w(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta}|\boldsymbol{\rho})}{p(\boldsymbol{\theta}|\boldsymbol{\rho}')}$$

is called the *importance weight*. An intuition behind importance sampling is that if we know how "important" a sample drawn from the sampling distribution is in the target distribution, we can make adjustment by importance weighting.

Let b^*_{IW} be the optimal baseline with importance sampling, we call it importance weighted optimal baseline (IW-OB). Then the following theorem gives IW-OB:

Theorem 2. *The optimal constant baseline estimated by using data D' is given by*

$$b^*_{\mathrm{IW}} = \frac{\mathbb{E}_{p(h,\boldsymbol{\theta}|\boldsymbol{\rho}')}[r(h)w^2(\boldsymbol{\theta})\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2]}{\mathbb{E}_{p(h,\boldsymbol{\theta}|\boldsymbol{\rho}')}[w^2(\boldsymbol{\theta})\|\nabla_{\boldsymbol{\rho}}\log p(\boldsymbol{\theta}|\boldsymbol{\rho})\|^2]},$$

where $\mathbb{E}_{p(h,\boldsymbol{\theta}|\boldsymbol{\rho}')}[\cdot]$ *denotes the expectation of the function of random variables h and* $\boldsymbol{\theta}$ *with respect to* $(h, \boldsymbol{\theta}) \sim p(h, \boldsymbol{\theta}|\boldsymbol{\rho}')$.

The above theorem gives the importance weighted optimal baseline, which efficiently reuses samples collected from a sampling distribution that is different from the current target distribution.

2.4 Symmetric Sampling

While sampling with a baseline is efficient and reasonably accurate for most scenarios, especially if one uses the optimal baseline, it has several drawbacks. In particular, if the reward distribution is strongly skewed then the comparison between the sample reward and the baseline reward is misleading. A more robust gradient approximation can be found by measuring the difference in reward between two symmetric samples on either side of the current mean. That is, we pick a perturbation $\boldsymbol{\varepsilon}$ from the distribution $\mathcal{N}(0,\boldsymbol{\sigma})$, then create symmetric parameter samples $\boldsymbol{\theta}^{+} = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$ and $\boldsymbol{\theta}^{-} = \boldsymbol{\mu} - \boldsymbol{\varepsilon}$. Defining r^{+} as the reward given by $\boldsymbol{\theta}^{+}$ and r^{-} as the reward given by $\boldsymbol{\theta}^{-}$. We can insert the two samples into Eq. (3) and make use of Eq. (5) to obtain

$$\nabla_{\mu_i} J(\boldsymbol{\rho}) \approx \frac{\varepsilon_i(r^{+} - r^{-})}{2\sigma_i^2}, \tag{8}$$

which resembles the *central difference* approximation used in finite difference methods. Using the same step sizes as before gives the following update equation for the $\boldsymbol{\mu}$ terms

$$\Delta\mu_i = \frac{\alpha\varepsilon_i(r^{+} - r^{-})}{2}. \tag{9}$$

The updates for the standard deviations are more involved. As $\boldsymbol{\theta}^{+}$ and $\boldsymbol{\theta}^{-}$ are by construction equally probable under a given $\boldsymbol{\sigma}$, the difference between them cannot be used to estimate the $\boldsymbol{\sigma}$ gradient. Instead we take the mean $\frac{r^{+}+r^{-}}{2}$ of the two rewards and compare it to the baseline reward b. This approach yields

$$\Delta\sigma_i = \alpha\left(\frac{r^{+}+r^{-}}{2} - b\right)\left(\frac{\varepsilon_i^2 - \sigma_i^2}{\sigma_i}\right). \tag{10}$$

SyS removes the problem of misleading rewards, and therefore improves the $\boldsymbol{\mu}$ gradient estimates. It also improves the $\boldsymbol{\sigma}$ gradient estimates, since both samples are equally probable under the current distribution, and therefore reinforce each other as predictors of the benefits of altering $\boldsymbol{\sigma}$. Even though symmetric sampling requires twice as many histories per update, [1] and [4] have shown that it gives a considerable improvement in convergence quality and time.

2.5 Super-Symmetric Sampling

While SyS removes the misleading baseline problem for the $\boldsymbol{\mu}$ gradient estimate, the $\boldsymbol{\sigma}$ gradient still uses a baseline and is prone to this problem. On the other hand, there is no correct *symmetric* sample with respect to the standard deviation, because the standard deviation is bounded on the one *side* to 0 and is unbounded on the positive *side*. Another problem is that $\frac{2}{3}$ of the samples are on one *side* of the standard deviation and only $\frac{1}{3}$ on the other - *mirroring* the samples to the opposite side of the standard deviation in some way, would therefore deform the normal distribution so much, that it would no longer be a close enough approximation to fulfill the assumptions that lead to the PGPE update rules.

Algorithm 1. The PGPE algorithm with Reward Normalisation and Symmetric Sampling. $\boldsymbol{s}$ is P sized vector with P the number of parameters. The baseline is updated accordingly after each step. α is the learning rate or step size.

```
Initialise $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$
Initialise $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$

while TRUE do
    draw perturbation $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}\boldsymbol{\sigma}^2)$
    $\boldsymbol{\theta}^+ = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$
    $\boldsymbol{\theta}^- = \boldsymbol{\mu} - \boldsymbol{\varepsilon}$
    evaluate $r^+ = r(h(\boldsymbol{\theta}^+))$
    evaluate $r^- = r(h(\boldsymbol{\theta}^-))$
    $m := max(r, m)$
    update baseline $b$ accordingly

    $\boldsymbol{s} = [s_i]_i$ with $s_i := \frac{(\varepsilon_i)^2 - \sigma_i^2}{\sigma_i}$

    $r_\mu = \frac{(r^+ - r^-)}{2m - r^+ - r^-}$

    $r_\sigma = \frac{r^+ + r^- - 2b}{2(m-b)}$

    update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{r}_\mu \boldsymbol{\varepsilon}$
    if $\boldsymbol{r}_\sigma \geq 0$ then
        $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha \boldsymbol{r}_\sigma \boldsymbol{s}$
    end if
end while
```

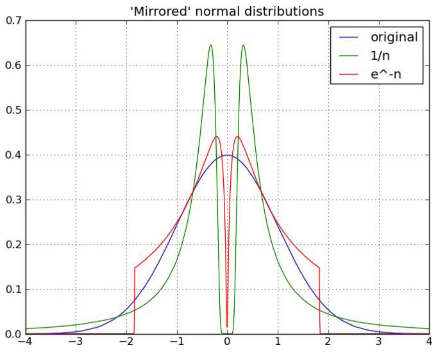


Fig. 1 Normal distribution and two first approximations of the 'mirrored' (at the median deviation) distribution

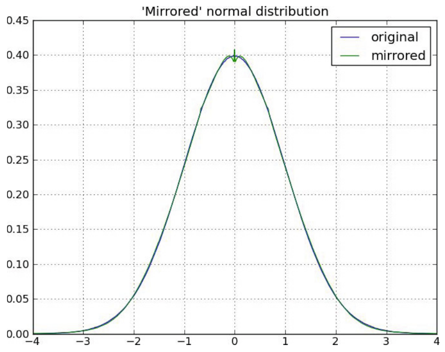


Fig. 2 Normal distribution and the final approximation of the 'mirrored' distribution

We therefore chose to define the normal distribution via the mean and the median deviation $\boldsymbol{\phi}$. The median deviation is due to the nice properties of the normal distribution simply defined by: $\boldsymbol{\phi} = 0.67449 \cdot \boldsymbol{\sigma}$. We can therefore draw samples from the new defined normal distribution: $\boldsymbol{\varepsilon} \sim \mathcal{N}_m(0, \boldsymbol{\phi})$.

The median deviation has by construction an equal amount of samples on either side and solves therefore the symmetry problem of *mirroring* samples. The update rule Eq. (9) stays unchanged while Eq. (10) is only scaled by $\frac{1}{0.67449}$ (the factor that transforms $\boldsymbol{\phi}$ in $\boldsymbol{\sigma}$) that can be substituted in $\alpha_{\boldsymbol{\sigma}}$.

While the update rules stay the same for normal distributed sampling using the median deviation (despite a larger $\alpha_{\boldsymbol{\sigma}}$), the median deviation is still also bounded on one side, so that samples cannot be simply *transfered to the other side* by subtraction. A first approximation to transfer every sample to a sample on the other side of the median deviation is the exponential:

$$\varepsilon_i^* = \phi_i \cdot sign(\varepsilon_i) \cdot e^{\frac{\phi_i - |\varepsilon_i|}{\phi_i}}. \tag{11}$$

This *mirrored* distribution has a standard deviation of 0.908 times the original standard deviation and looks like depicted in Fig. 1 (red curve).

This distribution is bounded on both sides and reaches neither absolute 0 nor infinity and is therefore a numerical stable transformation (especially important if `float32` are used, for instance when calculating on a GPU). Because of the smaller standard deviation the convergence process is biased towards smaller $\boldsymbol{\sigma}$. This is not critical for problems that converge relatively fast with a relatively high step size for the $\boldsymbol{\sigma}$ update. If the algorithm converges under a low step size the bias becomes relevant.

A closer approximation needs to be used for a general method. Because the *mirroring* cannot be solved in closed form we resort to approximation via a polynomial that can be transfered to an infinite series. We found a good approximation for *mirroring* samples by:

$$a_i = \frac{\phi_i - \mid \varepsilon_i \mid}{\phi_i}, \quad \varepsilon_i^* = sign(\varepsilon_i) \cdot \phi_i \cdot \begin{cases} e^{c_1 \frac{|a_i|^3 - |a_i|}{\log(|a_i|)} + c_2 |a_i|} & \text{if } a_i \leq 0 \\ e^{a_i} / (1. - a_i^3)^{c_3 a_i} & \text{if } a_i > 0, \end{cases} \tag{12}$$

with the following constants: $c_1 = -0.06655, c_2 = -0.9706, c_3 = 0.124$. This *mirrored* distribution has a standard deviation of 1.002 times the original standard deviation and looks like depicted in Fig. 2. Fig. 3 shows the regions of samples that are mapped into each other while generating the quasi symmetric samples.

In addition to the symmetric sample with respect to the mean hypothesis, we can also generate two quasi symmetric samples with respect to the median deviation. We named this set of four samples super symmetric samples or SupSyS samples. They allow for completely baseline free update rules, not only for the $\boldsymbol{\mu}$ update but also for the $\boldsymbol{\sigma}$ updates.

Therefore the two symmetric sample pairs are used to update $\boldsymbol{\mu}$ according to Eq. (9). $\boldsymbol{\sigma}$ is updated in a similar way by using the mean reward of each symmetric sample pair, there r^{++} is the mean reward of the original symmetric sample pair and r^{--} is the mean reward of the *mirrored* sample pair. The SupSyS update rule for the $\boldsymbol{\sigma}$ update is given by:

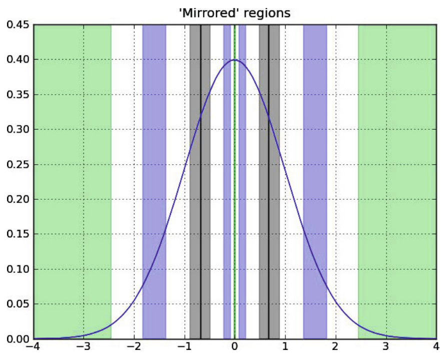


Fig. 3 Normal distribution and the regions that are mapped into each other by 'reflecting' the samples on the other side of the median deviation

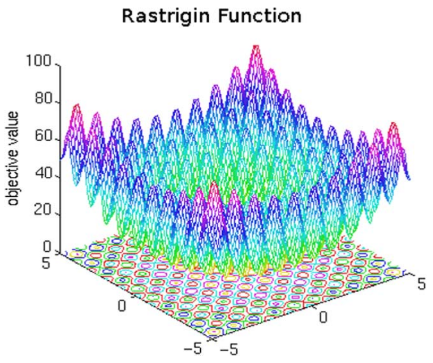


Fig. 4 Visualization of the 2D Rastrigin function

$$\Delta\sigma_i = \frac{\alpha \frac{\varepsilon_i^2-\sigma_i^2}{\sigma_i}(r^{++}-r^{--})}{2}. \tag{13}$$

Since the *mirroring* of exploration parameters is not linear, Eq. 13 becomes inaccurate if σ is changed *towards* ε_i^*. We therefore split the sigma update rule into two cases:

$$\varepsilon^\tau = \begin{cases} \varepsilon & \text{if } (r^{++}-r^{--}) \geq 0 \\ \varepsilon^* & \text{if } (r^{++}-r^{--}) < 0. \end{cases} \tag{14}$$

Eq. 13 changes to

$$\Delta\sigma_i = \frac{\alpha_\sigma \frac{\varepsilon_i^{\tau 2}-\sigma_i^2}{\sigma_i}|r^{++}-r^{--}|}{2}. \tag{15}$$

2.6 Multimodal Super-Symmetric Sampling

For completeness we show how SupSyS works with multimodal distributions over the parameters θ_i. However, the experiments in Section 3 were done with the standard unimodal version. Following [19] in MultiPGPE $\boldsymbol{\rho}$ consists of a set of mixing coefficients $\{\pi_i^k\}$, means $\{\mu_i^k\}$ and standard deviations $\{\sigma_i^k\}$ defining an independent mixture of Gaussians for each parameter θ_i:

$$p(\theta_i|\rho_i) = \sum_{k=1}^{K} \pi_i^k \mathcal{N}(\theta_i|\mu_i^k,(\sigma_i^k)^2), \tag{16}$$

Algorithm 2. The PGPE algorithm with Reward Normalisation and Super Symmetric Sampling. $\boldsymbol{T}$ is a $2 \times P$ matrix and $\boldsymbol{s}$ a vector of size P, with P the number of parameters. α is the learning rate or step size.

```
Initialise $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$
Initialise $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$

while TRUE do
  draw perturbation $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}\boldsymbol{\sigma}^2)$
  generate $\boldsymbol{\varepsilon}^*$ by mirroring $\boldsymbol{\varepsilon}$
  $\boldsymbol{\theta}^1 = \boldsymbol{\mu} + \boldsymbol{\varepsilon}$
  $\boldsymbol{\theta}^2 = \boldsymbol{\mu} - \boldsymbol{\varepsilon}$
  $\boldsymbol{\theta}^3 = \boldsymbol{\mu} + \boldsymbol{\varepsilon}^*$
  $\boldsymbol{\theta}^4 = \boldsymbol{\mu} - \boldsymbol{\varepsilon}^*$
  evaluate $r^1$ to $r^4 = r(h(\boldsymbol{\theta}^1))$ to $r(h(\boldsymbol{\theta}^4))$
  $m := max(r, m)$

  $\boldsymbol{T} = \left[ \begin{smallmatrix} \varepsilon \\ \varepsilon^* \end{smallmatrix} \right]$

  if $r^1 + r^2 \geq r^3 + r^4$ then
    $\boldsymbol{s} = [s_i]_i$ with $s_i := \frac{\varepsilon_i^2 - \sigma_i^2}{\sigma_i}$
  else
    $\boldsymbol{s} = [s_i]_i$ with $s_i := \frac{\varepsilon_i^{*2} - \sigma_i^2}{\sigma_i}$
  end if

  $\boldsymbol{r}_\mu = \left[ \frac{(r^1 - r^2)}{2m - r^1 - r^2}, \frac{(r^3 - r^4)}{2m - r^3 - r^4} \right]$
  $r_\sigma = \frac{(r^1 + r^2) - (r^3 + r^4)}{4m - r^1 - r^2 - r^3 - r^4}$

  update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{r}_\mu \boldsymbol{T}$
  update $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha\, r_\sigma \boldsymbol{s}$
end while
```

where

$$\sum_{k=1}^{K} \pi_i^k = 1, \; \pi_i^k \in [0,1] \; \forall i, k.$$

From the sum and product rules follows that sampling from a mixture distribution as defined in (16) is equivalent to first choosing a component according to the probability distribution defined by the mixing coefficients, then sampling from that component:

$$p(\theta_i|\rho_i) = \sum_{k=1}^{K} p(k|\boldsymbol{\pi}) \mathcal{N}(\theta_i|\mu_i^k, (\sigma_i^k)^2),$$

if we define $\pi_k = p(k|\boldsymbol{\pi})$ as the prior probability of picking the k^{th} Gaussian.

This suggests what is called simplified MultiPGPE and leads to the following gradient calculations: First pick k with probability π_i^k, then set the mixing coefficients to:

$$l_i^k = 1, \quad l_i^{k'} = 0 \; \forall k' \neq k.$$

If we again use a step size $\alpha_i = \alpha\sigma_i^2$ in the direction of positive gradient (where α is a constant) the following parameter update equations for MultiPGPE are obtained:

$$\begin{aligned}\Delta\pi_i^k &= \alpha_\pi(r-b)l_i^k,\\ \Delta\mu_i^k &= \alpha_\mu(r-b)l_i^k(\theta_i-\mu_i^k),\\ \Delta\sigma_i^k &= \alpha_\sigma(r-b)l_i^k\frac{(\theta_i-\mu_i^k)^2-(\sigma_i^k)^2}{\sigma_i^k}.\end{aligned} \tag{17}$$

For the simplified version of MultiPGPE we also can use SupSyS. That is, we pick again a perturbation $\boldsymbol{\varepsilon}$ from the distribution $\mathcal{N}(\mathbf{0},\boldsymbol{\sigma})$, then create symmetric parameter samples

$$\boldsymbol{\theta}^+ = \boldsymbol{\mu}+\boldsymbol{\varepsilon}, \quad \boldsymbol{\theta}^- = \boldsymbol{\mu}-\boldsymbol{\varepsilon}.$$

Again defining r^+ as the reward given by $\boldsymbol{\theta}^+$ and r^- as the reward given by $\boldsymbol{\theta}^-$, we obtain

$$\nabla_{\mu_i^k} J(\boldsymbol{\rho}) \approx \frac{\varepsilon_i(r^+-r^-)}{2(\sigma_i^k)^2}, \tag{18}$$

which resembles again the *central difference* approximation used in finite difference methods. Using the same step sizes as before gives the following update equation for the μ terms

$$\Delta\mu_i^k = \frac{\alpha_\mu\varepsilon_i(r^+-r^-)}{2}. \tag{19}$$

By generating super symmetric samples with ε^* following Eq. 12, we receive the mean reward r^{++} for the ε samples and r^{--} for the ε^* samples and obtain the update equations for the σ terms under consideration of Eq. 14

$$\Delta\sigma_i^k = \frac{\alpha_\sigma\frac{\varepsilon_i^{\tau 2}-\sigma_i^{k2}}{\sigma_i^k}|r^{++}-r^{--}|}{2}. \tag{20}$$

Note again that SupSyS is only possible for the simplified version of MultiPGPE. This is the reason that no mixing coefficient l_i^k appears in above equations because it is assumed to be 1 (or 0). For MultiPGPE a baseline is still required for the $\boldsymbol{\pi}$ updates — so it is only applicable if a (useful) baseline can be obtained.

2.7 Reward Normalisation for SupSymPGPE

It is important to make the step size independent from the (possibly unknown) scale of the rewards by introducing a normalisation term. This is standard for all PGPE variants. However, for SupSymPGPE it contradicts with the motivation of using no

Algorithm 3. The simplified MultiPGPE algorithm with SupSyS and Reward Normalisation. $\boldsymbol{T}$ is a $2 \times P$ matrix and $\boldsymbol{s}$ a vector of size P, with P the number of parameters. α is the learning rate or step size.

```
Initialise $\boldsymbol{\pi}$ to $\boldsymbol{\pi}_{\text{init}}$
Initialise $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$
Initialise $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$

while TRUE do
    draw Gaussians $\boldsymbol{k}^n \sim p(\boldsymbol{k}|\boldsymbol{\pi})$
    draw perturbation $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}\boldsymbol{\sigma}_{\boldsymbol{k}}^2)$
    generate $\boldsymbol{\varepsilon}^*$ by mirroring $\boldsymbol{\varepsilon}$
    $\boldsymbol{\theta}^1 = \boldsymbol{\mu}_{\boldsymbol{k}} + \boldsymbol{\varepsilon}$
    $\boldsymbol{\theta}^2 = \boldsymbol{\mu}_{\boldsymbol{k}} - \boldsymbol{\varepsilon}$
    $\boldsymbol{\theta}^3 = \boldsymbol{\mu}_{\boldsymbol{k}} + \boldsymbol{\varepsilon}^*$
    $\boldsymbol{\theta}^4 = \boldsymbol{\mu}_{\boldsymbol{k}} - \boldsymbol{\varepsilon}^*$
    evaluate $r^1$ to $r^4 = r(h(\boldsymbol{\theta}^1))$ to $r(h(\boldsymbol{\theta}^4))$
    $m := max(r, m)$
    update baseline $b$ accordingly

    $\boldsymbol{T} = \left[\begin{smallmatrix}\varepsilon \\ \varepsilon^*\end{smallmatrix}\right]$

    if $r^1 + r^2 \geq r^3 + r^4$ then
        $\boldsymbol{s} = [s_i]_i$ with $s_i := \frac{\varepsilon_i^2 - \sigma_{i,k}^2}{\sigma_{i,k}}$
    else
        $\boldsymbol{s} = [s_i]_i$ with $s_i := \frac{\varepsilon_i^{*2} - \sigma_{i,k}^2}{\sigma_{i,k}}$
    end if

    $\boldsymbol{r}_p = \left[\frac{r^1+r^2-2b}{2(m-b)}, \frac{r^3+r^4-2b}{2(m-b)}\right]$
    $\boldsymbol{r}_\mu = \left[\frac{(r^1-r^2)}{2m-r^1-r^2}, \frac{(r^3-r^4)}{2m-r^3-r^4}\right]$
    $r_\sigma = \frac{(r^1+r^2)-(r^3+r^4)}{4m-r^1-r^2-r^3-r^4}$

    update $\boldsymbol{\pi}_{\boldsymbol{k}} = \boldsymbol{\pi}_{\boldsymbol{k}} + \alpha \sum \boldsymbol{r}_p$
    Normalize all $\boldsymbol{\pi}_i$ to $||1||$
    update $\boldsymbol{\mu}_{\boldsymbol{k}} = \boldsymbol{\mu}_{\boldsymbol{k}} + \alpha \boldsymbol{r}_\mu \boldsymbol{T}$
    update $\boldsymbol{\sigma}_{\boldsymbol{k}} = \boldsymbol{\sigma}_{\boldsymbol{k}} + \alpha r_\sigma \boldsymbol{s}$
end while
```

baseline, that is to have no need for a sample history. Remembering the best so far received reward requires both a history of rewards and the prerequisite that the rewards stay the same over time. In cases like described in 1.2 reward normalization is not possible. In cases where SupSymPGPE is used for performance reasons, reward normalization is still an important step.

Let m be the maximum reward the agent can receive, if this is known, or the maximum reward received so far if it is not. We normalise the $\boldsymbol{\mu}$ updates by dividing them by the difference between m and the mean reward of the respective symmetric samples, and we normalise the $\boldsymbol{\sigma}$ updates by dividing by the difference between m and the mean of all four samples, yielding

$$\Delta\mu_i = \alpha_\mu \frac{\varepsilon_i(r^+ - r^-)}{2m - r^+ - r^-},$$
$$\Delta\sigma_i = \alpha_\sigma \frac{\frac{\varepsilon_i^{\tau 2} - \sigma_i^2}{\sigma_i}(r^{++} - r^{--})}{2m - r^{++} - r^{--}}. \tag{21}$$

3 Experiments and Results

We use the square function as search space instance with no local optima and the Rastrigin function (see Fig. 4) as search space with exponentially many local optima, to test the different behavior of SupSym- and SyS-PGPE. We also show the performance of both methods on a real world example that optimizes installed capacities of renewable energies in a distributed energy system. This experiment shows how both methods cope with constraints that are implemented as penalty terms in the reward function and resemble regions with steep slopes or *cliffs* in the search space. The two meta-parameters connected with SyS-PGPE as well as with SupSymPGPE, namely the step sizes for the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ updates, were optimized for every experiment via a grid search. The Figures 5 to 9 show the means and standard

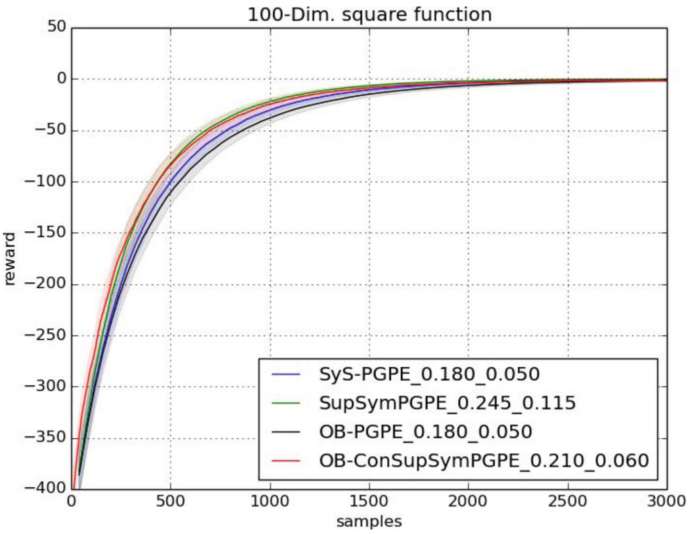


Fig. 5 Convergence plots of PGPE (SyS-PGPE), optimal baseline PGPE (OB-PGPE), conditional SupSymPGPE with optimal baseline (OB-ConSupSymPGPE) and SupSymPGPE on the 100 dimensional square function. The mean and standard deviation of 200 independent runs are shown.

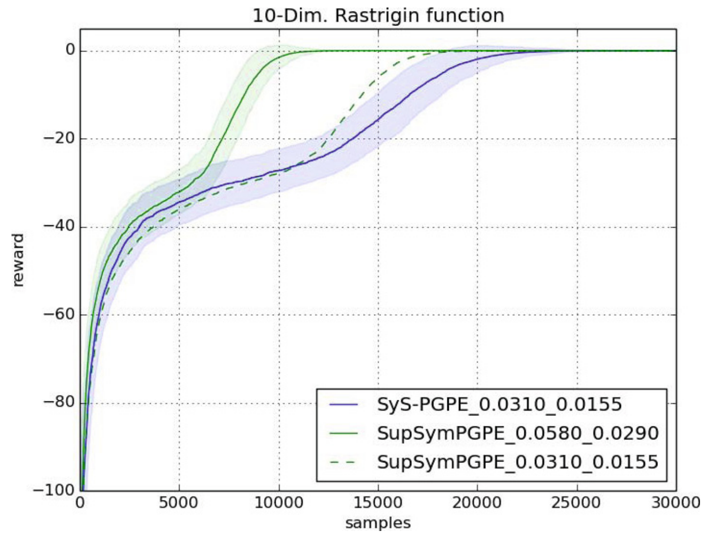


Fig. 6 Convergence plots of PGPE and SupSymPGPE on the 10 dimensional Rastrigin function. The mean and standard deviation of 200 independent runs are shown.

deviations of 200 independent runs each. Figure 13 shows the means and standard deviations of 20 independent runs.

3.1 Square Function

It can be seen in Fig. 5 that for a search space with no local optima SupSymPGPE shows no advantage over standard SyS-PGPE. However, despite using 4 samples per update the performance is also not reduced by using SupSymPGPE — the two methods become merely equivalent. Also the use of the optimal baseline makes no apparent difference.

3.2 Rastrigin Function

The situation changes drastically if the Rastrigin function is used as test function. Not only needs SupSymPGPE about half the samples compared to PGPE, the effect seems also to become stronger the higher dimensional the search space gets (see Fig. 6 to Fig. 9).

We also added SupSymPGPE plots with the meta parameters optimal (less greedy) for SyS-PGPE to show that the effect is not only due to the (optimal) more aggressive meta parameters. This runs were also more efficient than for PGPE, the effect was however not so distinct.

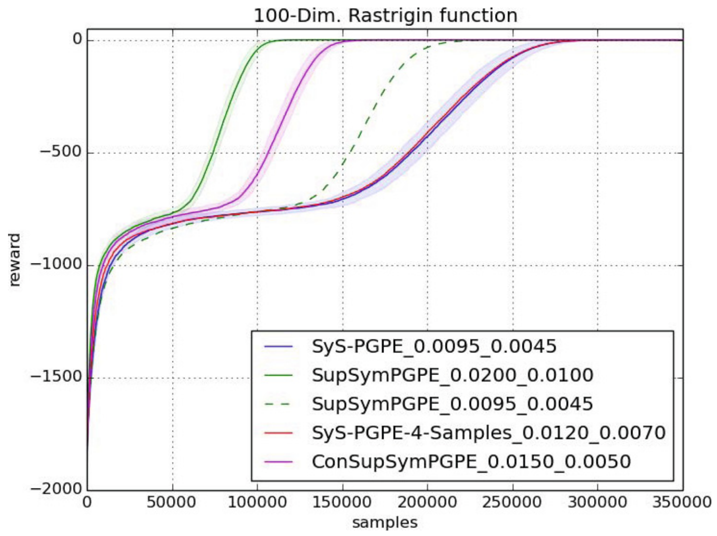


Fig. 7 Convergence plots of PGPE, PGPE with 4 samples (SyS-PGPE-4-Sampels), conditional SupSymPGPE (ConSupSymPGPE) and SupSymPGPE on the 100 dimensional Rastrigin function. The mean and standard deviation of 200 independent runs are shown.

In Fig. 7 we also show a standard PGPE experiment with 4 samples (2 SyS samples — SyS-PGPE-4-Sample) instead of 2 to show that the improved performance is not due to the different samples per update.

Fig. 7 additionally shows an experiment there symmetric samples are only drawn if the first sample(s) result in worse reward than a decaying average (ConSupSymPGPE) or optimal baseline (OB-ConSupSymPGPE). We will call this kind of sampling conditional sampling. The intuitive idea behind symmetric samples was initially that changing the parameters *away* from the current hypotheses if the sample resulted in lower than average reward may move the mean hypothesis still in a worse region of the parameter space. Search spaces like the one given in the Rastrigin function can visualize this problem (see Fig. 11). For ConSupSymPGPE one Sample is drawn. If the reward is larger than the baseline then an update is done immediately. If not, a symmetric sample is drawn. If the mean reward connected with both samples better than the baseline an SyS-PGPE update is done. If also this mean reward is worse than the baseline, a full SupSymPGPE update with 2 additional SyS samples is performed. As can be seen in Fig. 7 the performance for ConSupSymPGPE is worse by some degree — the difference is however small enough that the optimal baseline approach becomes challenging for SupSymPGPE, like can be seen in Fig. 8. This last result gives a clue that the improvement in performance of SupSymPGPE is indeed due to misleading rewards due to asymmetry in the search space.

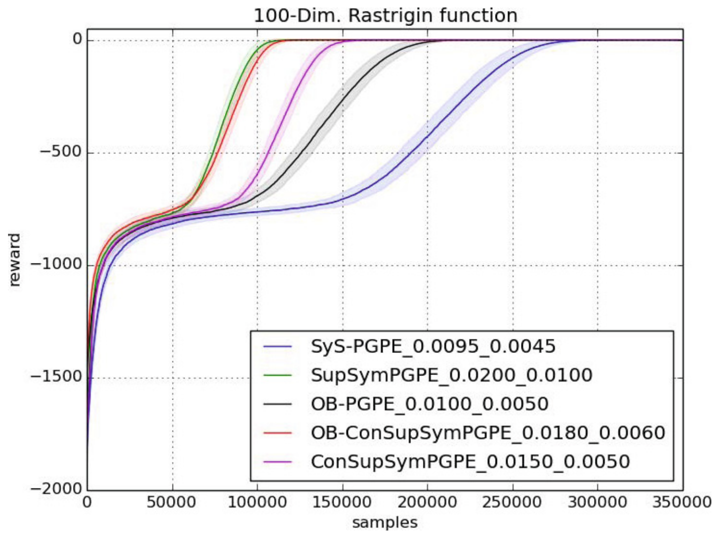


Fig. 8 Convergence plots of PGPE (SyS-PGPE), optimal baseline PGPE (OB-PGPE), conditional SupSymPGPE with optimal baseline (OB-ConSupSymPGPE) and with standard baseline (ConSupSymPGPE) and SupSymPGPE on the 100 dimensional Rastrigin function. The mean and standard deviation of 200 independent runs are shown.

The optimal meta-parameters are an exponential function of the search space dimension, like to expect, so that we observe a line in the *loglog*-plot of Fig. 10. For SupSyS-PGPE the meta-parameters are about 2 times larger than for SyS-PGPE. This is partly because SupSymPGPE uses four samples per update instead of two. But the optimal meta-parameters are also larger than for the SyS-PGPE-4-Sample experiment so that the symmetric nature of the four SupSyS samples obviously brings additional stability in the gradient estimate than a pure averaging over 4 samples would.

3.3 Renewable Energy Problem

We also used PGPE for optimizing distributed energy systems that consist of a range of electrical power producers including conventional and renewable sources as well as different consumers, long and short term storages, converters and power sinks. The whole system is included in a simulation framework for virtual- and hybrid power plants called P^{2}IONEER. An overview of the included technologies and how they interact is shown in Fig. 12. The goal of optimization is to achieve a certain fraction of renewable energies (up to 100%) with the lowest Levelised Cost of Electricity (LCoE). Time-series for wind, photovoltaics (PV) and water power production from real sites are feed into the system as well as demand and temperature profiles and are simulated together with on site availability of biomass and storage capacities for natural gas and the like.

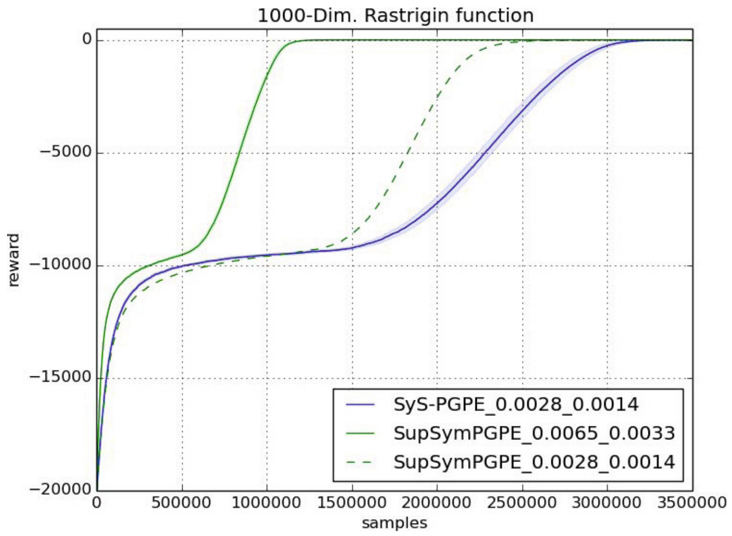


Fig. 9 Convergence plots of PGPE and SupSymPGPE on the 1000 dimensional Rastrigin function. The mean and standard deviation of 200 independent runs are shown.

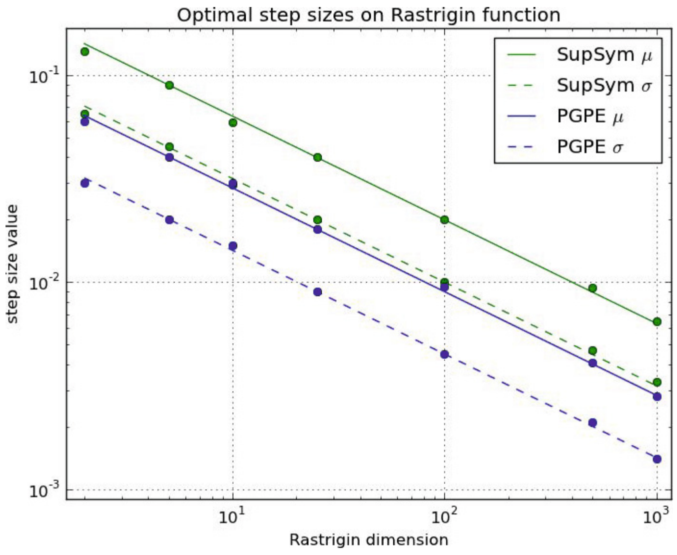


Fig. 10 Optimal meta-parameters for the multi-dimensional Rastrigin function for PGPE and SupSymPGPE

This optimization problem is extremely complex since most of the free parameters are highly correlated and intertwined. Changing for example only the installed amount of wind power plants will not improve the LCoE if the PV and

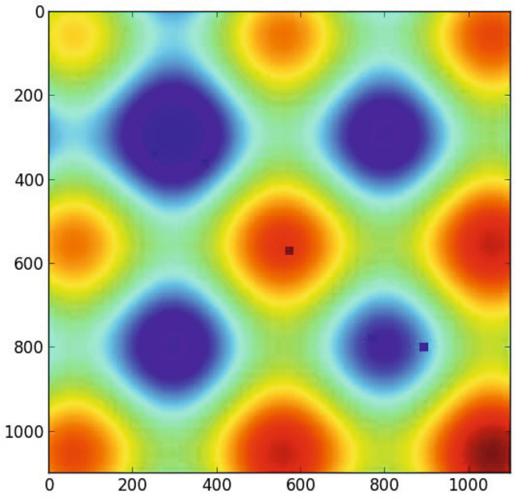


Fig. 11 Visualization of 4 super symmetric sample (blue squares) around the mean hypothesis (red square) in a zoom view of the Rastrigin function. The first sample (rightmost) is worse than the current baseline and would lead to an update in a direction that is worse in the baseline-PGPE case. Due to the symmetric sample (leftmost) this *error in judgment* is compensated. This symmetric sample pair would lead to a decrease in the exploration parameters in both dimensions that would increase the probability to remain in the local optima. The second pair of symmetric samples is correcting that and even increases the exploration parameters slightly.

storage capacities are not also adjusted appropriately. But what makes the problem interesting for SupSymPGPE is that it shows several constraints that are included as penalty terms in the reward function resembling very steep slopes in the search space. While testing the performance of SupSymPGPE compared to standard PGPE on this problem it became apparent that SupSymPGPE behaves much better in this kind of setting. SupSymPGPE can cope with the constraints introduced in the reward function much better and not only converges to the optimal LCoE with much less samples but also does that much smoother by avoiding samples that violate the given constraints much more efficient. Figure 13 shows the results on this optimization problem. Again we also made one experiment for SupSymPGPE with the optimal, less greedy, meta-parameters for PGPE to show that the effect is not solely due to the fact that SupSymPGPE allows for higher step sizes. Please note that the convergence curves for SupSymPGPE are also much smother than for PGPE that stems from PGPE generating more samples that violate the given constraints.

The experiments were done with wind, PV and water time series from a region in central Germany. The goal is a 100% renewable power supply for the region for the years 2011 to 2012. All constraints and surrounding conditions have been investigated and were used from that in reality existing region.

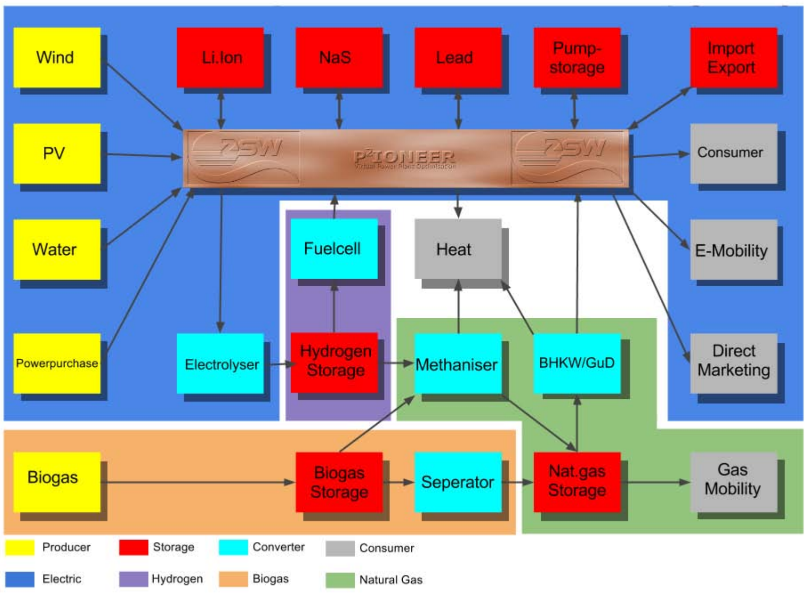


Fig. 12 The available technologies and the energy flow in P^{2}IONEER

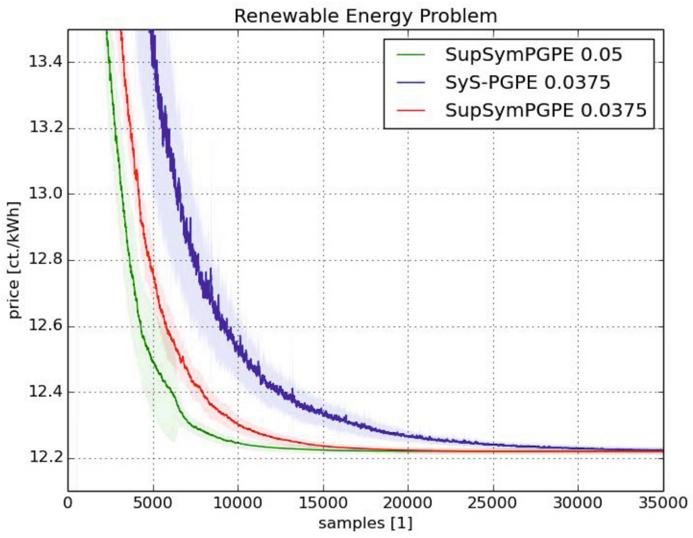


Fig. 13 Convergence plots of PGPE and SupSymPGPE on the Renewable Energy problem. The mean and standard deviation of 20 independent runs are shown.

4 Conclusions and Future Work

We introduced SupSymPGPE, a completely baseline free PGPE that uses quasi-symmetric samples wrt. the exploration parameters. We showed that on the Rastrigin function, as example for a test function with exponentially many local optima, this novel method is clearly superior to standard SyS-PGPE and that both methods become equivalent in performance if the search space lack *distracting* local optima. The performance was tested for the standard and the optimal baseline. We also showed an example problem with several constraints introduced as penalty terms to the reward function there SupSymPGPE also outperformed standard PGPE clearly. In all experiments so far conducted (also ones not listed here) SupSymPGPE was never less efficient than standard PGPE. The most compelling property is, however, that SupSymPGPE is much more robust if search spaces become erratic.

For future work we want to highlight that SupSymPGPE can be easily combined with other extensions of PGPE. Multi-modal PGPE [19] can be equipped straight forward with SupSyS, like shown in 2.6. Experimental results on this is considered as interesting future work. Also the natural gradient used for PGPE in [3] can be defined over the SupSyS gradient instead over the vanilla gradient. While it is hard to imagine a sampling scheme that is symmetric to full covariance samples one can easily genereate super symmetric samples in the rotated space defined by a covariance matrix.

While importance sampling is a very effective method to reduce the needed evaluations, like shown by [5] it cannot be applied to SupSymPGPE directly. It can however be used if SupSymPGPE is used for performance reasons and a baseline is available by adding standard PGPE updates for the sample history and SupSymPGPE updates for the direct samples. Another alternative would be to use Importance Mixing that was used for the same reasons in [20].

Finally a big open point for future work is the validation of the mere theoretical findings also for robotic tasks, for SupSymPGPE and its combination with other PGPE extensions. Since in robotic tasks an experienced experimentator will define the reward function well (reward shaping), it is expected that the reward landscape tends to be closer to the square function than to the Rastrigin function and constraints may be avoided or introduced in a less harmful way. So we guess that the impact on robotic behavior learning will be less drastic than in the presented cases.

Acknowledgments. This work was funded by the Zentrum für Sonnenenergie- und Wasserstoff- Forschung and MEXT Scholarship.

References

1. Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. Neural Networks 23(4), 551–559 (2010)
2. Rückstieß, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., Schmidhuber, J.: Exploring parameter space in reinforcement learning. Paladyn. Journal of Behavioral Robotics 1(1), 14–24 (2010)
3. Miyamae, A., Nagata, Y., Ono, I.: Natural Policy Gradient Methods with Parameter-based Exploration for Control Tasks. In: NIPS, pp. 1–9 (2010)

4. Zhao, T., Hachiya, H., Niu, G., Sugiyama, M.: Analysis and improvement of policy gradient estimation. Neural networks: the Official Journal of the International Neural Network Society, 1–30 (October 2011)
5. Zhao, T., Hachiya, H., Tangkaratt, V., Morimoto, J., Sugiyama, M.: Efficient sample reuse in policy gradients with parameter-based exploration. arXiv preprint arXiv:1301.3966 (2013)
6. Stulp, F., Sigaud, O.: Path integral policy improvement with covariance matrix adaptation. arXiv preprint arXiv:1206.4621 (2012)
7. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: Evolutionary Computation, CEC 2008, pp. 3381–3387. IEEE (2008)
8. Sehnke, F.: Parameter Exploring Policy Gradients and their Implications. PhD thesis, München, Technische Universität München, Diss., 2012 (2012)
9. Henderson, P., Morris Jr., J.H.: A lazy evaluator. In: Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages, pp. 95–103. ACM (1976)
10. Heinemann, P., Streichert, F., Sehnke, F., Zell, A.: Automatic calibration of camera to world mapping in robocup using evolutionary algorithms. In: IEEE Congress on Evolutionary Computation, CEC 2006, pp. 1316–1323. IEEE (2006)
11. Heinemann, P., Sehnke, F., Streichert, F., Zell, A.: An automatic approach to online color training in robocup environments. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4880–4885. IEEE (2006)
12. Schmidhuber, J.: Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. Connection Science 18(2), 173–187 (2006)
13. Grüttner, M., Sehnke, F., Schaul, T., Schmidhuber, J.: Multi-dimensional deep memory atari-go players for parameter exploring policy gradients. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) ICANN 2010, Part II. LNCS, vol. 6353, pp. 114–123. Springer, Heidelberg (2010)
14. Sehnke, F.: Efficient baseline-free sampling in parameter exploring policy gradients: Super symmetric pgpe. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 130–137. Springer, Heidelberg (2013)
15. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256 (1992)
16. Greensmith, E., Bartlett, P.L., Baxter, J.: Variance reduction techniques for gradient estimates in reinforcement learning. Journal of Machine Learning Research 5, 1471–1530 (2004)
17. Sutton, R.S., Barto, G.A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
18. Fishman, G.S.: Monte Carlo: Concepts, Algorithms, and Applications. Springer, Berlin (1996)
19. Sehnke, F., Graves, A., Osendorfer, C., Schmidhuber, J.: Multimodal parameter-exploring policy gradients. In: 2010 Ninth International Conference on Machine Learning and Applications (ICMLA), pp. 113–118. IEEE (2010)
20. Sun, Y., Wierstra, D., Schaul, T., Schmidhuber, J.: Efficient natural evolution strategies. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 539–546. ACM (2009)

Sparse Approximations to Value Functions in Reinforcement Learning

Hunor S. Jakab and Lehel Csató

Abstract. We present a novel sparsification and value function approximation method for on-line reinforcement learning in continuous state and action spaces. Our approach is based on the kernel least squares temporal difference learning algorithm. We derive a recursive version and enhance the algorithm with a new sparsification mechanism based on the topology obtained from proximity graphs. The sparsification mechanism – necessary to speed up the computations – favors data-points minimizing the divergence of the target-function gradient, thereby also considering the *shape* of the target function. The performance of our sparsification and approximation method is tested on a standard benchmark RL problem and comparisons with existing approaches are provided.

1 Introduction

Reinforcement learning addresses the problem of learning optimal decision making strategies (policies) for diverse control problems. RL puts special emphasis on the autonomous nature of the learning process. The learning agent develops an efficient policy to reach a predefined goal, without external inference, the only information source being its state and the reward signal that it receives from the environment. Whilst the definition of autonomy is difficult, the design of a *generic* autonomous learning "agent" is based on the idea that the agent explores its environment and aims to collect "rewards" – i.e. to exploit too – during exploration [22]. Given a reinforcement learning algorithm, one needs to build the reward function for it to be applied to a given application domain.

Approximate reinforcement learning (RL) methods are algorithms dealing with real-world problems that are characterized by continuous, high dimensional

Hunor S. Jakab · Lehel Csató
Babeş-Bolyai University, Faculty of Mathematics and Computer Science
e-mail: {jakabh,csatol}@cs.ubbcluj.ro

state-action spaces and noisy measurements. The purpose of this chapter is to investigate the problem of accurately estimating value functions for a given policy in continuous RL problems, also called *policy evaluation*[22].

Policy evaluation is the process of estimating the utility of a state or state-action pair when the learning agent starts from that state (state-action pair) and follows a fixed strategy (policy) for an infinite number of steps. The function that maps states to utility values is called a value function and it needs to be represented by a function approximation framework. Many reinforcement learning methods rely on the accurate approximation of value functions, which can be used to derive greedy policies that choose the action which leads to the state with highest utility value. Thus the accuracy of value functions around decision boundaries (regions where the utility values change rapidly) is especially important. Traditionally the representation of value functions on continuous domains has been treated using linear maps with non-linear features (*ex. using different order polinomial, fourier or wavelet basis*)[11], neural networks [16], non-parametric approximators [7] etc. To improve the accuracy of the approximated value functions a large variety of algorithms have been proposed.

In this article we focus on linear least-squares algorithms for temporal difference learning [2]. These algorithms similarly to most RL algorithms, require a lot of hand-tuning to work in different application domains. One of the major research areas within RL is to eliminate the need for hand-tuning and to develop methods which can be applied generically to a large variety of problems. A popular way to address this design issue is by applying a nonlinear extension of the algorithm using "kernelization". Kernelization in turn raises the problem of complexity, over-fitting, and increased computational cost. To avoid these problems, different sparsification mechanisms are employed which are meant to eliminate non-essential training-data based on a given selection criteria. The quality of the sparsification procedure influences the accuracy and the generalization capability of the function approximator.

The ability of an RL algorithm to operate on-line, to acquire training-data sequentially by interacting with its environment, is crucial. In this setting, the training data acquired during environment interaction presents some spatio-temporal characteristics which in our opinion can be exploited to develop a superior approximation framework. To do this we introduce a proximity-graph based sparsification mechanism which is based on the on-line construction of a proximity-graph that captures the temporal correlation of the training data. The resulting graph is a coarse representation of the manifold upon which the training data lies, and it allows the sparsification to fine-tune itself by increasing the density of *support points*[1] in the areas where the function changes rapidly. At the same time our algorithm reduces the density of support-points in areas where the target function hardly changes, keeping computational costs down. We apply the sparsification mechanism to the kernel least squares temporal difference learning (KLSTD) [29] algorithm. We also introduce a recursive version of KLSTD that is much better suited to the on-line learning scenario frequently seen in reinforcement learning.

[1] Support points are similar to the support points in Parzen-windowing or in vector-quantization summarizing in a few – weighted – samples a large amount of input location.

The rest of this paper is structured as follows: In Section 2 we introduce some of the basic notions related to reinforcement learning and the notation which we will be using throughout the rest of the paper. Section 3 surveys some of the recent value function approximation methods from the literature. In section 4 we introduce the kernel least squares algorithm for value approximation, while in section 4.1 we present our first major contribution, a recursive version of KLSTD.The sparsification problem is introduced in Section 5, and in Section 5.1 the second major contribution of the paper, the laplacian sparsification mechanism is presented. Section 6 deals with the problem of iteratively constructing proximity graphs to be used in our laplacian sparsification mechanism. The complete value approximation algorithm based on our recursive version of KLSTD together with the laplacian sparsification method is presented in detail in Section 7. Section 8 presents the experimental results and Section 9 concludes the paper.

2 Notation and Background

We introduce the notation we will be using and sketch the background material required to understand the proposed methods. We mention first that in RL data results from interacting with the environment and consequently the successive states of the agent, the actions taken in those states and the corresponding rewards define the training data. Moreover – bringing the method closer to applicability – we assume that data is acquired on-line. The size of the training data therefore is not fixed and it expands continuously as the interaction process takes place.

To describe the environment where the learning process takes place we use the formalism of Markov decision processes (MDP) [15], commonly used for modeling reinforcement learning problems:

An MDP is a quadruple $M(S,A,P,R)$, where S is the (possibly infinite) set of states, A is the set of actions, $P(s'|s,a) : S \times S \times A \to [0,1]$ describes the usually unknown transition probabilities, and $R(s,a) : S \times A \to R$ is the instantaneous reward function defining the feedback to the learner.

The decision making mechanism is modeled with the help of an action selection *policy*: $\pi : S \times A \to [0,1]$. Here π is the conditional probability distribution – $\pi(a|s)$ – of taking action a while being in the state s. In reinforcement learning the goal is to find the *optimal policy* π^*, that maximizes the expected discounted cumulative reward received by the learner:

$$\pi^* = \arg\max_{\pi} E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$$

An important element of many RL algorithm is a representation of the *utility* of the state or state-action pairs as value functions $V_\pi : S \to \mathbb{R}$ or action-value functions $Q_\pi : S \times A \to \mathbb{R}$:

$$V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right]; \quad Q_\pi(s,a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s, a_0 = a\right]$$

Value functions express the expected long term discounted reward received when starting from a given state or state-action pair and following a given policy π. The majority of reinforcement learning algorithms can be included in two main categories: value based and model-based methods. Model-based algorithms [7], [6] approximate both the system dynamics and the state-value function and apply dynamic programming for the derivation of the optimal policy. In both of these categories the accurate and consistent estimation of state or state-action values plays an essential role.

Since this paper focuses on value approximation, from now on we consider the action selection policy to be fixed. This restriction gives rise to a so-called Markov Reward process [23] which we will refer to as MRP. Throughout the rest of this paper for ease of exposition we will omit the policy from the notation. For better readability we will use state value functions $V(\cdot)$ for the derivation of our algorithms, since their extension to state-action value functions is straightforward.

3 Related Work in Value Approximation

The approximation of value functions can be accomplished by different function approximation architectures. There have been attempts to use neural networks [16], hierarchical function approximation [26], non-parametric methods [19],[20],[9].

Estimation and approximation of value functions is easiest with a linear model: the value is a linear function of the state. These models are too simple to be applied for RL problems. A non-linear extension is to consider models that are linear in their parameters but *non-linearly* map the inputs to the output space, called *linear-in weight models* [21], or *generalized linear models* [13]. Linear regression models with non-linear features have been preferred for approximate policy evaluation, due to their good convergence properties and relative ease of analysis. The main drawback of linear-in-weight regression models is the difficulty in obtaining *good features* for each problem. One way to reduce the hand-tuning required is to decouple the approximation of value functions or state-transition dynamics from the construction of problem-specific feature functions which transform the input data into a more interpretable form. The proto-value function framework presented in [12] achieves this by using diffusion wavelets and the eigenvectors of a graph laplacian for the automatic construction of feature functions. Another popular approach is to use kernel methods for this purpose [18] by formulating the algorithms in such a way that feature functions only appear in dot product form. Gaussian processes are a good example of this, and have been successfully used in a number of references for approximating both value functions and system transition dynamics [6],[7]. Another class of value approximation algorithms is based on the least squares minimization principle and use linear architectures with updates known from temporal difference learning [2],[1]. Least squares based methods are believed to have better sample efficiency, however the expressive power of linear architectures is largely determined by the quality of the feature functions that they use. The learning algorithms from this class can also be kernelised as seen in [29]. An overview of kernel-based value approximation frameworks can be found in [24].

4 Kernel Least Squares Value Approximation

LSTD is based on a generalized linear approximation to the value function using a set of predefined feature vectors:

$$\tilde{V}(s) = \phi(s)^T \mathbf{w}^H \quad , \text{where} \quad \phi(s) = [\phi_1(s), \dots, \phi_d(s)]^T. \tag{1}$$

Here $\mathbf{w}^H = [w_1, \dots, w_d]^T$ is the vector containing the feature weights, and d is the dimension of the feature space. Notice that $\mathbf{w}^H \in \mathbb{R}^d$, *i.e.* it has to match the dimension of the feature space where the input s is projected and superscript H signifies this fact. In temporal difference we incrementally update the value function based on the *temporal difference error* [22]:

$$\delta_{t+1} = R_{t+1} + \gamma \tilde{V}_t(s_{t+1}) - \tilde{V}_t(s_t) = R_{t+1} - (\phi(s_t) - \gamma\phi(s_{t+1}))^T \mathbf{w}_t^H \tag{2}$$

and the parameter update – using the learning parameter α_t – is the following:

$$\mathbf{w}_{t+1}^H \leftarrow \mathbf{w}_t^H + \alpha_t \delta_{t+1} \phi(s_t).$$

where t is time and, since each datum is new, it is also the sample number. In the limit of infinite time t, the algorithm leads to a converged vector $\mathbf{w}^H$ that satisfies $E[\phi(s_t)\delta_{t+1}] = 0$ [23]. We are looking at its equilibrium value, by replacing the theoretical average with the sample average, to obtain the following equation:

$$\frac{1}{n} \sum_{t=0}^{n} \phi(s_t)\delta_{t+1} = 0 \tag{3}$$

Substituting δ_{t+1} from eq. (2) – and dropping n, the sample size – we obtain:

$$\left(\sum_{t=0}^{n} \phi(s_t) (\phi(s_t) - \gamma\phi(s_{t+1}))^T \right) \mathbf{w} = \sum_{t=0}^{n} \phi(s_t) R_t \tag{4}$$

If we use the following notation:

$$\tilde{A}^H = \sum_{t=0}^{n} \phi(s_t) (\phi(s_t) - \gamma\phi(s_{t+1}))^T , \quad \tilde{b}^H = \sum_{t=0}^{n} \phi(s_t) R_t,$$

Then the equation becomes: $\tilde{A}^H \mathbf{w}^H = \tilde{b}^H$, a linear problem. For this system we have the following properties:

- The matrix $\tilde{A}^H$ is a sum of individual outer products, therefore it can be calculated incrementally;
- Each component in the sum is a square matrix with the size of the feature space;
- When $\tilde{A}^H$ is invertible, there is a direct solution for $\mathbf{w}^H$, see *e.g.* [2].

The problem with the solution presented above is that manually constructing suitable feature representations for each problem-domain is time-consuming and error prone. To alleviate this drawback, a non-linear extension via "kernelisation" of the above algorithm has been introduced in [28], briefly sketched in what follows. To eliminate the direct projection into the feature space, we assume that the projection is into a *reproducing kernel Hilbert space* [25, 18]. The solutions to the linear systems can then be written as a linear combination of the feature images of the inputs, *i.e.* $\mathbf{w}^H \overset{\text{def}}{=} \sum_{t=1}^{n} w_i \phi(s_i)$. This is equivalent with a re-parametrization of the problem from eq. (4), where (1) we are looking for $\mathbf{w} = [w_1, \ldots, w_n]$ as coefficients solving the problem, and (2) we express all equations in terms of a *dot product* $\phi(s_i)^T \phi(s_j) \to k(s_i, s_j)$ (called kernel transformation of the problem). The re-parameterized problem optimizes $\mathbf{w}$ and the system is $\mathbf{w}^* = \tilde{A}^{-1}\tilde{\mathbf{b}}$ with parameters:

$$\tilde{A} = \sum_{t=0}^{n} \mathbf{k}(s_t) \left[\mathbf{k}^T(s_t) - \gamma \mathbf{k}^T(s_{t+1})\right], \quad \tilde{\mathbf{b}} = \sum_{t=0}^{n} \mathbf{k}(s_t) R_t, \tag{5}$$

where $\tilde{A}$, $\tilde{\mathbf{b}}$ are the re-parametrized entities of eq. (4) and $k(\cdot,\cdot)$ is an arbitrary kernel function [18]. $\mathbf{k}(s) = [k(s,s_1), \ldots, k(s,s_n)]^T$ is the vector of kernels at the new data s and the training data set $\{s_i \,|\, i = \overline{1,n}\}$. An important consequence is that the expression for the value function – originally in feature space, eq. (1) – is expressed using kernels too:

$$\tilde{V}(s) = \sum_{i=0}^{n} w_i k(s, s_i), \quad \forall\, s \in S. \tag{6}$$

In what follows we reduce the computational time required to obtain the vector $\mathbf{w}$, by first proposing a recursive calculation of the matrix $\tilde{A}^{-1}$ and vector b, then present a sparsification mechanism that reduces the number of parameters in the system.

4.1 Recursive Computation

We assume that we are reading the elements of the data-set one by one. When n inputs have been processed, the size $\tilde{A}$ is n, consequently the computation of $\mathbf{w}$ is $O(n^3)$. On-line learning makes this problem worse since it requires the inversion of $\tilde{A}$ to be performed at each step which is highly impractical. The computational burden of the matrix inversion can be mitigated by keeping $C_t \overset{\text{def}}{=} \tilde{A}_t^{-1}$ and updating it incrementally. Suppose we have processed t data-points – therefore $C_t \in \mathbb{R}^{txt}$ and $\tilde{\mathbf{b}} \in \mathbb{R}^{tx1}$ – and let us write implicitly update to $\tilde{A}_{t+1}$:

$$\begin{aligned}
\tilde{A}_{t+1} &= \frac{t}{t+1}\left(\begin{bmatrix} \tilde{A}_t & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \begin{bmatrix} k(s_t, s_1) \\ \vdots \\ k(s_t, s_{t+1}) \end{bmatrix} \cdot \begin{bmatrix} k(s_t, s_1) - \gamma k(s_{t+1}, s_1) \\ \vdots \\ k(s_t, s_{t+1}) - \gamma k(s_{t+1}, s_{t+1}) \end{bmatrix}^T \right) \\
&\overset{\text{def}}{=} \frac{t}{t+1}\left(\begin{bmatrix} \tilde{A}_t & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ u^* \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ v^* \end{bmatrix}^T \right)
\end{aligned} \tag{7}$$

where – in the second line – we defined, for a more concise notation, the vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{t\times 1}$ and the scalars u^* and v^*. We also see that the inclusion of the new input increases the size of the matrix $\tilde{A}_{t+1}$. With the above notation we can express the recursion for the new value of the *inverse*, that is:

$$C_{t+1} = \tilde{A}_{t+1}^{-1} = \frac{t+1}{t}\begin{bmatrix} \tilde{A}_t + \mathbf{u}\mathbf{v}^T & v^*\mathbf{u} \\ u^*\mathbf{v}^T & u^*v^* \end{bmatrix}^{-1} = \begin{bmatrix} C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^*v^*} \end{bmatrix} \tag{8}$$

where we used the inversion rules for block matrices and the Sherman-Woodbury formula, similarly to [5]. Notice the particular simplicity of the update: as its name suggests, it is indeed a rank-one update, involving only the last line and column of a – potentially large – matrix. The iterative update of $\mathbf{b}_t$ is:

$$\tilde{\mathbf{b}}_{t+1} = \frac{t}{t+1}\left(\begin{bmatrix} \tilde{\mathbf{b}}_t \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ u^* \end{bmatrix} R_t\right) \tag{9}$$

Combining the above defined representation of the inverse of $\tilde{A}$ with Eq. (5), the optimal coefficients for the approximation of the target function after processing the $(t+1)$-th input data-point can be obtained as:

$$\begin{aligned} \mathbf{w}_{t+1} &= C_{t+1}\tilde{\mathbf{b}}_{t+1} = \begin{bmatrix} C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^*v^*} \end{bmatrix}\left(\begin{bmatrix} \tilde{\mathbf{b}}_t \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{u} \\ u^* \end{bmatrix} R_t\right) \\ &= \begin{bmatrix} \mathbf{w}_t \\ \left(R_t - \mathbf{v}^T\mathbf{w}_t\right)/v^* \end{bmatrix} = \begin{bmatrix} \mathbf{w}_t \\ \left(R_t - (\tilde{V}(s_t) - \gamma\tilde{V}(s_{t+1}))\right)/v^* \end{bmatrix} \end{aligned} \tag{10}$$

In the rest of this paper we will refer to the presented iterative algorithm as kernel recursive least squares temporal difference learning algorithm or (KRLSTD). For a detailed derivation of a policy iteration algorithm based on the least squares value function approximation method, see [10].

In the next section we present a new type of sparsification mechanism to keep the matrix $\tilde{A}$ at a small fixed size. Since the sparsification depends on the *values of the inputs*, we need a *"good"* subset, the criteria presented on the next section.

5 Sparsification of the Representation

Sparsification reduces the computational cost of kernel-based algorithms by finding a small set of data-points called *dictionary*, which will be used as basis set for subsequent computation. The subject of selecting a representative set of data-points for reducing computational cost in case of kernel methods has been studied in-depth in the scientific literature. To achieve sparsity a large number of different approaches have been employed. Error-insensitive cost functions in support vector machine regression, reduction of the rank of the kernel matrix by low-rank approximations, greedy feature construction algorithms, using a Bayesian prior to force sparsity in case of fully probabilistic models *etc.* For non-parametric kernel methods this process can also be called feature selection, since the feature representation of a data-point is determined by the subset of data-points in the dictionary.

The commonly used sparsification method in RL [8] uses approximate linear independence (ALD) as a criterion for discarding data-points. In this context a new point is approximated by the linear combination of dictionary points in feature space, and the approximation error is as follows:

$$\varepsilon = \min_{\alpha} \| \textstyle\sum_{i=1}^{d} \alpha_i \phi(s_i) - \phi(s^*) \|^2 \tag{11}$$

Here $\phi(s)$ are the feature images of the corresponding data-points, s^* is the new data point and α_i are the variational coefficients. Minimizing the error from (11) and applying the kernel trick leads to the following expression fo the approximation coefficients:

$$\alpha = K^{-1}\mathbf{k}(\mathbf{s}^*) \tag{12}$$

where $K_{i,j} = k(s_i, s_j)$ is the kernel matrix and $\mathbf{k}(\mathbf{s}^*) = \left[k(s_1, s^*) \ldots k(s_n, s^*)\right]$ is the vector of kernel values computed on the new data-point s^*.

If the approximation error is below a predefined threshold ν the data-point is discarded and the kernel-matrix is updated using the linear combination coefficients α_i[2]. Otherwise the new data-point is incorporated into the dictionary. In [4] a sparsification mechanism for Gaussian processes is presented which considers the influence of individual data-points in the approximation accuracy. The method is based on calculating the Kullback-Leibler divergence between two posterior Gaussian process distributions: one with a new data-point included and one without. The data-point that causes the smallest change in the posterior mean is discarded. Unfortunately this can only be applied when we have a full probabilistic model available as in the case of Gaussian Processes. Moreover the kernel hyper-parameters have a major influence on the accuracy of this method.

Proponents of sparsification methods based on approximate linear dependency argue that one of its major advantages is that it is not affected by noise in the training data, since it does not take into account the training labels, only the linear dependency of inputs in feature space. In the problem-domain of reinforcement learning, however this becomes a drawback since for the accurate approximation of value functions, the information contained in the training labels needs to be taken into account.

In what follows we introduce a sparsification mechanism that puts a larger emphasis on the characteristics of the target-function in the selection process of dictionary data-points. Our method can be applied on-line, is not dependent on the initial values of the kernel hyper-parameters and it has only one parameter that requires tuning.

5.1 Laplacian Sparsification Criteria

The main idea of our method is to approximate the manifold upon which the training-data lies, and use the characteristics of this manifold to adjust the density of

[2] For a detailed description of the update procedure see [3]

training data-points contained in the dictionary in specific regions of the input space. To obtain a coarse representation of the manifold we propose the on-line construction of a similarity graph based on the observed state-transitions of the system. The graph construction mechanism uses the information contained in the sequential nature of the training data to create edges between vertexes(states or state-action pairs) that are successively traversed[3]. The spectral information contained in the Laplacian of a properly constructed similarity graph can be exploited when deciding on the inclusion or removal of a new data-point to the dictionary.

To provide the necessary background some notions from graph theory are needed. Let $\mathscr{G}(E,V)$ be an undirected graph with E and V denoting the set of edges and vertices respectively. We define the unnormalized graph Laplacian associated with $G(\mathscr{E},\mathscr{V})$ as: $L^{\mathscr{G}} = D^{\mathscr{G}} - A^{\mathscr{G}}$, where $A^{\mathscr{G}}$ is the weighted adjacency matrix of $\mathscr{G}$ and $D^{\mathscr{G}}_{i,i} = \sum_{j\neq i} A^{\mathscr{G}}_{i,j}$ is a diagonal matrix containing the node degrees on the diagonal.

An interesting property of the graph Laplacian is that it is symmetric, positive semi-definite, and it gives rise to the discrete Laplacian operator Δ:

$$f : V \to \mathbb{R} \quad \Delta f(s_i) = \sum_{j=1}^{n} A^{\mathscr{G}}_{ij} \left[f(s_i) - f(s_j)\right] \tag{13}$$

For a given set of data-points consisting of the vertexes of $\mathscr{G} : s_i \in V \quad i = \overline{1,|V|}$ the Laplacian operator applied to the function f is expressed as:

$$\Delta \mathbf{f} = L^{\mathscr{G}} \mathbf{f} \quad \text{where} \quad \mathbf{f} = [f(s_1) \ldots f(s_n)]^T \tag{14}$$

For now we assume that the graph approximates the manifold upon which the training data-points are situated, and the shortest paths calculated on $\mathscr{G}$ between two points approximate the geodesic distances between them[4].

Let us denote the approximated function f and the neighbor set of a vertex s_i as $\mathrm{neig}(s_i) = \{s_j \in V | A^{\mathscr{G}}_{i,j} \neq 0\}$. The vertexes of the graph $\mathscr{G}(E,V)$ are the location of samples from the target function. To obtain a sparse dictionary we introduce a score function $\mu(\cdot)$ for a vertex $s_i \in V$ as the weighted sum of the squared differences between target function values of s_i and its neighbouring vertexes:

$$\mu(s_i \in V) = \sum_{s_j \in \mathrm{neig}(s_i)} A^{\mathscr{G}}_{ij} \left[f(s_i) - f(s_j)\right]^2 \tag{15}$$

Summing up the individual vertex scores gives an overall measure about the quality of the dictionary set:

$$\sum_{s_i \in V} \mu(s_i) = \sum_{i,j=1}^{n} A^{\mathscr{G}}_{ij} \left[f(s_i) - f(s_j)\right]^2 = \mathbf{f}^T L^{\mathscr{G}} \mathbf{f} \tag{16}$$

[3] The on-line construction of the similarity graph is explained in detail in section 6

[4] Geodesic distance is the distance between states along the manifold determined by the transition dynamics of the MRP.

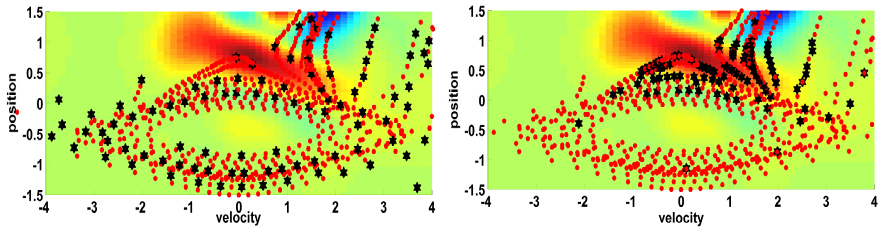


Fig. 1 Dictionary points (*black stars*) obtained after processing 5 roll-outs ($\approx$ 450 data-points) – *red dots* – for the mountain-car control problem using different sparsification techniques. Figure (a): ALD sparsification. Figure (b) our proximity graph-based sparsification. The maximum number of dictionary points was set to 100, KLSTD approximation used.

The above presented score is equivalent to applying the discrete Laplacian operator to the target function, and can be interpreted as the overall divergence of the gradient of f. In Section 7 we describe the complete sparsification algorithm that uses the score from Eq. 16 as a criterion to construct an optimal dictionary.

5.2 *Illustrative Example*

To illustrate how the above proposed sparsification mechanism changes the density of the dictionary data-points in different regions of the approximated function, we performed a simple experiment on approximating the value function for a fixed policy π on the mountain-car control problem (A detailed description is given in section 8.2). Figure 1 illustrates the differences between approximate linear dependence-based sparsification and our Laplacian-based method. The degree of sparseness in case of ALD is controlled by the threshold parameter ν from section 5. To be able to compare the two algorithms, we adjusted the upper limit on the dictionary size in case of our method to match the number of dictionary points produced by the threshold ν in case of ALD.

The red dots on figure 1 show all the training data-points, the black stars show the elements of the dictionary after sparsification has been performed. The horizontal and vertical axis correspond to the two elements of the state vector and the state values are color coded in the background. Performing sparsification based on the maximization of the score from (16) leads to a higher sample-density in regions where the function changes rapidly (fig. 1b). At the same time the algorithm keeps a sufficient number of data-points in slowly changing regions to obtain good approximation accuracy.

6 On-Line Proximity Graph Construction

The information contained in the sequential nature of the training data in RL can be used to construct a manifold that reflects the state-transitions of the underlying MDP and the true geodesic distance between training data-points. We present two modalities for storing this information iteratively in so-called proximity graphs encountered in dimensionality reduction and surface reconstruction methods from point-cloud sets.

Let $\mathscr{G}(E,V)$ be a proximity graph that is the representation of a manifold upon which the training data-points are located.

We denote the new data-point which needs to be added to the existing graph structure with s_i and an edge between two vertexes s_i and s_j as e_{s_i,s_j}. After the addition of the new data-point to the graph vertex set $V = V \cup \{s_i\}$ new edges need to be constructed which connect it to existing vertexes in $\mathscr{G}$. Existing work [17], [27] on graph edge construction has focused on treating training data as either i.i.d or batch data. Since the approximation algorithm proposed by us operates on-line we present iterative versions of two well-known edge construction methods, the k-nearest-neighbor(KNN) and the extended sphere of influence (ε-SIG) methods.

6.1 Extended Sphere of Influence Graphs

The extended sphere of influence graph (eSIG) produces a good description of non-smooth surfaces and can accommodate variations in the point sample density [17]. In this approach, edges are constructed between vertexes with intersecting spheres of influence:

$$\begin{aligned} &E = E \cup \{e_{s_i,s_j} = \|s_i - s_j\|^2 \quad | \quad (R(s_i) + R(s_j)) > \|s_i - s_j\|\} \\ &R(s_i) = \|s_i - s_k\|^2 \end{aligned} \tag{17}$$

In the above formula $R(s_i)$ denotes the radius of the sphere of influence of s_i and s_k is the k-th nearest neighbor of s_i. The sphere of influence of a graph vertex is the sphere centered at the point, with radius given by its distance to its k-th nearest neighbor. Disconnected components can appear with this construction, however adjusting k the sphere of influence radius can be increased which helps in alleviating this problem.

6.2 KNN Edge Construction

The K-nearest neighbor method eliminates the problem of unconnected sub-graphs but introduces new problems such as : connections between points that are too far in state-space, asymmetric property of the adjacency matrix and increased computational cost. This strategy effectively limits the number of outgoing connections that a node can have by selecting the k nearest neighbors of x_t and creating an edge to them with length equal to their spatial distance.

$$e_{s_i,s_j} = \begin{cases} \|s_i - s_j\|^2 & \text{if} \quad s_j \in knn(s_i) \\ 0 & \text{otherwise} \end{cases} \qquad i = 1 \ldots n \tag{18}$$

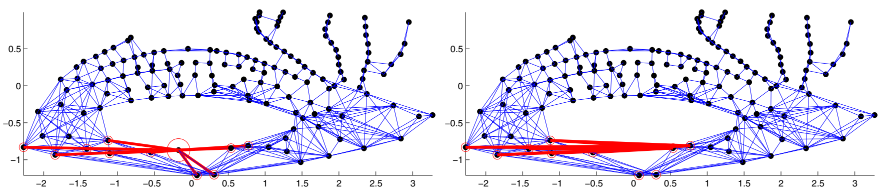


Fig. 2 Dictionary points (*black dots*) and graph structure (*blue lines*) before and after node removal. Figure (a) shows the node to be removed (*red circle*), its neighbours (*red circles with dots*) and the removable edges (*red lines*). On (b) we see the resulting graph with the newly created edges shown in red.

Here we used $knn(s_i)$ to denote the set containing the k vertexes nearest to s_i. The value of k also determines the number of edges present in the graph.

To achieve symmetry in the obtained proximity graph, we use undirected edges. As a consequence, when creating edges between data-points s_i and s_j we update the adjacency matrix as follows: $A^{\mathscr{G}}_{i,j} = A^{\mathscr{G}}_{j,i} = \|s_i - s_j\|^2$.

6.3 *Updating the Graph Structure*

The deletion of a vertex from the graph $\mathscr{G}(E,V)$ also removes the edges connecting it to its neighbors. In order to keep the information contained in the original edges about the topology we use the following pruning algorithm:

Let us denote the vertex which is to be removed by s^*. After removal of s^* we get a new graph structure $\mathscr{G}'(V',E')$ with the following components:

$$V' = V \setminus \{s^*\} \quad E' = E \setminus \{e(s^*,s) | s \in \text{neig}(s^*)\} \cup E_{new} \tag{19}$$

where $e(s^*,s)$ denotes an edge between the vertexes s^* and s of the graph G(E,V). The set E_{new} contains the new edges between former neighbors of s^* with weights equal to the sum of the edge weights that connected them to s^*:

$$E_{new} = \{e(s_i,s_j) = \|s_i,s^*\| + \|s_j,s^*\| | s_i,s_j \in \text{neig}(s^*), e(s_i,s_j) \notin E\} \tag{20}$$

Figure 2 illustrates the removal procedure on a sample graph structure obtained after interacting for 400 steps with the mountain-car environment.

The addition of the new edges serves two purposes: First the connected property of the graph is maintained, secondly it can be shown that using the previously defined update the shortest path distance matrix of the graph will remain the same except for removing the row and column corresponding to the index of s^*. In consequence the geodesic distances between the remaining graph nodes are preserved.

7 The KSLTD Algorithm with On-Line Laplacian Sparsification

The algorithm described in this section is used to incrementally construct a similarity graph, decide upon the addition of a new data-point to the dictionary and calculate the linear coefficients for the approximation of the value function.

To obtain a representative set of data-points we use the score function defined in Eq. (15) and the iterative update steps of the KLSTD data structures from Eq. (8) and (9). In the KLSTD setting, the target function values $f(s)$ from the laplacian sparsification criterion in Eq. (16) are replaced by the approximated values given by the KLSTD coefficients w^* and the actual composition of the dictionary D, according to Eq. (10). According to this the score function for an individual data-point takes the following form:

$$\mu(s_i) = \sum_{s_j \in neig(s_i)} A^{\mathscr{G}}_{i,j} (\sum_{t=1}^{|D|} w^*_t (k(s_i, s_t) - k(s_j, s_t))^2 \tag{21}$$

Let us denote the upper limit to the size of the dictionary by $maxBV$. Whenever the upper limit has been reached a dictionary point is selected to be discarded, based on the score from Eq. 21. Discarding a data-point has two consequences: (1) the KRLSTD coefficients C and $\tilde{\mathbf{b}}$ and (2) the structure of the graph $\mathscr{G}(E,V)$ need to be updated accordingly. Due to the linear relationship between the approximation coefficients and the KRLSTD parameters, the parameters w^*, the rows and columns of C and the corresponding entries of $\tilde{\mathbf{b}}$ can be arbitrarily permuted. For example if we want to discard data-point s_i from D where $|D| = n$ The following permutation can be performed:

$$\begin{aligned} w^* &= [w^*_1 \dots w^*_i \dots w^*_n]^T \rightarrow [w^*_1 \dots w^*_n \dots w^*_i]^T \\ \tilde{\mathbf{b}} &= \left[\tilde{b}_1 \dots \tilde{b}_i \dots \tilde{b}_n\right]^T \rightarrow \left[\tilde{b}_1 \dots \tilde{b}_n \dots \tilde{b}_i\right]^T \end{aligned}$$

The i-th row and column of C can be exchanged with the last row and column:

$$C = \begin{bmatrix} C_{1,1} \dots C_{1,i} \dots C_{1,n} \\ \vdots \\ C_{i,1} \dots C_{i,i} \dots C_{i,n} \\ \vdots \\ C_{n,1} \dots C_{n,i} \dots C_{n,n} \end{bmatrix} \rightarrow \begin{bmatrix} C_{1,1} \dots C_{1,n} \dots C_{1,i} \\ \vdots \\ C_{n,1} \dots C_{n,n} \dots C_{n,i} \\ \vdots \\ C_{i,1} \dots C_{i,n} \dots C_{i,i} \end{bmatrix} \tag{22}$$

The final step of the update after the above transformations is to simply delete the last row and column of C and the last entry from the approximation coefficient vector w^* and $\tilde{\mathbf{b}}$

Algorithm 1. KRLSTD with Laplacian sparsification

```
1:  KRLSTD data structures: $\tilde{A}_1 = k(s_1,s_1), C_1 = \frac{1}{k(s_1,s_1)}$,   $\tilde{b}_1 = k(s_1,s_1)\cdot r_1$
2:  Dictionary parameters: $D_1 = \{s_1\}$, maxBV=const, n=1
3:  Graph parameters: $\mathcal{G}(E,V)$   $E = \emptyset$   $V = s_1$
4:  **repeat**
5:      get a new training data-point $\{s_t , R_t\}$
6:      $D_t = D_{t-1} \cup \{s_t\}$
7:      expand proximity graph $\mathcal{G}(E,V)$                                   Eq. (17),(18)
8:      update $C_t \to C_{t+1}$, $\tilde{\mathbf{b}}_t \to \tilde{\mathbf{b}}_{t+1}$                                   Eq. (8),(9)
9:      calculate new approximation coefficients $w_t^*$                            Eq. (10)
10:     **if** $|D_t| >= maxBV$ **then**
11:         calculate scores: $\mu(s_i)$   $i = \overline{1,n}$   $n = |D_t|$              Eq. (21)
12:         select least significant point $s^* = \arg\min_s(\mu(s)|s \in D_t)$
13:         eliminate $s^*$ from the dictionary: $D_t = D_t \setminus \{s_r\}$
14:         update $C_{t+1}$ , $\tilde{\mathbf{b}}_{t+1}$ and $w_{t+1}^*$                        Eq. (22),(22)
15:         update $\mathcal{G}(E,V)$                                               Eq. (19),(20)
16:     **else**
17:         dictionary remains unchanged
18:     **end if**
19: **until** approximation coefficients $w^*$ converged
20: Output: $D$   $w^*$
```

8 Performance Evaluation

To measure the performance of our policy evaluation framework in a continuous Markov reward process we make use of two quality measures: the Bellman Error (BE) and the mean absolute error with respect to the true value function.

The Bellman error expresses the ability of the approximator to predict the value of the next state based on the current state's approximated value. It also indicates how well the function approximator has converged

$$BE(\tilde{V}(s)) = R(s) + \gamma \int_{s' \in S} P(s'|s,a) \int_{a \in A} \pi(a|s)\tilde{V}(s')dads' - \tilde{V}(s) \tag{23}$$

According to a study in [24] the Bellman error gives an upper bound on the approximation of $\tilde{v}$:

$$||V - \tilde{V}||_\infty \leq \frac{||BE(\tilde{V})||_\infty}{1-\gamma} \tag{24}$$

The mean absolute error simply compares the accuracy of the approximated value of a set of states to the values given by the true value function. It is defined as the sum of the absolute differences between approximated and true state values:

$$MAE(\tilde{V}(.)) = \int_{s \in S} |\tilde{v}(s) - V(s)|ds \tag{25}$$

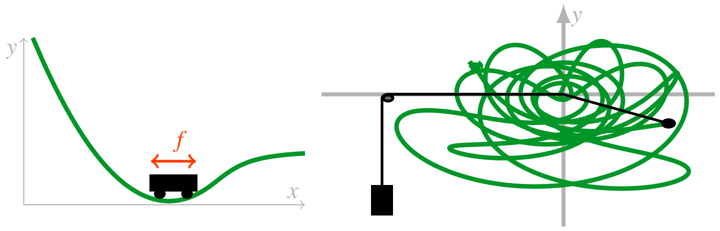


Fig. 3 Illustrations of the (a) car on the hill and (b) swinging Atwood's machine dynamics systems

In our experiments we approximate the above integral by the sample average evaluated over a predefined set of data-points sampled from the state space of the corresponding MRP.

The true value function V used as a baseline in our experiments is calculated in all experimental settings using exhaustive Monte Carlo approximation and can be considered very close to the actual value function induced by the policy π.

To test the performance of our methods on higher dimensional data we choose the problem of learning the state value functions of highly non-linear dynamical systems. In this setting we model the value function of the dynamical system with the help of a KLSTD approximator, where inputs are states and the outputs are the associated values:

$$\tilde{V}_{\pi}(s_t)\sum_{i=0}^{n} w_i^* k(s_i,s_t) \sim E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^t r_i | r_i = R(s_i), s_0 = s_t\right] \tag{26}$$

For testing we used two well-known toy problems from reinforcement learning: the Car on hill (a.k.a mountain car) (Fig. 3a) control and the swinging Atwood's machine (Fig. 3b).

8.1 Experimental Setup

To assess the performance of the algorithms we averaged the results of a large number of experiments performed on data generated from the dynamical systems. In each experiment we generated training and test-points, sampled from trajectories[5] where the start states were chosen randomly from within the state-boundaries of the system. For each sampled state-action vector we also collected the corresponding immediate reward, defined by the reward function.

To generate the system inputs we used a fixed Gaussian policy with a linear controller of the following form: $\pi(a,s) \sim N(ctrl(s),\Sigma)$ where $ctrl(s) = \sum_{i=1}^{d} \alpha_i s_i$ is the controller, α_i being the linear coefficients and d the dimensionality of the state vector, $\Sigma = diag(\sigma)$ is the d dimensional diagonal covariance matrix.

[5] A trajectory is a set of successive states starting from an initial state and applying a fixed policy, until a terminal state has been reached.

We also perturbed the transition probabilities of the systems with a Gaussian distributed noise of variance: $\sigma = 0.01$ to simulate real-world conditions. Another important detail of our experimental setup is the initialization of the kernel hyper-parameters in case of ALD and the initialization of the nearest neighbor count value k in the graph construction algorithms. The kernel function that we used in our experiments was the Gaussian kernel with added Gaussian noise. The kernel hyper-parameters (the characteristic length scale, the noise variance and the signal variance) were optimized using a scaled conjugate gradient optimization to fit the data as well as possible. For setting the nearest neighbor count value k we used the dimensionality of the input data-points as a reference.

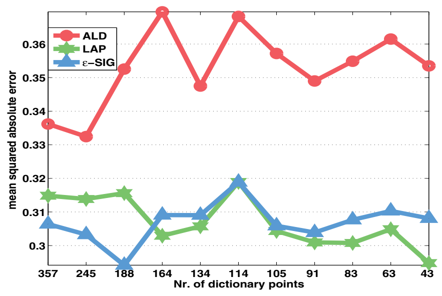
In what follows we describe each of the studied dynamical systems and present the test results.

8.2 Car on Hill Control Problem

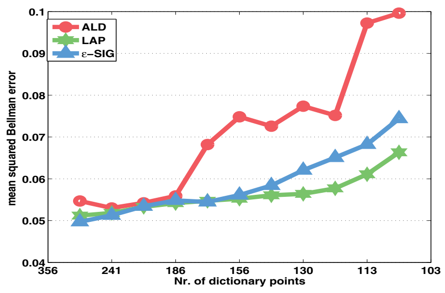
The first test system is the well-known mountain-car control problem [22], shown schematically in Fig. 3.a. A car is placed in the a valley and the goal is to apply a sequence forces along the longitudinal axis of the car such that it will climb out the valley. This problem imposes a two dimensional continuous state space $s = \begin{bmatrix} x & v \end{bmatrix}$ composed of the position of the car x and its speed $v = \dot{x}$. The applied action is one dimensional and continuous. To demonstrate the benefits of our Laplacian-based sparsification mechanism for the approximation accuracy we performed 15 experiments each having 150 episodes consisting of 350 steps each – ~ 45000 training-points – for this problem. To see the effects of the sparsification on the approximation accuracy, we performed the same experiments for a number of different maximum dictionary sizes. Figure 4.a shows the mean absolute errors with respect to the true value function.

According to our experiments, the Laplacian-based sparsification mechanism leads to a lower approximation error than standard ALD for all maximum dictionary sizes. The approximation accuracy even increases slightly when the dictionary size is drastically reduced as opposed to ALD where having fewer dictionary points raises the approximation error. This may be attributed to the better placement of data-points into regions of the state-space where the target function changes more rapidly.

Figure 4.b shows the evolution of the Bellman error from the same perspective. It can be seen that the Laplacian-based sparsification mechanism with knn or $\varepsilon - SIG$ proximity graphs performs better at low dictionary sizes than ALD, the difference becoming more obvious as the dictionary size is further decreased. Figure 5.a illustrates the evolution of the mean Bellman error as a function of the number of training points used for the estimation of the approximation parameters $\mathbf{w}$ from section 4. We calculated the dictionary beforehand based on a fixed training data set using each of the three sparsification methods. When small number of training data is used, the three dictionaries perform similarly. However when the training-data size increases the dictionary obtained by ALD leads to unstable value estimates, whereas using the

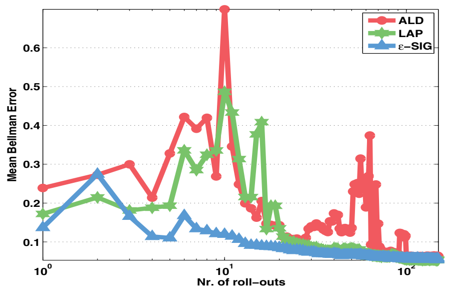


(a)

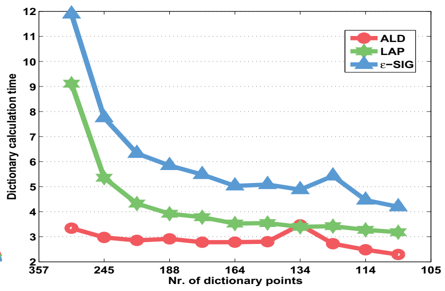


(b)

Fig. 4 Evolution of the mean absolute error for different maximum dictionary sizes averaged over 15 experiments, in case of standard *ALD* sparsification, and our Laplacian based sparsification with *knn* and $\varepsilon - SIG$ proximity graphs. The horizontal axis represents the maximum number of dictionary points, the vertical axis on figure (a) shows the mean squared absolute error while on figure (b) the mean squared Bellman error. Measurement of the errors was done on 3600 equally distributed points from the state space.



(a)



(b)

Fig. 5 (a) Mean Bellman error as a function of training data set size. (b)Time required for the computation of the dictionary from approximately 45000 data-points using ALD and Laplacian-based sparsification with *knn* and $\varepsilon - SIG$ proximity graphs.

dictionary obtained by the Laplacian-based sparsification we obtain more accurate and stable values.

8.3 Swinging Atwood's Machine

The swinging Atwood's machine is composed of two weights connected by a rope and two pulleys, the larger weight can be moved only in the vertical plane, while the smaller weight can rotate freely around the second pulley. A schematic representation of the system with a sample trajectory described by the smaller weight can be seen on fig. 3(b). The goal of the control problem is to guide the vertical force applied to the larger weight in such a manner that the small weight moves roughly along a circle with a given radius. The state representation is four

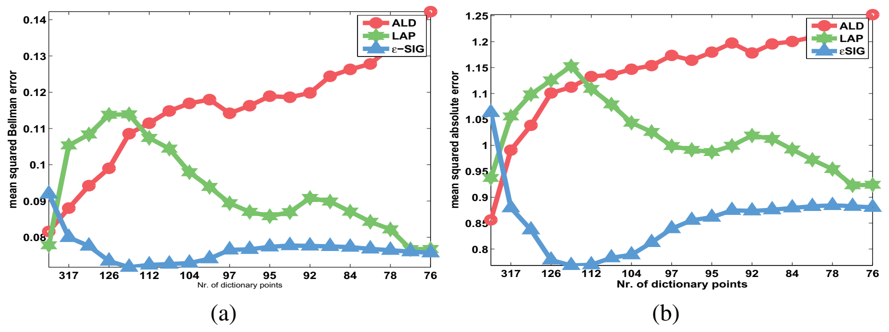


Fig. 6 Evolution of the mean squared bellman error (a) and the mean absolute error (b) as a function of maximum dictionary size, for the swinging Atwood's machine control problem. The horizontal axis shows the maximum number of basis vectors allowed while the vertical axis shows the error values. The uneven distribution of the numbers on the horizontal axis is due to the fact that ALD does not put on upper limit on the dictionary size, it is determined by the threshold parameter. Therefore the Laplacian and eSIG algorithms are adjusted to contain the same number of dictionary points as resulted from ALD sparsification

dimensional, $s \overset{\text{def}}{=} [l, \dot{l}, \theta, \omega]$, where l is the length of the rope from the second pulley to the smaller weight; $\dot{l}$ is the rate of change in the rope's length; θ is the angle of the smaller weight; ω is the angular velocity. Our implementation of this system is based on a simplified version of the dynamic equations from [14], where a detailed description of the system and its unstable dynamics is also given.

Figure 8.3 illustrates the performance of the three sparsification algorithms (ALD, Laplacian with *knn* (LAP) and Laplacian with extended sphere of influence graphs (eSIG)) as a function of the maximum size of the dictionary. The two sub-figures correspond to the mean squared absolute error and the mean squared Bellman error. Similarly to the case of the car-on-hill control problem, when the maximum size of the dictionary is reduced the two laplacian based sparsification mechanisms produce more accurate estimates both when measured according to the Bellman and the absolute error. In case of the swinging Atwood's machine the extended sphere of influence graph construction produces the best results.

9 Conclusion

We have seen a number of proposed methods for reducing the computational costs of kernel-based value function approximation. As a major contribution of this work we presented a new method that exploits the spatio-temporal relationship between training data-points and the shape of the target function to be approximated. Our method can adjust the density of the dictionary set according to the characteristics of the target function, leading to better approximation accuracy in value function approximation and as a consequence faster convergence rates in value-based reinforcement learning algorithms.

The computational complexity of the presented approximation framework is influenced by the nearest neighbor search in case of graph expansion and the search for vertexes with minimal score in case of the sparsification mechanism. The calculation of the individual scores of each graph vertex $v_i \in V$ has a computational complexity of $O(0)$ since the scores can be stored and updated on-line for each vertex. Compared to the cost of approximate linear independence test our methods are slower as it can be seen from 5(b), but not by orders of magnitude, the difference becomes significant only by large dictionary sizes. The better approximation accuracy and faster convergence rates compensate for the higher computational requirements.

The use of proximity graphs for sparsification enables the extension of our methods with different distance-substitution kernels operating on graphs. An superficial exploration of this subject can be found in [9]. Our method also opens up ways to different exploration strategies like probabilistic road-maps or rapidly exploring random trees and experience replay mechanisms aimed at reducing the number of actual trials needed for learning. The use of proximity graphs for sparsification enables the extension of our methods with different distance-substitution kernels operating on graphs. A superficial exploration of this subject can be found in [9]. Our method also opens up ways to different exploration strategies like probabilistic road-maps or rapidly exploring random trees and experience replay mechanisms aimed at reducing the number of actual trials needed for learning.

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

References

1. Boyan, J.A.: Technical update: Least-squares temporal difference learning. Machine Learning 49(2-3), 233–246 (2002)
2. Bradtke, S.J., Barto, A.G., Kaelbling, P.: Linear least-squares algorithms for temporal difference learning. Machine Learning, 22–33 (1996)
3. Csató, L.: Gaussian Processes – Iterative Sparse Approximation. PhD thesis, Neural Computing Research Group (March 2002)
4. Csató, L., Opper, M.: Sparse representation for Gaussian process models. In: Leen, T.K., Dietterich, T.G., Tresp, V. (eds.) NIPS, vol. 13, pp. 444–450. The MIT Press (2001)
5. Csató, L., Opper, M.: Sparse on-line Gaussian Processes. Neural Computation 14(3), 641–669 (2002)
6. Deisenroth, M.P., Rasmussen, C.E.: PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA (June 2011)
7. Deisenroth, M.P., Rasmussen, C.E., Peters, J.: Gaussian process dynamic programming. Neurocomputing 72(7-9), 1508–1524 (2009)
8. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. IEEE Transactions on Signal Processing 52, 2275–2285 (2003)
9. Jakab, H., Csató, L.: Manifold-based non-parametric learning of action-value functions. In: Verleysen, M. (ed.) European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, pp. 579–585. UCL, KULeuven (2012)
10. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. J. Mach. Learn. Res. 4, 1107–1149 (2003)

11. Maei, H., Szepesvari, C., Bhatnagar, S., Precup, D., Silver, D., Sutton, R.: Convergent temporal-difference learning with arbitrary smooth function approximation. In: Advances in Neural Information Processing Systems NIPS 22, pp. 1204–1212 (2009)
12. Mahadevan, S., Maggioni, M.: Value function approximation with diffusion wavelets and laplacian eigenfunctions. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems 18, pp. 843–850. MIT Press, Cambridge (2006)
13. McCullagh, P., Nelder, J.A.: Generalized Linear Models. Chapman & Hall, London (1989)
14. Olivier, P., Perez, J.-P., Simó, C., Simon, S., Weil, J.-A.: Swinging atwood's machine: Experimental and numerical results, and a theoretical study. Physica D 239, 1067–1081 (2010)
15. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York (1994)
16. Riedmiller, M.: Neural fitted q iteration: first experiences with a data efficient neural reinforcement learning method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
17. Ruggeri, M.R., Saupe, D.: Isometry-invariant matching of point set surfaces. In: Eurographics Workshop on 3D Object Retrieval (2008)
18. Schölkopf, B., Smola, A.J.: Learning with Kernels. The MIT Press, Cambridge (2002)
19. Seeger, M.W., Kakade, S.M., Foster, D.P.: Information consistency of nonparametric gaussian process methods
20. Sugiyama, M., Hachiya, H., Kashima, H., Morimura, T.: Least absolute policy iteration for robust value function approximation. In: Proceedings of the 2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Piscataway, NJ, USA, pp. 699–704. IEEE Press (2009)
21. Sugiyama, M., Kawanabe, M.: Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation (2012)
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
23. Szepesvári, C.: Algorithms for Reinforcement Learning. Morgan & Claypool Publishers (2011)
24. Taylor, G., Parr, R.: Kernelized value function approximation for reinforcement learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, New York, NY, USA, pp. 1017–1024. ACM (2009)
25. Vapnik, V.N.: Statistical learning theory. John Wiley (1997)
26. Vollbrecht, H.: Hierarchic function approximation in kd-q-learning. In: Proc. Fourth Int. Knowledge-Based Intelligent Engineering Systems and Allied Technologies Conf., vol. 2, pp. 466–469 (2000)
27. von Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing 17(4) (2007)
28. Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks, 973–992 (2007)
29. Xu, X., Xie, T., Hu, D., Lu, X.: Kernel least-squares temporal difference learning. International Journal of Information Technology, 55–63 (2005)

Neural Networks Solution of Optimal Control Problems with Discrete Time Delays and Time-Dependent Learning of Infinitesimal Dynamic System

Tibor Kmet and Maria Kmetova

Abstract. The paper presented describes two possible applications of artificial neural networks. The first application is related to solve optimal control problems with discrete time delays in state and control variables subject to control and state constraints. The optimal control problem is transcribed into nonlinear programming problem which is implemented with feed forward adaptive critic neural network to find optimal control and optimal trajectory. The proposed simulation methods are illustrated by the optimal control problem of nitrogen transformation cycle model with discrete time delay of nutrient uptake. The second application deals with back-propagation learning of infinite-dimensional dynamical systems. The proposed simulation methods are illustrated by the back-propagation learning of continuous multilayer Hopfield neural network with discrete time delay using optimal trajectory as teacher signal. Results show that adaptive critic based systematic approach are promising in obtaining the optimal control with discrete time delays in state and control variables subject to control and state constraints and that continuous Hopfield neural networks are able to approximate signals generated from optimal trajectory.

1 Introduction

The essence of this paper, which elucidates and clarifies on our previous work [14], is to use neural networks methods for studying non-linear complex systems. We

Tibor Kmet
Constantine the Philosopher University, Department of Informatics,
Tr. A. Hlinku 1, 949 74 Nitra, Slovakia
e-mail: `tkmet@ukf.sk`

Maria Kmetova
Constantine the Philosopher University, Department of Mathematics,
Tr. A. Hlinku 1, 949 74 Nitra, Slovakia
e-mail: `mkmetova@ukf.sk`

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_15

examine in depth the simulation process of nitrogen transformation cycle thought a dynamical model based on adaptive critic designs and feed forward neural networks. It is assumed that the phytoplankton production must be simultaneously optimized both locally as well as globally. For this strategy we use methods from control of nonlinear systems with discrete time delays to obtain optimal trajectory. In addition we examine the approximation of the optimal trajectory. In this simulation we make use of Hopfield neural network with discrete time delays.

It is well known that the ability of artificial neural networks to approximate arbitrary nonlinear functions plays a primary role in the use of such networks as component or subsystems in identifiers and controllers [25]. Besides, it has been used for universal function approximation to solve optimal control problems forward in time to approximate co-state variable. Co-state variables play important role also in back-propagation learning of infinite-dimensional dynamical systems [24] in the case of time-dependent recurrent learning. The paper presented describes two possible applications of artificial neural networks. The first application is related to solve optimal control problems with discrete time delays in state and control variables subject to a control and state constraints. The second application is concerned to back-propagation learning of infinite-dimensional dynamical systems. Optimal control of nonlinear systems with discrete time delays in state and control variables is one of the most active subjects in control theory. There are rarely analytical solutions [7] although several numerical computation approaches have been proposed e.g. see [9], [11], [18], [23]. The most of the literature dealing with numerical methods for the solution of general optimal control problems focuses on algorithms for solving discretized problems. The basic idea of these methods is to apply nonlinear programming techniques to the resulting finite dimensional optimization problem [3], [9]. Then neural networks are used as universal function approximation to solve finite dimensional optimization problems forward in time with ”adaptive critic designs” [16], [17], [25]. For the neural network, a feed forward neural network with one hidden layer, a steepest descent error back-propagation rule, a hyperbolic tangent sigmoid transfer function and a linear transfer function were used.

The paper presented extends adaptive critic neural network architecture proposed by [13] to the optimal control problems with discrete time delays in state and control variables subject to control and state constraints. This paper is organized as follows. In Section 2, optimal control problems with delays in state and control variables subject to control and state constraints are introduced. We summarize the necessary optimality conditions, give a short overview of the basic results including the iterative numerical methods. In Section 3, we discuss the discretization methods for the given optimal control problem and formulate the resulting nonlinear programming problems. Section 4 presents a short description of adaptive critic neural network synthesis for the optimal control problem with delays in state and control variables subject to control and state constraints. We also present a new algorithm to solve optimal control problems. In Section 5 we present a description of back-propagation learning of infinite-dimensional dynamical systems and propose a new algorithm to calculate gradient of cost function. In Section 6, we present a description of the model of nitrogen transformation cycle with discrete time delay in nutrients uptake.

We apply the new proposed methods to the model presented to compare short-term and long-term strategies of nutrients uptake by phytoplankton. Numerical results are also given. Conclusions are being presented in Section 7.

2 The Optimal Control Problem

We consider the nonlinear control problem with delays in state and control variables subject to control and state constraints. Let $x(t) \in R^n$ and $u(t) \in R^m$ denote the state and control variable, respectively in a given time interval $[t_0, t_f]$. The optimal control problem is to minimize

$$\mathbb{J}(u) = g(x(t_f)) + \int_{t_0}^{t_f} f_0(x(t), x(t-\tau_x), u(t), u(t-\tau_u))dt \tag{1}$$

subject to

$$\begin{aligned} \dot{x}(t) &= f(x(t), x(t-\tau_x), u(t), u(t-\tau_u)), \\ x(t) &= \phi_s(t),\ \ u(t) = \phi_c(t),\ \ t \in [t_0 - \tau_u, t_0], \\ \psi(x(t_f)) &= 0,\ \ c(x(t), u(t)) \le 0,\ \ t \in [t_0, t_f], \end{aligned}$$

where $\tau_x \ge 0$ and $\tau_u \ge 0$ are discrete time delay in the state and control variable, respectively. The functions $g: R^n \to R$, $\ f_0: R^{n+m} \to R$, $\ f: R^{n+m} \to R^n$, $\ c: R^{n+m} \to R^q$ and $\psi: R^{n+m} \to R^r$, $0 \le r \le n$ are assumed to be sufficiently smooth on appropriate open sets and the initial conditions $\phi_s(t)$, $\phi_c(t)$ are continuous functions. The theory of necessary conditions for the optimal control problem of form (1) is well developed, see e.g. [9], [11], [19]. We introduce an additional state variable

$$x_0(t) = \int_0^t f_0(x(s), x(s-\tau_x), u(s), u(s-\tau_u)ds$$

defined by the

$$\dot{x}_0(t) = f_0(x(t), x(t-\tau_x), u(t), u(t-\tau_u),\ x_0(t) = 0,\ t \in [t_0 - \tau_x, t_0].$$

Then the augmented Hamiltonian function for problem (1) is

$$\mathscr{H}(x, x_{\tau_x}, u, u_{\tau_u}, \lambda, \mu) = \sum_{j=0}^{n} \lambda_j f_j(x, x_{\tau_x}, u, u_{\tau_u}) + \sum_{j=0}^{q} \mu_j c_j(x, u),$$

where $\lambda \in R^{n+1}$ is the adjoint variable and $\mu \in R^q$ is a multiplier associated to the inequality constraints. Assume that $\tau_x,\ \tau_u \ge 0$, $(\tau_x, \tau_u) \ne (0,0)$ and $\frac{\tau_x}{\tau_u} \in \mathbb{Q}$ for $\tau_u > 0$ or $\frac{\tau_u}{\tau_x} \in \mathbb{Q}$ for $\tau_x > 0$. Let $(\hat{x}, \hat{u})$ be an optimal solution for (1.) Then the necessary optimality condition for (1) implies [9] that there exist a piecewise continuous and piecewise continuously differentiable adjoint function $\lambda : [t_0, t_f] \to R^{n+1}$, a piecewise continuous multiplier function $\mu : [t_0,t_f] \to R^q,\ \hat{\mu}(t) \geq 0$ and a multiplier $\sigma \in R^r$ satisfying

$$\begin{aligned}
\dot{\lambda}_j(t) &= -\frac{\partial \mathscr{H}}{\partial x_j}(\hat{x}(t),\hat{x}(t-\tau_x),\hat{u}(t),\hat{u}(t-\tau_u),\lambda(t),\mu(t)) - \qquad (2)\\
&\quad \chi_{[t_0,t_f-\tau_x]}\frac{\partial \mathscr{H}}{\partial x_{\tau_x j}}(\hat{x}(t+\tau_x),\hat{x}(t),\hat{u}(t+\tau_x),\hat{u}(t-\tau_u+\tau_x),\lambda(t+\tau_x),\mu(t+\tau_x)),\\
\lambda_j(t_f) &= g_{x_j}(\hat{x}(t_f)) + \sigma\psi_{x_j}(\hat{x}(t_f)),\ j=0,\ldots,n, \qquad (3)\\
0 &= -\frac{\partial \mathscr{H}}{\partial u_j}(\hat{x}(t),\hat{x}(t-\tau_x),\hat{u}(t),\hat{u}(t-\tau_u),\lambda(t),\mu(t)) - \qquad (4)\\
&\quad \chi_{[t_0,t_f-\tau_u]}\frac{\partial \mathscr{H}}{\partial u_{\tau_u j}}(\hat{x}(t+\tau_u),\hat{x}(t-\tau_x+\tau_u),\hat{u}(t+\tau_u),\hat{u}(t),\lambda(t+\tau_u),\mu(t+\tau_u)),\\
&\quad j=1,\ldots,m.
\end{aligned}$$

Furthermore, the complementary conditions hold, i.e. in $t \in [t_0,t_f],\ \mu(t) \geq 0$, $c(x(t),u(t)) \leq 0$ and $\mu(t)c(x(t),u(t)) = 0$. Herein, the subscript $x,\ x_{\tau_x},\ u$ and u_{τ_u} denotes the partial derivative with respect to $x,\ x_{\tau_x},\ u$ and u_{τ_u}, respectively.

For the free terminal time t_f, an additional condition needs to be satisfied:

$$\mathscr{H}(t_f) = \left(\sum_{j=0}^{n}\lambda_j f_j(x,u) + \sum_{j=0}^{q}\mu_j c_j(x,u)\right)|_{t_f} = 0. \qquad (5)$$

and define $\mathscr{H}(i)$ and ϕ as follows [2]:

$$\begin{aligned}
\mathscr{H}(i) &= \lambda(i+1)(x^i + hf(x^i,x^{i-k},u^i,u^{i-l}),\\
\phi &= \mathscr{G} + \sigma\psi.
\end{aligned}$$

3 Discretization of the Optimal Control Problem

The direct optimization methods for solving the optimal control problem are based on a suitable discretization of (1), see e.g. [3], [9]. We assume that $\tau_u = l\frac{\tau_x}{k}$ with $l,k \in \mathbb{N}$. Defining $h_{max} = \frac{\tau_x}{k}$ gives the maximum interval length for an elementary transformation interval that satisfies $\frac{\tau_x}{h_{max}} = k \in \mathbb{N}$ and $\frac{\tau_u}{h_{max}} = l \in \mathbb{N}$. The minimum grid point number for an equidistant discretization mesh $N_{min} = \frac{t_f-t_0}{h_{max}}$. Choose a natural number $K \in \mathbb{N}$ and set $N = KN_{min}$. Let $t_i \in \langle t_0,t_f\rangle,\ i=0,\ldots,N$, be an equidistant mesh point with $t_i = t_0 + ih, i=0,\ldots,N$, where $h = \frac{b-a}{N}$ is a time step and $t_f = Nh + t_0$. Let the vectors $x^i \in R^{n+1},\ u^i \in R^m, i=0,\ldots,N$, be an approximation of the state variable and control variable $x(t_i),\ u(t_i)$, respectively at the mesh point t_i. Euler's approximation applied to the differential equations yields $x^{i+1} = x^i + hf(x^i,x^{i-k},u^i,u^{i-l}),\ i=0,\ldots,N-1$. Choosing the optimal variable $z := (x^0,x^1,\ldots,x^{N-1},u^0,\ldots,u^{N-1}) \in R^{N_s},\ N_s = (n+m)N,$ the optimal control problem is replaced by the following discretized control problem in the form of nonlinear programming problem with inequality constraints: Minimize

$$\mathbb{J}(z) = \mathbb{G}(x^N) = g((x_1,\dots,x_n)^N) + x_0^N \tag{6}$$

subject to

$$\begin{aligned} x^{i+1} &= x^i + hf(x^i, x^{i-k}, u^i, u^{i-l}),\ i=0,\dots,N-1, \\ x^{-i} &= \phi_x(t_0 - ih),\ \ i=k,\dots,0,\ \ u^{-i} = \phi_u(t_0 - ih),\ \ i=l,\dots,0, \\ \psi(x^N) &= 0,\ \ c(x^i,u^i) \le 0,\ i=0,\dots,N-1. \end{aligned} \tag{7}$$

In a discrete-time formulation we want to find an admissible control which minimizes objective function (6). Let us introduce the Lagrangian function for the nonlinear optimization problem (6):

$$\begin{aligned} \mathscr{L}(z,\lambda,\sigma,\mu) = & \sum_{i=0}^{N-1} \lambda^{i+1}(-x^{i+1} + x^i + hf(x^i,x^{i-k},u^i,u^{i-l})) + \mathbb{G}(x^N) + \\ & \sum_{i=0}^{N-1} \mu^i c(x^i,u^i) + \sigma\psi(x^N). \end{aligned}$$

The first order optimality conditions of Karush-Kuhn-Tucker [18] for the problem (6) are:

$$\begin{aligned} 0 &= \mathscr{L}_{x^i}(z,\lambda,\sigma,\mu) = \lambda^{i+1} - \lambda^i + h\lambda^{i+1} f_{x^i}(x^i,x^{i-k},u^i,u^{i-l}) + \\ &\qquad h\lambda^{i+k+1} f_{x^i_{\tau_x}}(x^{i+k},x^i,u^{i+k},u^{i-l+k}) + \mu^i c_{x^i}(x^i,u^i), \\ &\qquad i=0,\dots,N-k-1, \\ 0 &= \mathscr{L}_{x^i}(z,\lambda,\sigma,\mu) = \lambda^{i+1} - \lambda^i + h\lambda^{i+1} f_{x^i}(x^i,x^{i-k},u^i,u^{i-l}) + \mu^i c_{x^i}(x^i,u^i), \\ &\qquad i=N-k,\dots,N-1, \end{aligned} \tag{8}$$

$$0 = \mathscr{L}_{x^N}(z,\lambda,\sigma,\mu) = \mathbb{G}_{x^N}(x^N) + \sigma\psi_{x_N}(x_N) - \lambda^N, \tag{9}$$

$$\begin{aligned} 0 &= \mathscr{L}_{u^i}(z,\lambda,\sigma,\mu) = h\lambda^{i+1} f_{u^i}(x^i,x^{i-k},u^i,u^{i-l}) + \\ &\qquad h\lambda^{i+l+1} f_{u^i_{\tau_u}}(x^{i+l},x^{i-k+l},u^{i+l},u^i) + \mu^i c_{u^i}(x^i,u^i), \\ &\qquad i=0,\dots,N-l-1, \\ 0 &= \mathscr{L}_{u^i}(z,\lambda,\sigma,\mu) = h\lambda^{i+1} f_{u^i}(x^i,x^{i-k},u^i,u^{i-l}) + \mu^i c_{u^i}(x^i,u^i), \\ &\qquad i=N-l,\dots,N-1, \end{aligned} \tag{10}$$

$$0 = \mathscr{L}_h(s,\lambda,\mu,h) = \phi_h + \sum_{i=0}^{N-1} \mathscr{H}_h(i) + \sum_{i=0}^{N-1} \mu^i c_h(x^i,u^i). \tag{11}$$

Eqs. (8) – (11) represent the discrete version of the necessary condition (2) - (4) for optimal control problem (1).

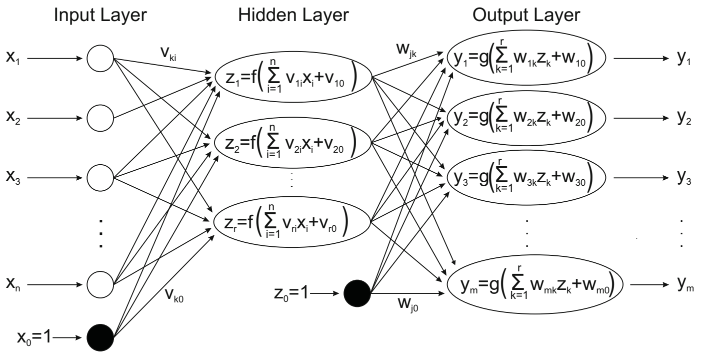

Input Layer
Hidden Layer
Output Layer
x_1
x_2
x_3
x_n
$x_0=1$
v_{ki}
v_{k0}
$z_1=f\left(\sum_{i=1}^{n} v_{1i}x_i+v_{10}\right)$
$z_2=f\left(\sum_{i=1}^{n} v_{2i}x_i+v_{20}\right)$
$z_r=f\left(\sum_{i=1}^{n} v_{ri}x_i+v_{r0}\right)$
$z_0=1$
w_{jk}
w_{j0}
$y_1=g\left(\sum_{k=1}^{r} w_{1k}z_k+w_{10}\right)$
$y_2=g\left(\sum_{k=1}^{r} w_{2k}z_k+w_{20}\right)$
$y_3=g\left(\sum_{k=1}^{r} w_{3k}z_k+w_{30}\right)$
$y_m=g\left(\sum_{k=1}^{r} w_{mk}z_k+w_{m0}\right)$
y_1
y_2
y_3
y_m

Fig. 1 Feed forward neural network topology with one hidden layer, $v_{ki},\ w_{jk}$ are values of connection weights, $v_{k0},\ w_{j0}$ are values of bias, f(.), g(.) are activation functions

4 Adaptive Critic Neural Network for an Optimal Control Problem with Control and State Constraints

It is well known that a neural network can be used to approximate the smooth time-invariant functions and the uniformly time-varying functions [6], [21]. Experience has shown that optimization of functional over admissible sets of functions made up of linear combinations of relatively few basis functions with a simple structure and depending nonlinearly on a set of "inner" parameters e.g., feed-forward neural networks with one hidden layer and linear output activation units often provides surprisingly good suboptimal solutions [1], [8], [15]. Fig. 1 shows a feed forward neural networks with n inputs node, one hidden layer of r units and m output units. Let $x = [x_1,\dots,x_n]'$ and $y = [y_1,\dots,y_m]'$ be the input and output vectors of the network, respectively. Let $V = [v_1,\dots,v_r]'$ be the matrix of synaptic weights between the input nodes and the hidden units, where $v_k = [v_{k0}, v_{k1}\dots,v_{kn}]$; v_{k0} is the bias of the kth hidden unit, and v_{ki} is the weight that connects the ith input node to the kth hidden unit.

Let also $W = [w_1,\dots,w_m]'$ be the matrix of synaptic weights between the hidden and output units, where $w_j = [w_{j0}, w_{j1}\dots,w_{jr}]$; w_{j0} is the bias of the jth output unit, and w_{jk} is the weight that connects the kth hidden unit to the jth output unit. The response of the kth hidden unit is given by $z_k = tanh\left(\sum_{i=0}^{n} v_{ki}x_i\right),\ k = 1,\dots,r,$ where tanh(.) is the activation function for the hidden units. The response of the jth output unit is given by $y_j = \sum_{k=0}^{r} w_{jk}z_k,\ j = 1,\dots,m.$ The multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between the input and output vectors. The number of neurons in the input and output layers is given by the number of input and output variables, respectively. The multi-layered feed forward networks shown in Fig. 2 is trained using the backpropagation algorithm with supervised learning. Basically, it is a gradient descent, a parallel distributed optimization technique to minimize the error between the network and the target output [20] (least mean squared (LMS) algorithm). To

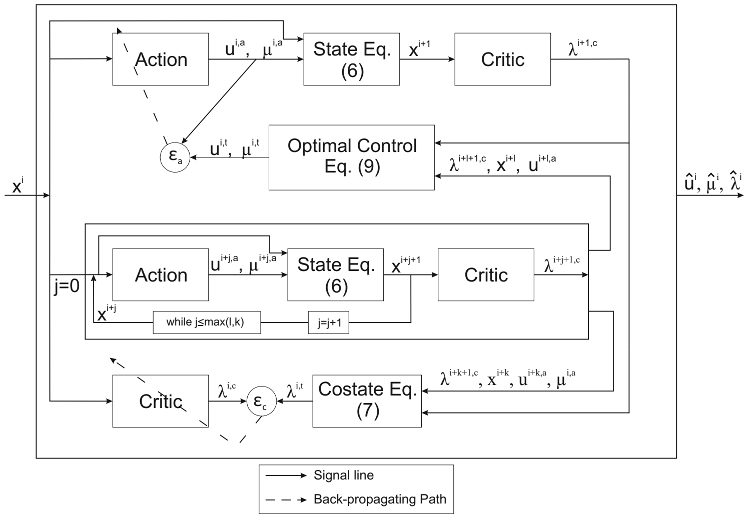


Fig. 2 Architecture of adaptive critic feed forward network synthesis, x^i-input signal to the action and critic network, $\hat{u}^{i,a}, \hat{\mu}^{i,a}$ and $\hat{\lambda}^{i,c}$ are output signal from action and critic network, respectively and $\hat{u}^{i,t}, \hat{\mu}^{i,t}$ and $\hat{\lambda}^{i,t}$ are solutions of equation (10) and co-state equation (8), respectively

solve the equations (10) we are concerned with the following nonlinear projection equation (for detail description see [26]):

$$\alpha\mathscr{F}(\mathscr{P}_X(y)) + y - \mathscr{P}_X(y) = 0, \tag{12}$$

where $\alpha > 0$ is a constant, $\mathscr{F} : R^l \to R^l$, $X = \{y \in R^l \mid y_{imin} \le y_i \le y_{imax}\}$ and $\mathscr{P}_X : R^l \to X$ is a projection operator defined by $\mathscr{P}_X(y) = (\mathscr{P}_X(y_1), \dots, \mathscr{P}_X(y_l))$

$$\mathscr{P}_X(y_i) = \begin{cases} y_{imin} & : \quad y_i < y_{imin} \\ y_i & : \quad y_{imax} \le y_i \le y_{imax} \\ y_{imax} & : \quad y_i > y_{imax}, \end{cases}$$

which can be solved by the following dynamic model

$$\dot{y}(t) = -\beta(\alpha\mathscr{F}(\mathscr{P}_X(y)) + y - \mathscr{P}_X(y)). \tag{13}$$

Note that y_{imin} and y_{imax} are lower and upper limits of y_i, $i = 1, \dots, l$. Asymptotic and exponential stability of the present recurrent neural network (13) are proven in [26]. The equilibrium points of (13) coincide with the solutions of (12). We can state the algorithm to solve the optimal control problem using the adaptive critic and recurrent neural network. In the Pontryagin's maximum principle for deriving an optimal control law, the interdependence of the state, co-state and control dynamics is made clear. Indeed, the optimal control $\hat{u}$ and multiplier $\hat{\mu}$ is given by

Eq. (10), while the co-state Eqs. (8) - (9) evolves backward in time and depends on the state and control. The adaptive critic design based on neural networks [17] is shown in Fig. 2. It consists of two networks at each node: an action network, the inputs for which are the current states and its outputs are the corresponding control $\hat{u}$ and multiplier $\hat{\mu}$, and the critic network for which the current states are inputs and current co-states are outputs for normalizing the inputs and targets (zero mean and standard deviations). For detail explanation see [20]. Based on Fig. 2 the adaptive critic design based on neural networks procedure of the optimal control problem is summarized in Algorithm 1. In the adaptive critic synthesis, the action and critic network were selected such that they consist of $n+m$ subnetworks, respectively, each having $n-3n-1$ structure (i.e. n neurons in the input layer, $3n$ neurons in the hidden layer and one neuron in the output layer). The training procedure for the action and critic networks, respectively are given by [17]. From the free terminal condition ($\psi(x) \equiv 0$) from Eqs. (8) - (9) we obtain that $\lambda_0^i = -1,\, i = N, \dots, 0$ and $\lambda_j^N = 0,\, j = 1, \dots, N$. We use this observation before proceeding to the actual train-

Algorithm 1. Algorithm to solve the optimal control problem.

```
   Input: Choose $t_0$, $t_f$, $N$ - number of steps, time step $h$, $\alpha > 0$, $\beta > 0$, $\varepsilon_a$, $\varepsilon_c$ and $\varepsilon_{rnn}$ -
          stopping tolerance for action, critic and recurrent neural network, respectively,
          $x^{-i} = \phi_s(t_0 - ih),\ i = k, \dots, 0,\ \ u^{-i} = \phi_c(t_0 - ih),\ i = l, \dots, 0$ -initial values.
   Output: Set of final approximate optimal control $\hat{u}(t_0 + ih) = \hat{u}^i$ and optimal trajectory
          $\hat{x}(t_0 + (i+1)h) = \hat{x}^{i+1},\ i = 0, \dots, N-1$, respectively
1  Set the initial weight $\mathscr{W}^a = (V^a, W^a)$, $\mathscr{W}^c = (V^c, W^c)$
   for $i \leftarrow 0$ to $N-1$ do
2      while $err_a \geq \varepsilon_a$ and $err_c \geq \varepsilon_c$ do
3          for $j \leftarrow 0$ to $max(k,l)$ do
4              Compute $u^{i+j,a}$, $\mu^{i+j,a}$ and $\lambda^{i+j+1,c}$ using action ($\mathscr{W}^a$) and critic ($\mathscr{W}^c$)
               neural networks, respectively and $x^{i+j+1}$ by Eq. (7)
5          Compute $\lambda^{i,t}$, $u^{i,t}$, and $\mu^{i,t}$ using Eqs. (8), (10) and (13) with
           $X = \{(u^i, \mu^i) \in R^{m+q} | \mu^i \geq 0\}$,
           $\mathscr{F}(u^i, \mu^i) = (\mathscr{L}_{u^i}(z, \lambda, \sigma, \mu), -c(x^i, u^i))$ and stopping tolerance $\varepsilon_{rnn}$.
6          if $i = N-1$ then
7              $X = \{(u^{N-1}, \mu^{N-1}, \sigma) \in R^{m+q+r} | \mu^{N-1} \geq 0,\}$,
               $\mathscr{F}(u^{N-1}, \mu^{N-1}, \sigma) = (\mathscr{L}_{u^{N-1}}(z, \lambda, \sigma, \mu), -c(x^{N-1}, u^{N-1}), -\psi(x^N))$ with
               $\lambda^N = \mathscr{G}_{x^N}(x^N) + \sigma \psi_{x_N}(x_N)$
8          $err_c = \| \lambda^{i,t} - \lambda^{i,c} \|$
9          $err_a = \| (u,\mu)^{i,t} - (u,\mu)^{i,a} \|$
10         With the data set $x^i$, $\lambda^{i,t}$ update the weight parameters $\mathscr{W}^c$
11         With the data set $x^i$, $(u,\mu)^{i,t}$ update the weight parameters $\mathscr{W}^a$
12         Set $\lambda^{i,c} = \lambda^{i,t}$, $(u,\mu)^{i,a} = (u,\mu)^{i,t}$
13     Set $\hat{\lambda}^i = \lambda^{i,t}$, $(\hat{u}^i, \hat{\mu}^i) = (u,\mu)^{i,t}$
14     Compute $\hat{x}^{i+1}$ using Eq. (7) and $\hat{u}^i$
15     return $\hat{\lambda}^i$, $\hat{u}^i, \hat{\mu}^i$, $\hat{x}^{i+1}$
```

ing of the adaptive critic neural network. Further discussion and detail explanation of these adaptive critic methods can be found in [13], [16], [17] and [20].

5 Discrete Time Delay Continuous Hopfield Neural Network Learning

Hopfield neural nets are fully conected which extended the ideas of linear associative memories by adding cyclic connections [4]. In 1984, Hopfield [5] showed how an analog electrical circuit could behave as a small network of neurons with graded response. He derived a Lyapunov function for the network to check for stability and used it as a content-addressable memory. The differential equations derived by Hopfield for the electrical circuit using Kirchhoffs laws could be reduced to the system of differential equations. We can use a continuous Hopfield net to model a dynamical systems that carries out its computations by the change of its states with time delays. Let us consider a supervised learning to teach discrete time delay dynamic to the discrete time delay continuous Hopfield neural network. In the learning of nonlinear dynamics, the following form of multilayer continuous Hopfield neural network with discrete time delay we utilize:

$$\begin{aligned}\dot{x}(t) &= F(x(t),x(t-\tau),W)\\ &= -Ax(t)+W_of(W_hx(t)+W_{hd}x(t-\tau)),\end{aligned} \tag{14}$$

where $x=(x_1,\dots,x_n)$, $f=(f_1,\dots,f_n)$, A is diagonal matrix, W_o is a weight matrix between hidden and output layer, W_h, W_{hd} are weight matrix between input and hidden layer and $W=(A,W_h,W_{hd},W_o)$. Function f is $tanh(.)$ activation function. For a given continuous initial condition $\phi(t),\ t\in\langle-\tau,0\rangle$ there exists unique solution $x(t)$ satisfying Eq. (14) for $t\in\langle 0,T\rangle$. The aim is to find the weight parameters W that give rise to a solution $x(t)$ approximately following a teacher signal $p(t)$, where $p(t)$ is a solution of the following delay differential equation:

$$\dot{p}(t) = G(x(t),x(t-\tau)). \tag{15}$$

First, the cost function is defined for the weight parameters W as

$$E(W) = \int_0^T \frac{1}{2}\sum_{i=1}^{n}(x_i(t)-p_i(t))^2dt. \tag{16}$$

Then, the cost function (16) is minimized by the steepest descent method

$$w^{j+1} = w^j - \eta\frac{\partial E}{\partial w}(W^j), \tag{17}$$

where $w\in W$. To compute gradient of function (16) we use time-dependent recurrent learning (TDRL) [24]. In the TDRL algorithm the gradients are computed by using the Lagrange multipliers $\lambda(t) = (\lambda_1(t), \ldots, \lambda_n(t))$. For detail explanation see [24]. We can rewrite the cost function $E(W)$ as

$$\mathbb{L}(W) = \int_0^T \frac{1}{2} \sum_{i=1}^{n} \left((x_i(t) - p_i(t))^2 - \lambda_i(t) \left(\dot{x}_i(t) - F_i(x(t), x(t-\tau), W) \right) \right) dt. \tag{18}$$

Partial derivatives with respect to weight coefficients $w \in W$ are calculated as

$$\begin{aligned} \frac{\partial \mathbb{L}}{\partial w} = \int_0^T \sum_{i=1}^{n} & [(x_i(t) - p_i(t)) \frac{\partial x_i(t)}{\partial w} - \lambda_i(t) \frac{\partial \dot{x_i}(t)}{\partial w} + \lambda_i(t) \frac{F_i(x(t), x(t-\tau), W)}{\partial w} \\ & + \lambda_i(t) \sum_{j=1}^{n} \frac{F_i(x(t), x(t-\tau), W)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial w} + \lambda_i(t) \sum_{j=1}^{n} \frac{F_i(x(t), x(t-\tau), W)}{\partial x_j(t-\tau)} \frac{\partial x_j(t-\tau)}{\partial w} \\ & - \frac{\lambda_i(t)}{\partial w} (\dot{x}_i(t) - F_i(x(t), x(t-\tau), W))] dt \end{aligned} \tag{19}$$

If $x(t)$ is a solution of Eq. (14), then the final term of Eq. (19) vanishes. Since $\frac{\partial x(t)}{\partial w} = 0$ for $t \in \langle -\tau, 0 \rangle$ the fourth term of (19) can be written by the transformation $s = t - \tau$ as

$$\begin{aligned} & \int_0^T \sum_{i=1}^{n} \lambda_i(t) \sum_{j=1}^{n} \frac{F_i(x(t), x(t-\tau), W)}{\partial x_j(t-\tau)} \frac{\partial x_j(t-\tau)}{\partial w} dt = \\ & \int_{-\tau}^{T-\tau} \sum_{i=1}^{n} \lambda_i(t+\tau) \sum_{j=1}^{n} \frac{F_i(x(t+\tau), x(t), W)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial w} ds = \\ & \int_0^T \sum_{i=1}^{n} \lambda_i(t+\tau) \sum_{j=1}^{n} \frac{F_i(x(t+\tau), x(t), W)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial w} \chi_{\langle 0, T-\tau \rangle} ds. \end{aligned}$$

The derivatives $\frac{\partial \mathbb{L}}{\partial w}$ become

$$\begin{aligned} \frac{\partial \mathbb{L}}{\partial w} = \int_0^T \sum_{i=1}^{n} & [(x_i(t) - p_i(t)) \frac{\partial x_i(t)}{\partial w} - \lambda_i(t) \frac{\partial \dot{x_i}(t)}{\partial w} + \lambda_i(t) \frac{F_i(x(t), x(t-\tau), W)}{\partial w} \\ & + \lambda_i(t) \sum_{j=1}^{n} \frac{F_i(x(t), x(t-\tau), W)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial w} \\ & + \lambda_i(t+\tau) \sum_{j=1}^{n} \frac{F_i(x(t+\tau), x(t), W)}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial w} \chi_{\langle 0, T-\tau \rangle} dt. \end{aligned} \tag{20}$$

Lagrange multipliers are solutions of the following discrete time delay differential equations with terminal condition $\lambda(T) = 0$.

$$-\dot{\lambda}_i(t) = \sum_{j=1}^{n} \lambda_j(t) \frac{F_j(x(t), x(t-\tau), W)}{\partial x_i(t)} \tag{21}$$

$$+\sum_{j=1}^{n}\lambda_j(t+\tau)\frac{F_j(x(t+\tau),x(t),W)}{\partial x_i(t)}\chi_{\langle 0,T-\tau\rangle}+(x_i(t)-p_i(t)).$$

Since Lagrange multipliers $\lambda(t)$ satisfy Eq. (21) with terminal condition $\lambda(T)=0$ and $\frac{\partial x(t)}{\partial w}=0$, the first, the second, the third and the fourth terms of Eq. (20) vanish. We can state the following algorithm for time dependent recurrent learning.

Algorithm 2. Time dependent recurrent learning algorithm to determine weight matrix of time delay continuous Hopfield neural network.

```
   Input: Choose $T,\ \tau,\ x^T(t)$ - teacher signal, $maxit$, $\varepsilon_E$, - stopping tolerance,
          $\phi(t), t\in\langle-\tau,0\rangle$,  -initial condition.
   Output: Weight matrix $\mathbb{W}=(A,W_o,W_h,W_{dh})$;
 1 Set the initial weight $\mathbb{W}=(A,W_o,W_h,W_{dh})$, $i=0$
   while $err_E\geq\varepsilon_E$ and $i\leq$ maxit do
 2     Compute solution $x(t)$ of Eq. (14) on the interval $\langle 0,T\rangle$ with initial condition
       $\phi(t), t\in\langle-\tau,0\rangle$,
 3     Compute solution $\lambda(t)$ of Eq. (19) on the interval $\langle T,0\rangle$ with terminal condition
       $\lambda(T)=0$
 4     Compute $E(W)$ by Eq. (15)
 5     Compute $\frac{\partial L}{\partial W}=\int_0^T\sum_{i=1}^n\lambda_i(t)\frac{F_i(x(t),x(t-\tau),W)}{\partial W}$ by Eq. (22)
 6     Compute $\alpha^*=min\ g(\alpha)=E\left(W^i-\alpha\frac{\partial J(W^i)}{\partial W}\right)$
 7     Set $W^{i+1}=W^i-\alpha^*\frac{\partial L(W^i)}{\partial W}$
 8     Compute $E(W^{i+1})$ by Eq. (15)
 9     Set $err_E=abs(E(W^{i+1}-E(W^i))$
10 return $\mathbb{W}^{i+1}=(W_o^{i+1},W_h^{i+1},W_{dh}^{i+1})$
```

The partial derivatives $\frac{\partial\mathbb{L}}{\partial w}$ can be calculated by the following form:

$$\begin{aligned}\frac{\partial\mathbb{L}}{\partial a_{ii}}&=\int_0^T x_i(t)\lambda_i(t)dt,\ \ \frac{\partial\mathbb{L}}{\partial w_{ij}^o}=\int_0^T\lambda_i(t)f_j(t)dt\\ \frac{\partial\mathbb{L}}{\partial w_{ij}^h}&=\int_0^T\sum_{k=1}^n\lambda_k(t)w_{ki}^o f_i'(t)x_j(t)dt,\\ \frac{\partial\mathbb{L}}{\partial w_{ij}^{hd}}&=\int_0^T\sum_{k=1}^n\lambda_k(t)w_{ki}^o f_i'(t)x_j(t-\tau)dt,\end{aligned}\tag{22}$$

where $f_j(t)=tanh\left(\sum_{k=1}^n(w_{jk}^h x_k(t)+w_{jk}^{hd}x_k(t-\tau))\right)$.

6 Nitrogen Transformation Cycle

The aerobic transformation of nitrogen compounds [12] includes: Specific groups of microorganisms participate in transformation of nitrogen compounds. Heterotrophic bacteria (x_1) assimilate and decompose the soluble organic nitrogen compounds DON (x_6) derived from detritus (x_5). Ammonium (x_7), one of the final decomposition products undergoes a biological transformation into nitrate (x_9). This is carried out by aerobic chemoautotrophic bacteria in two stages: ammonia is first oxidized by nitrifying bacteria from the genus Nitrosomonas (x_2) into nitrites (x_8) that serve as an energy source for nitrating bacteria mainly from the genus Nitrobacter (x_3). The resulting nitrates may be assimilated together with ammonia and soluble organic forms of nitrogen by the phytoplankton (x_4), whereby the aerobic transformation cycle of nitrogen compounds is formed. The individual variables $x_1,\ldots,x_9$ represent nitrogen concentrations contained in the organic as well as in inorganic substances and living organisms presented in a model. The following system of ordinary differential equations is proposed as a model for the nitrogen transformation cycle:

$$\begin{aligned}
\dot{x}_i(t) &= x_i(t)U_i(x(t)) - x_i(t)E_i(x(t)) - x_i(t)M_i(x(t))),\ i = 1,2,3,\\
\dot{x}_4(t) &= x_4(t-\tau)(U_4(x(t-\tau)) - E_i(x(t-\tau)) - M_i(x(t-\tau))),\\
\dot{x}_5(t) &= \sum_{i=1}^{4} x_i M_i(x) - K_5 x_5(t),\\
\dot{x}_6(t) &= K_5 x_5(t) - x_1(t)U_1(x(t)) + x_4(t)E_4(x(t)) - x_4(t)P_6(x(t)),\\
\dot{x}_7(t) &= x_1(t)E_1(x(t)) - x_2(t)U_2(x(t)) - x_4(t)P_7(x(t)),\\
\dot{x}_8(t) &= x_2(t)E_2(x(t)) - x_3(t)U_3(x(t)),\\
\dot{x}_9(t) &= x_3(t)E_3(x(t)) - x_4(t)P_9(x(t)),
\end{aligned} \tag{23}$$

where $x_i(t)$ are the concentration of the recycling matter in microorganisms, the available nutrients and detritus, respectively. The constant τ stands for the discrete time delay in uptake of nutrients by phytoplankton. Functions occurring in the model are given in Table 1 in ecological and mathematical notation, respectively. Three variables $u = (u(1),u(2),u(3))$ express the preference coefficients for update of x_6, x_7, x_9. It can be expected that the phytoplankton will employ control mechanisms in such a way as to maximize its biomass over a given period t_f of time:

$$J(u) = \int_0^{t_f} x_4(t)dt \to max \tag{24}$$

under the constraint

$$\begin{aligned}
C(x,u) &:= b_1U_4(x,u) + b_2P_6(x,u) + b_3P_9(x,u) + b_4E_4(x,u) \le W(I),\\
u_i &\in [0,u_{imax}]\ \ for\ i = 1,2,3.
\end{aligned} \tag{25}$$

Table 1 Description of functions occurring in the model

$U_i(x)$	$= \frac{K_i x_{i+5}}{1+g_i x_{i+5}}, i=1,2,3$	
p	$= u_1x_6 + u_2x_7 + u_3x_9$	
$U_4(x)$	$= \frac{K_4 p}{1+g_4 p}$	U_i - uptake rate
$L_i(x)$	$= \frac{a_{2i-1}U_i(x)}{1+a_{2i}U_i(x)} + 1 - \frac{a_{2i-1}}{a_{2i}}$	L_i - excretion activity
$M_i(x)$	$= g_{2i+3} + g_{2i+4}L_i(x)$	M_i - mortality rate
$E_i(x)$	$= U_i(x)L_i(x), i=1,\dots,4$	E_i - excretion rate
$P_i(x)$	$= \frac{K_4 u_i x_i}{1+g_4 p}, \ i=6,7,9$	P_i - uptake rate.

The last inequality expresses the fact that amount of energy used for "living expenses" (synthesis, reduction and excretion of nutrients) by phytoplankton cannot exceed a certain value $W(I)$ which depends on light intensity I (for detail explanation see [12]). We are led to the following optimal control problems:
(1) instantaneous maximal biomass production with respect to u:

$$\dot{x}_4 = x_4(U_4(x,u) - E_4(x,u) - M_4(x,u)) \to max \tag{26}$$

under the constraint $C(x,u) \le W(I)$, for all $t \in [t_0,t_f]$ and $u_i \in [0,u_{imax}]$, i=1,2,3 (To maximize (26) is equivalent to find the maximum of the function

$$p(u) = u_1x_6 + u_2x_7 + u_3x_9$$

under the constraint $C(x,u) \le W(I)$, $u_i \in [0,u_{imax}]$ for i=1,2,3.),

(2) global maximal biomass production with respect to u:

$$J(u) = \int_{t_0}^{t_f} x_4(t)dt \to max \tag{27}$$

under the constraint $C(x,u) \le W(I)$, for all $t \in [t_0,t_f]$ and $u_i \in [u_{imin},u_{imax}]$ for i=1,2,3. We introduce an additional state variable

$$x_0(t) = \int_0^t x_4(s)ds. \tag{28}$$

We are led to the following optimal control problem: Maximize

$$x_0(t_f) \tag{29}$$

under the constraints

$$c_1(x,u) = C(x,u) - W(I), \le 0$$

$$c_{i+1}(x,u) = u_{imin} - u_i \leq 0,$$
$$c_{i+4}(x,u) = u_i - u_{imax} \leq 0, \; i = 1,2,3.$$

Discretization of Eqs. (23) and (24) using Eqs. (8) – (9) and (6) leads to minimize

$$-x_0^N$$

subject to

$$\begin{aligned} x^{i+1} &= x^i + hF(x^{i-k}, x^i, u^i, u^{i-l}), \; i = 0, \ldots, N-1, \\ \lambda^i &= \lambda^{i+1} + h\lambda^{i+1} F_{x^i}(x^{i-k}, x^i, u^i, u^{i-l}) + \\ &\quad h\lambda^{i+k+1} F_{x^i_{\tau_x}}(x^{i+k}, x^i, u^{i+k}, u^{i-l+k}) + \mu^i c_{x^i}(x^i, u^i), \end{aligned} \tag{30}$$

$$\begin{aligned} \lambda_0^i &= -1, \; i = 0, \ldots, N-1, \\ \lambda^N &= (-1,0,0,0,0,0,0,0,0,0), \end{aligned} \tag{31}$$

$$\begin{aligned} 0 &= h\lambda^{i+1} F_{u^i}(x^{i-k}, x^i, u^i, u^{i-l}) + \\ &\quad h\lambda^{i+l+1} F_{u^i_{\tau_u}}(x^{i+l}, x^{i-k+l}, u^{i+l}, u^i) + \mu^i c_{u^i}(x^i, u^i), \end{aligned}$$

where the vector function $F(x,u) = (-x_4, F_1(x,u), \ldots, F_9(x,u))$ is given by Eq. (24) and by right-hand side of Eq. (23).

6.1 Numerical Simulation

The solution of optimal control problem (29) with state and control constraints using adaptive critic neural network and NLP methods are displayed in Fig. 3. In the adaptive critic synthesis, the critic and action network were selected such that they consist of nine and four subnetworks, respectively, each having 9-27-1 structure (i.e. nine neurons in the input layer, twenty seven neurons in the hidden layer and one

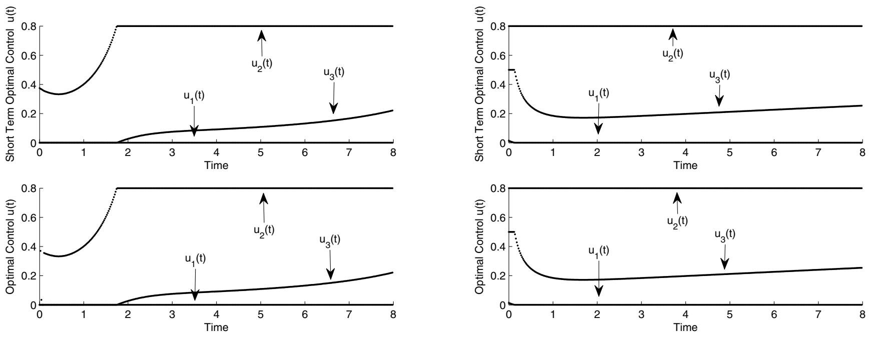


Fig. 3 Adaptive critic neural network simulation of optimal control $\hat{u}(t)$ and $\bar{u}(t)$ with initial condition $\psi_s(t) = (0.1, 0.1, 0.2, 0.8, 0.4, 0.5, 0.6, 0.7, .1)$ for $t \in [-1,0]$

neuron in the output layer). The proposed neural network is able to meet the convergence tolerance values that we choose, which led to satisfactory simulation results. Simulations show that there is a very good agreement between short-term and long-term strategy and proposed neural network is able to solve nonlinear optimal control problem with state and control constraints. The optimal strategy is the following. In the presence of high ammonium concentration, the uptake of DON and nitrate is stopped. If the concentration of ammonium drops below a certain limit value, phytoplankton start to assimilate DON or nitrate dependently on values b_2, b_3. If the

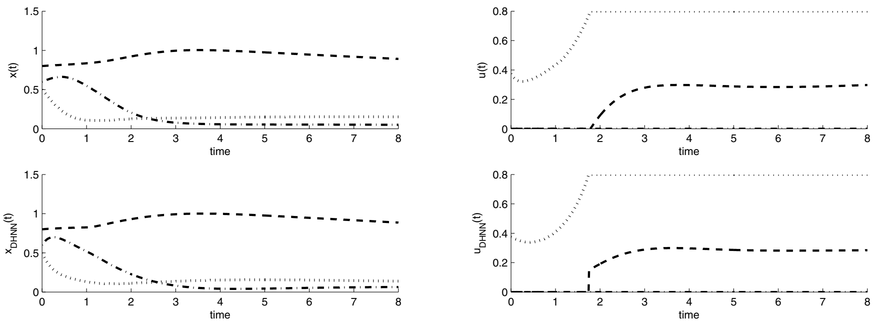


Fig. 4 Optimal trajectory $\bar{x}(t)$ and its continuous Hopfield neural network approximation $x_{DHNN}(t)$ with initial condition $\psi(t) = (0.1, 0.1, 0.2, 0.8, 0.4, 0.5, 0.6, 0.7, .1)$, for $t \in [-1, 0]$ ($\bar{x}_4(t)$, $x_{DHNN4}(t)$ - dash-dash line, $\bar{x}_6(t)$, $x_{DHNN6}(t)$ - dotted line, $\bar{x}_7(t)$, $x_{DHNN7}(t)$ - dash-dot line, optimal control $\bar{u}(t)$ and its continuous Hopfield neural network approximation $u_{DHNN}(t)$ ($\bar{u}_1(t)$, $u_{DHNN4}(t)$ - dash-dash line, $\bar{u}_2(t)$, $u_{DHNN2}(t)$ - dotted line, $\bar{u}_3(t)$, $u_{DHNN3}(t)$ - dash-dot line)

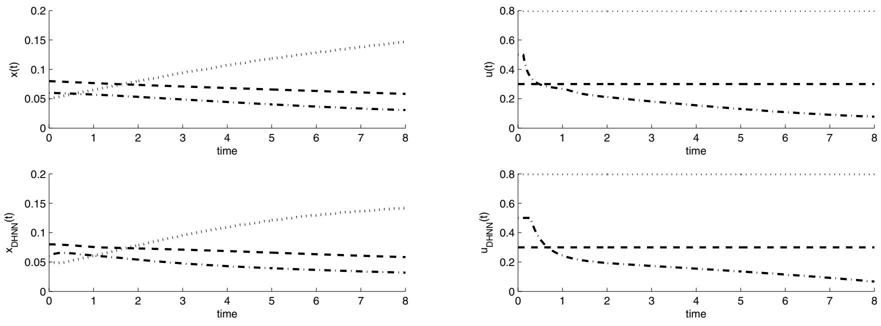


Fig. 5 Optimal trajectory $\bar{x}(t)$ and its continuous Hopfield neural network approximation $x_{DHNN}(t)$ with initial condition $\psi(t) = (0.01, 0.01, 0.02, 0.08, 0.04, 0.05, 0.06, 0.07, .01)$, for $t \in [-1, 0]$ ($\bar{x}_4(t)$, $x_{DHNN4}(t)$ - dash-dash line, $\bar{x}_6(t)$, $x_{DHNN6}(t)$ - dotted line, $\bar{x}_7(t)$, $x_{DHNN7}(t)$ - dash-dot line, optimal control $\bar{u}(t)$ and its continuous Hopfield neural network approximation $u_{DHNN}(t)$ ($\bar{u}_1(t)$, $u_{DHNN4}(t)$ - dash-dash line, $\bar{u}_2(t)$, $u_{DHNN2}(t)$ - dotted line, $\bar{u}_3(t)$, $u_{DHNN3}(t)$ - dash-dot line)

concentration of all three forms of nitrogen are low, all of them are assimilated by phytoplankton at the maximal possible rate, e.i. $\hat{u}_i(t) = u_{imax}$ for all $t \in [t_0, t_f]$ (Figure 3). Our results are quite similar to those obtained in [12] by using Pontryagins maximum principle.

Optimal trajectories found by short term strategy were used as teacher signals for continuous Hopfield neural network to find weight matrix $\mathbb{W} = (A, W_o, W_h, W_{dh})$. Numerical solutions are shown in Figs. 4, 5. It follows from Figs. 4 that proposed discrete time delay continuous Hopfield neural network is able to approximate discrete time delay differential equations signals.

7 Conclusion

The purpose of the paper is twofold. Firstly, a single new network adaptive critic approach is presented for optimal control synthesis with discrete time delay in state and control variables subject to control and state constraints. Using Euler's methods the optimal control problem is transcribed into a discrete-time high-dimensional nonlinear programming problem. Adaptive critic design based on neural networks and the iterative programming algorithm were developed to seek for the state, costate and control variables of the constrained optimal control problem with time delay. These approach is applicable to wide class of nonlinear systems. Simulation studies have demonstrated with an optimal control problems related to nitrogen transformation cycle including phytoplankton production. Using MATLAB, a simple simulation model based on adaptive critic neural network was constructed. Numerical simulations have shown that adaptive critic neural network is able to solve nonlinear optimal control problem with discrete time delay and with control and state constraints.

The second goal is to develop efficient learning algorithm for discrete time delay continuous Hopfield neural network to approximate general discrete time delay differential equations signals. Simulations, using MATLAB show that time-dependent recurrent learning is able to approximate discrete time delay differential equations.

Acknowledgements. The paper was worked out as a part of the solution of the scientific project number KEGA 010UJS-4/2014 and VEGA 1/0699/12.

References

1. Barron, A.R.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory 39, 930–945 (1993)
2. Bryson Jr., A.E.: Dynamic Optimization. Addison-Wesley Longman Inc., New York (1999)
3. Buskens, C., Maurer, H.: SQP-methods for solving optimal control problems with control and state constraints: adjoint variable, sensitivity analysis and real-time control. Journal of Computational and Applied Mathematics 120, 85–108 (2000)

4. Hopfield, J.J.: Neural Networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. 79, 2554–2554 (1982)
5. Hopfield, J.J.: Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Nat. Acad. Sci. 81, 3088–3092 (1984)
6. Hornik, M., Stichcombe, M., White, H.: Multilayer feed forward networks are universal approximators. Neural Networks 3, 256–366 (1989)
7. Hrinca, I.: An Optimal Control Problem for the Lotka-Volterra System with Delay. Nonlinear Analysis, Theory, Methods, Applications 28, 247–262 (1997)
8. Gnecco, A.: A Comparison Between Fixed-Basis and Variable-Basis Schemes for Function Approximation and Functional Optimization. Journal of Applied Mathematics 2012, article ID 806945 (2012)
9. Gollman, L., Kern, D., Mauer, H.: Optimal control problem with delays in state and control variables subject to mixed control-state constraints. Optim. Control Appl. Meth. 30, 341–365 (2009)
10. Chai, Q., Loxton, R., Teo, K.L., Yang, C.: A class of optimal state-delay control problems. Nonlinear Analysis: Real World Applications 14, 1536–1550 (2013)
11. Kirk, D.E.: Optimal Control Theory: An Introduction. Dover Publications, Inc., Mineola (1989)
12. Kmet, T.: Material recycling in a closed aquatic ecosystem. I. Nitrogen transformation cycle and preferential utilization of ammonium to nitrate by phytoplankton as an optimal control problem. Bull. Math. Biol. 58, 957–982 (1996)
13. Kmet, T.: Neural network solution of optimal control problem with control and state constraints. In: Honkela, T. (ed.) ICANN 2011, Part II. LNCS, vol. 6792, pp. 261–268. Springer, Heidelberg (2011)
14. Kmet, T., Kmetova, M.: Adaptive Critic Neural Network Solution of Optimal Control Problems with Discrete Time Delays. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 483–494. Springer, Heidelberg (2013)
15. Makozov, Y.: Uniform approximation by neural networks. Journal of Approximation Theory 95, 215–228 (1998)
16. Padhi, R., Unnikrishnan, N., Wang, X., Balakrishnan, S.N.: Adaptive-critic based optimal control synthesis for distributed parameter systems. Automatica 37, 1223–1234 (2001)
17. Padhi, R., Balakrishnan, S.N., Randoltph, T.: A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems. Neural Networks 19, 1648–1660 (2006)
18. Polak, E.: Optimization Algorithms and Consistent Approximation. Springer, Heidelberg (1997)
19. Pontryagin, L.S., Boltyanskii, V.G., Gamkrelidze, R.V., Mischenko, E.F.: The Mathematical Theory of Optimal Process. Nauka, Moscow (1983) (in Russian)
20. Rumelhart, D.F., Hinton, G.E., Wiliams, R.J.: Learning internal representation by error propagation. In: Rumelhart, D.E., McClelland, D.E. (eds.) PDP Research Group: Parallel Distributed Processing: Foundation, vol. 1, pp. 318–362. The MIT Press, Cambridge (1987)
21. Sandberg, E.W.: Notes on uniform approximation of time-varying systems on finite time intervals. IEEE Transactions on Circuits and Systems-1: Fundamental Theory and Applications 45, 305–325 (1998)
22. Sharif, H.R., Vali, M.A., Samat, M., Gharavisi, A.A.: A New Algorithm for Optimal Control of Time-Delay Systems. Applied Mathematical Science 5, 595–606 (2011)

23. Sun, D.Y., Huang, T.C.: A solutions of time-delayed optimal control problems by the use of modified line-up competition algorithm. Journal of the Taiwan Institute of Chemical Engineers 41, 54–64 (2010)
24. Tokuda, I., Tokunaga, R., Aihara, K.: Back-propagation learning of infinite-dimensional dynamical systems. Neural Networks 16, 1179–1193 (2003)
25. Werbos, P.J.: Approximate dynamic programming for real-time control and neural modelling. In: White, D.A., Sofge, D.A. (eds.) Handbook of Intelligent Control: Neural Fuzzy, and Adaptive Approaches, pp. 493–525 (1992)
26. Xia, Y., Feng, G.: A New Neural Network for Solving Nonlinear Projection Equations. Neural Network 20, 577–589 (2007)

Applying Prototype Selection and Abstraction Algorithms for Efficient Time-Series Classification

Stefanos Ougiaroglou*, Leonidas Karamitopoulos, Christos Tatoglou, Georgios Evangelidis, and Dimitris A. Dervos

Abstract. A widely used time series classification method is the single nearest neighbour. It has been adopted in many time series classification systems because of its simplicity and effectiveness. However, the efficiency of the classification process depends on the size of the training set as well as on data dimensionality. Although many speed-up methods for fast time series classification have been proposed and are available in the literature, state-of-the-art, non-parametric prototype selection and abstraction data reduction techniques have not been exploited on time series data. In this work, we present an experimental study where known prototype selection and abstraction algorithms are evaluated both on original data and a dimensionally reduced representation form of the same data from seven popular time series datasets. The experimental results demonstrate that prototype selection and abstraction algorithms, even when applied on dimensionally reduced data, can effectively reduce the computational cost of the classification process and the storage requirements for the training data, and, in some cases, improve classification accuracy.

1 Introduction

Classification methods based on similarity search have been proven to be effective for time series data analysis. More specifically, the one-Nearest Neighbour (1-NN) classifier is a widely-used method. It works by assigning to an unclassified time series the class label of its most similar training time series. The main drawback

Stefanos Ougiaroglou · Georgios Evangelidis
Department of Applied Informatics, School of Information Sciences,
University of Macedonia, 156 Egnatia St, GR-54006, Thessaloniki, Greece
e-mail: {stoug,gevan}@uom.gr

Leonidas Karamitopoulos · Christos Tatoglou · Dimitris A. Dervos
Information Technology Department, Alexander TEI of Thessaloniki,
GR-57400 Sindos, Greece
e-mail: lkaramit@otenet.gr, xtatty@gmail.com, dad@it.teithe.gr

* S. Ougiaroglou is supported by the State Scholarships Foundation of Greece (I.K.Y.).

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_16

of similarity-based classifiers is that all similarities between an unclassified time series item and the training time series items must be computed. For large and high dimensional time series training sets, the high computational cost involved renders the application of such classifiers prohibitive. Time series classification performance can be improved through indexing, representation and/or data reduction.

Indexing accelerates classification, but works well only in low dimensionality spaces. Thus, one must first use a dimensionality reduction technique to acquire a representation of the original data in lower dimensions. A representation may be considered as a transformation technique that maps a time series from the original space to a feature space, retaining the most important features. There have been several time series representations proposed in the literature, mainly for the purpose of reducing the intrinsically high dimensionality of time series [12].

The main goal of data reduction is to reduce the computational cost of the k-NN classifier and the storage requirements of the training set. Data Reduction Techniques (DRTs)[1] [31, 14, 20, 30, 33, 18, 16, 7, 21] build a small representative set of the initial training data. This set is called the condensing set and has the benefits of low computational cost and storage requirements while keeping the accuracy at high levels. DRT algorithms may be grouped into two categories: (i) Prototype Selection (PS) [14], and, (ii) Prototype Abstraction (PA) (or generation) [31]. Both categories share the same motivation. However, they differ on the way the condensing set is constructed. PS algorithms select some training items and use them as representatives, whereas, PA algorithms generate new item representatives by summarizing on similar training items.

Data reduction has recently been exploited for fast time series classification. More specifically, [8] and [34] propose PS algorithms for speeding-up 1-NN time series classification. The disadvantage of these methods is that they are parametric. The user must define the size of the condensing set by trial-and-error.

The present work has been motivated by the following two observations: (a) to the best of our knowledge, state-of-the-art non-parametric PS and PA algorithms have not been evaluated neither on original time series nor on their reduced dimensionality representations, and, (b) PA algorithms that we have proposed (RHC [24, 23], AIB2 [25, 22]) have not been evaluated on time series data. The contribution of this paper is the experimental evaluation of two PS algorithms, namely, CNN-rule [17] and IB2 [3, 2], and three PA algorithms, namely, RSP3 [28], RHC [24] and AIB2 [25, 22]. The algorithms are evaluated both against original time series datasets and their reduced dimensionality representations.

Our study adopts the Piecewise Aggregate Approximation (PAA) [19, 35] time series representation method. The goal is to investigate the degree to which classification accuracy gets affected when applying data reduction on dimensionally reduced time series. PAA is an effective and very simple dimensionality reduction technique that segments a time series into h consecutive sections of equal-width and calculates the corresponding mean for each section. The series of these means is the new representation of the original data.

[1] One can claim that dimensionality reduction is also data reduction. However, we consider DRTs only from the item reduction point of view.

The rest of the paper is organized as follows. Section 2 discusses the details of the five aforementioned DRTs. Section 3 presents the experimental setup and the results obtained, and Section 4 concludes the paper.

2 Data Reduction Techniques

In this section, we present the five DRTs used in our experimentation. They are based on a simple idea: data items that do not represent decision boundaries between classes are useless for the classification process. Therefore, they can be discarded. The idea is that the k-NN classifier achieves similar accuracy using either the training set or the condensing set. However, condensing set scanning is more efficient than training set scanning. Consequently, DRTs try to select or generate a sufficient number of items that lie in data areas close to decision boundaries. The DRTs we deal with in this section are non-parametric. They automatically determine the size of the condensing set based on the level of noise and the number of classes in the data (the more the classes, the more boundaries exist and, thus, the more items get selected or generated). Therefore, expensive trial-end-error procedures for parameter tuning are avoided.

2.1 Prototype Selection Algorithms

2.1.1 Hart's Condensing Nearest Neighbour Rule (CNN-Rule)

CNN-rule [17] is the earliest and the best known PS algorithm. It uses two sets, CS and TS. Initially, a training item is placed in CS, while all the other training items are placed in TS. Then, CNN-rule tries to classify the content of TS by using the 1-Nearest Neighbour (1-NN) classifier on the content of CS. When an item is misclassified, it is considered to lie in a data area close to decision boundaries. Thus, it is transferred from TS to CS. The algorithm terminates when there are no transfers from TS to CS during a complete pass of TS. The final instance of set CS constitutes the condensing set.

Algorithm 1 presents the pseudo-code of CNN rule: it starts with a training set TS and returns a condensing set CS. Initially, CS has only a training item (lines 1 and 2). Then, for each training item $x \in TS$ (line 5), the algorithm retrieves and examines the class label of its nearest neighbour (line 6). If the class label of x differs from that of its nearest neighbour (line 7), x is moved to CS (lines 8–9). The repeat-until loop terminates when there are no more TS items to migrate from TS to CS (lines 4,10,13).

The multiple passes on data ensure that the remaining (discarded) items in TS can be correctly classified by applying the 1-NN classifier on the condensing set. The algorithm is based on the following simple idea: items that are correctly classified by 1-NN, are considered to lie in a central-class data area and thus, they are ignored.On the other hand, items that are misclassified, are considered to lie in a close-class-border data area, and thus, they are placed in the condensing set. The weak point of

Algorithm 1. CNN-rule

```
Input: $TS$
Output: $CS$
 1: $CS \leftarrow \varnothing$
 2: pick an item of $TS$ and move it to $CS$
 3: repeat
 4:     $stop \leftarrow TRUE$
 5:     for each $x \in TS$ do
 6:         $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
 7:         if $NN_{class} \neq x_{class}$ then
 8:             $CS \leftarrow CS \cup \{x\}$
 9:             $TS \leftarrow TS - \{x\}$
10:             $stop \leftarrow FALSE$
11:         end if
12:     end for
13: until $stop == TRUE$
14: discard $TS$
15: return $CS$
```

the CNN-rule is that the resulting condensing set depends on the ordering by which training set items are considered. This means that different condensing sets may be constructed by considering the same training set data in a different order.

There are many other condensing algorithms that either extend the CNN-rule, or they are based on the same idea. Some of the these algorithms are the Reduced Nearest Neighbour (RNN) rule [15], the Selective Nearest Neighbour (SNN) rule [27], the Modified CNN rule [11], the Generalized CNN rule [10], the Fast CNN algorithms [4, 5], Tomek's CNN rule [29], the Patterns with Ordered Projection (POP) algorithm [26, 1], the recently proposed Template Reduction for *k*-NN (TR*k*NN) [13] and the IB2 algorithm [3, 2].

2.1.2 IB2

IB2 belongs to the well-known family of Instance-Based Learning (IBL) algorithms [3, 2] and is based on the CNN-rule. In effect, IB2 constitutes a simple one pass variation of the CNN-rule. Algorithm 2 presents IB2 in pseudo-code. Each training item $x \in TS$ is classified using 1-NN classifier on the current CS (line 4). If x is classified correctly, it is discarded (line 8). Otherwise, x is transferred to CS (line 6).

Contrary to the CNN-rule, IB2 does not ensure that all discarded items can be correctly classified by the final version of the condensing set. However, since it is a one-pass algorithm, it is very fast, i.e., it involves low preprocessing computational cost. In addition, IB2 builds its condensing set incrementally. New training items can be taken into consideration after the creation of the condensing set. Therefore, IB2 is appropriate for dynamic/streaming environments whereby new training items arrive in an one-by-one fashion. Certainly, IB2 can not deal with data streams with

Algorithm 2. IB2

```
Input: $TS$
Output: $CS$
1:  $CS \leftarrow \varnothing$
2:  an item is chosen at random to migrate from $TS$ to $CS$
3:  for each $x \in TS$ do
4:      $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
5:      if $NN_{class} \neq x_{class}$ then
6:          $CS \leftarrow CS \cup \{x\}$
7:      end if
8:      $TS \leftarrow TS - \{x\}$
9:  end for
10: return $CS$
```

concept drift [32]. IBL-DS [6] adopts the idea of the family of IBL algorithms and can deal with such data. It is worth mentioning that, contrary to the CNN-rule and to many other DRTs, IB2 does not require that all training data reside in main memory. Therefore, it can be applied in devices whose memory is insufficient for storing all the training data. Of course, like the CNN-rule, IB2 is a data ordering dependent algorithm.

2.2 Prototype Abstraction Algorithms

2.2.1 Abstraction IB2 (AIB2)

The AIB2 algorithm constitutes a PA variation of IB2. Therefore, it inherits all the aforementioned properties of IB2. The idea behind AIB2 is quite simple: prototypes should be at the center of the data area they represent. Therefore, the correctly classified items are not ignored. In effect, they contribute to the final condensing set by repositioning their nearest prototype. This is achieved by adopting the concept of prototype weight. Each prototype is characterized by a weight value. It denotes the number of items it represents.

Algorithm 3 presents the pseudo code of the algorithm. Initially, the condensing set (*CS*) has only one item whose weight is initialized to one (lines 1–3). For each training item *x*, AIB2 retrieves from the current *CS* its nearest prototype *NN* (line 5). If *x* is misclassified, it is placed in *CS* and its weight is initialized to one (lines 6–8). Otherwise, the attributes of *NN* are updated by taking into account its current weight and the attributes of *x*. In effect, *NN* "moves" towards *x* (lines 10–12). Finally, the weight of *NN* is increased by one (line 13) and *x* is removed (line 15).

AIB2 aims at improving the efficiency of IB2 by building a condensing set with better prototypes than IB2. Each prototype lies close to the center of the data area it represents. Therefore AIB2 is able to achieve higher classification accuracy. Moreover, the repositioned prototypes reduce the items placed in the final condensing set.

Algorithm 3. AIB2

```
Input: $TS$
Output: $CS$
 1: $CS \leftarrow \varnothing$
 2: move a random item $y$ from $TS$ to $CS$
 3: $y_{weight} \leftarrow 1$
 4: for each $x \in TS$ do
 5:     $NN \leftarrow$ Nearest Neighbour of $x$ in $CS$
 6:     if $NN_{class} \neq x_{class}$ then
 7:         $x_{weight} \leftarrow 1$
 8:         $CS \leftarrow CS \cup \{x\}$
 9:     else
10:         for each attribute $attr(i)$ do
11:             $NN_{attr(i)} \leftarrow \frac{NN_{attr(i)} \times NN_{weight} + x_{attr(i)}}{NN_{weight}+1}$
12:         end for
13:         $NN_{weight} \leftarrow NN_{weight} + 1$
14:     end if
15:     $TS \leftarrow TS - \{x\}$
16: end for
17: return $CS$
```

Hence, AIB2 can achieve higher reduction rates and even lower preprocessing cost than IB2.

2.2.2 RSP3

The RSP3 algorithm belongs to the popular family of Reduction by Space Partitioning (RSP) algorithms [28]. This family includes three PA algorithms. All of them are based on the idea of the early PA algorithm of Chen and Jozwik [9]. Chen and Jozwik's Algorithm (CJA) works as follows: First, the most distant items A and B of the training set that define its diameter are retrieved. Then, the training set is divided into two subsets, S_A and S_B. S_A includes training items that lie closer to A, whereas, S_B includes training items that lie closer to B. Then, CJA proceeds by selecting to divide subsets that include items of more than one classes (non-homogeneous subsets). The subset with the largest diameter is divided first. If all subsets are homogeneous, CJA divides the largest homogeneous subset. This procedure continues until the number of subsets becomes equal to a user specified value. In the end, for each subset S, CJA averages the items in S and creates a mean item that is assigned the label of the majority class in S. The created mean items constitute the final condensing set.

Algorithm 4 lists in pseudo-code a possible implementation of CJA. It accepts a training set (TS) and the number of prototypes n that will be generated. The algorithm uses a data structure to store the created subsets. Initially, the entire TS is stored in S (line 2). Then, the non-homogeneous subset C with the largest diameter is divided into two subsets (lines 4,8). If all subsets are homogeneous, CJA divides the homogeneous subset *C* with the largest diameter (lines 5–7). Both subsets are added to *S*, while *C* is removed (lines 9–11). The procedure for constructing subsets continues until *n* subsets have been created (line 3). The last step is the mean computation (or prototype generation) for each subset and its inclusion in the condensing set (*CS*) (lines 13–18).

Algorithm 4. CJA

```
Input: $TS$, $n$
Output: $CS$
 1: $S \leftarrow \varnothing$
 2: add($S$, $TS$)
 3: for $i = 2$ to $n$ do
 4:     $C \leftarrow$ select the non-homogeneous subset $\in S$ with the largest diameter
 5:     if $C == \varnothing$ {All subsets are homogeneous} then
 6:         $C \leftarrow$ select the homogeneous subset $\in S$ with the largest diameter
 7:     end if
 8:     $(S_x, S_y) \leftarrow$ divide $C$ into two subsets
 9:     add($S$, $S_x$)
10:     add($S$, $S_y$)
11:     remove($S$, $C$)
12: end for
13: $CS \leftarrow \varnothing$
14: for each subset $T \in S$ do
15:     $r \leftarrow$ compute the mean item by averaging the items in $T$
16:     $r.label \leftarrow$ find the most common class label in $T$
17:     $CS \leftarrow CS \cup \{r\}$
18: end for
19: return $CS$
```

CJA selects the next subset that will be divided by examining its diameter. The idea is that a subset with a large diameter probably includes more training items. Therefore, if this subset is divided first, a higher reduction rate will be achieved. A desirable property is that CJA builds the same condensing subset regardless of the ordering of the data in the training set. However, it has two weak points. The first is that the algorithm is parametric. The user has to specify the number of prototypes. This usually involves tuning via a costly trial-end-error procedure. In certain domains, this property may be desirable, since it allows one to control the size of the condensing subset. However, it prohibits the automatic determination of the size of the condensing subset in accordance with the nature of the available data. This constitutes a drawback for the algorithm, in general. The second weakness is that the items that do not belong to the most common class of the subset are not represented in the condensing set. Since the mean item of each subset is labeled by the most common class, items that belong to other classes are practically ignored.

RSP1 deals with the second drawback. More specifically, RSP1 computes as many mean items as the number of different classes in each subset. Therefore, it averages the items that belong to each class in the subset. Of course, RSP1 builds larger CSs than CJA. However, it attempts to improve accuracy since it takes into account all training items.

RSP1 and RSP2 differ on how they select the next subset to be divided. Similar to CJA, RSP1 uses the subset diameter as the splitting criterion. In contrast, RSP2 uses as its splitting criterion the highest overlapping degree. This criterion assumes that items that belong to a specific class lie as close to each other as possible while items that belong to different classes lie as far as possible. According to [28], it is better to divide the subset with the highest overlapping degree. The overlapping degree of a subset is the ratio of the average distance between items belonging to different classes and the average distance between items that belong to the same class.

RSP3 adopts the concept of homogeneity. A subset is homogeneous when it includes items of only a specific class. The algorithm continues dividing the created subsets until all of them become homogeneous. RSP3 can use either the largest diameter or the highest overlapping degree as spiting criterion. Actually, since all non-homogeneous subsets are divided, the choice of splitting criterion becomes an issue of secondary importance.

Algorithm 5 lists the pseudo-code of RSP3. It utilizes a data structure S to hold the unprocessed subsets. Initially, the whole training set (TS) is an unprocessed subset and is put in S (line 2). At each repeat-until iteration, RSP3 selects the subset C with the highest splitting criterion value (line 5) and checks if C is homogeneous or not. If it is homogeneous, the mean item is computed by averaging the items in C and is placed in the condensing set (CS) as a prototype (lines 6–9). Otherwise, C is divided into two subsets D_1 and D_2 (line 11) in the CJA fashion. These new subsets are added to S and C is removed from S (lines 12–14). The repeat-until loop continues until S becomes empty (line 16), i.e., all subsets are homogeneous.

Certainly, RSP3 generates more prototypes for the "close border" data areas and fewer for the "central" data areas. RSP3 is the only non-parametric algorithm of the RSP family (CJA included). It is worth mentioning that like CJA, RSP1 and RSP2, the condensing set built by RSP3 does not depend on the data order in the training set. The procedure for finding the most distant items in each subset is quite expensive. Thus, the main drawback of RSP3 is that it usually involves high preprocessing computation cost. In cases of large datasets, this drawback may render its execution prohibitive.

2.2.3 Reduction through Homogeneous Clusters (RHC)

RHC [24, 23] is also based on the concept of homogeneity. Initially, the whole training set is considered as a non-homogeneous cluster C. RHC begins by computing a mean item for each class (class-mean) in C. Then, it applies k-means clustering on C using the class means as initial means. The clustering procedure builds as many clusters as the number of classes in C. The aforementioned clustering procedure is applied recursively on each non-homogeneous cluster. In the end, the means of the homogeneous clusters are stored in the condensing set as prototypes.

Algorithm 5. RSP3

```
Input: $TS$
Output: $CS$
 1: $S \leftarrow \varnothing$
 2: add($S$, $TS$)
 3: $CS \leftarrow \varnothing$
 4: repeat
 5:     $C \leftarrow$ select the subset $\in S$ with the highest splitting criterion value
 6:     if $C$ is homogeneous then
 7:         $r \leftarrow$ calculate the mean item by averaging the items in $C$
 8:         $r.label \leftarrow$ class of items in $C$
 9:         $CS \leftarrow CS \cup \{r\}$
10:     else
11:         $(D_1, D_2) \leftarrow$ divide $C$ into two subsets
12:         add($S$, $D_1$)
13:         add($S$, $D_2$)
14:         remove($S$, $C$)
15:     end if
16: until IsEmpty($S$)
17: return $CS$
```

Algorithm 6 lists the pseudo-code of RHC. It utilizes a queue data structure, Q, to store clusters. Initially, the training data (TS) is considered as an unprocessed cluster. Therefore, it is placed in Q (line 2). At each one iteration, the algorithm examines the head C of Q (line 5). Then, it checks whether C is a homogeneous cluster or not. If it is homogeneous (line 6), the mean of C is placed in the condensing set (CS) (line 8) and its items are removed. If C is non-homogeneous, the algorithm computes the class means (M): one class mean for each of the classes in C (lines 11–14). Then, RHC calls k-means clustering, with parameters C and M. k-means produces a new set of clusters ($clusters$) (line 15) that are enqueued into Q (lines 16–18). The algorithm stops iterating when Q becomes empty (line 20), i.e., all clusters are homogeneous.

Like RSP3, RHC builds many prototypes for close-class-border data areas and fewer for the non-close-class-border data areas. By using the class means as initial means for the k-means clustering, the algorithm attempts to quickly find homogeneous clusters and achieve high reduction rates (the larger the clusters built, the higher the reduction rates achieved). Moreover, since RHC is based on k-means clustering, it is very fast and can easily be integrated into much of the existing software. In addition, RHC is independent on the ordering of the data in the training set. The results of the experimental study in [24, 23] indicate that RHC achieves higher reduction rates (smaller CSs) and is faster than RSP3 and CNN-rule, while accuracy remains high.

Algorithm 6. RHC

```
Input: $TS$
Output: $CS$
 1: $Q \leftarrow \varnothing$
 2: Enqueue($Q$, $TS$)
 3: $CS \leftarrow \varnothing$
 4: repeat
 5:   $C \leftarrow$ Dequeue($Q$)
 6:   if $C$ is homogeneous then
 7:     $r \leftarrow$ mean of $C$
 8:     $CS \leftarrow CS \cup \{r\}$
 9:   else
10:     $M \leftarrow \varnothing$
11:     for each class $L$ in $C$ do
12:       $m_L \leftarrow$ mean of $L$
13:       $M \leftarrow M \cup \{m_L\}$
14:     end for
15:     $clusters \leftarrow K$-MEANS($C$, $M$)
16:     for each cluster $cl \in clusters$ do
17:       Enqueue($Q$, $cl$)
18:     end for
19:   end if
20: until IsEmpty($Queue$)
21: return $CS$
```

3 Experimental Study

3.1 Experimental Setup

The five DRT algorithms presented were evaluated on seven known time series datasets distributed by the UCR time-series classification/clustering website[2]. Table 1 summarizes on the datasets used. All datasets are available in a training/testing form. We merged the training and testing parts and then we randomized the resulting datasets. No other data transformation was performed. All algorithms were coded in C and as a similarity measure we used the Euclidean distance.

We report on the experiment we conducted with a certain value for the parameter of the PAA representation. We applied the PAA representation on time series by setting the number of dimensions equal to twelve (h=12). Most of the research work provides experimental results with values of h ranging from 2 to 20. We found that lower values of h have a negative effect on classification accuracy, whereas, higher values produce time series that cannot be efficiently indexed by multi-dimensional indexing methods. Hence, we decided to use h=12.

All experiments were run twice, once on the original time series and once on their 12-dimensional representations. We wanted to test how the combination of

[2] http://www.cs.ucr.edu/~eamonn/time_series_data/.

Table 1 Time-series datasets description

Time-series dataset	Size (time-series)	Length (Attr.)	Classes
Synthetic Control (SC)	600	60	6
Face All (FA)	2250	131	14
Two-Patterns (TP)	5000	128	4
Yoga (YG)	3300	426	2
Wafer (WF)	7164	152	2
Sweadish Leaf (SL)	1125	128	15
CBF	930	128	3

data reduction and dimensionality reduction affects the performance of 1-NN classification.

We evaluated the five DRTs by estimating four measurements, namely, accuracy (ACC), classification cost (CC), reduction rate (RR), and, preprocessing cost (PC). Cost measurements were estimated by counting the distance computations multiplied by the number of time series attributes (time series length). Of course, RR and CC measurements relate to each other: the lower the RR, the higher is the CC. However, CC measurements can express the cost introduced by the dimensionality of data. We report on the average values of these measurements obtained via five-cross-fold validation.

3.2 Comparisons

Tables 2 and 3 present the experimental measurements. Table 2 presents the results obtained on the original datasets while table 3 presents the results obtained on the 12-dimensional representations of the datasets we got after applying PAA on them. Both tables include the measurements obtained by applying the 1-NN classifier on the non-reduced data (conventional 1-NN). Each table cell includes the four measurements obtained by first applying a DRT on the original or 12-dimensional time series datasets (preprocessing step) and then by using 1-NN on the resulting condensing set (classification step). The cost measurements are in million (M) distance computations. The PC measurements do not include the small cost overhead introduced by PAA execution.

It is noted that 1-NN classification on the 12-dimensional datasets is very fast. In most cases, the preprocessing and classification cost are extremely low, while classification accuracy remains at high, acceptable levels. Therefore, a first conclusion is that one can obtain efficient time series classifiers by combining prototype selection or abstraction algorithms with time-series dimensionality reduction representations.

It is worth mentioning that the three PA algorithms, RSP3, RHC and AIB2, achieved higher classification accuracy than the conventional 1-NN. In the case of SC dataset, accuracy improvement was very high. Almost in all cases, RSP3 achieved the highest accuracy. However, it is the slowest method in terms of both preprocessing and classification (RSP3 had the lowest reduction rates). The high

Table 2 Experimental results on original datasets

Dataset		Original dimensionality					
		Conv. 1-NN	**CNN**	**IB2**	**RSP3**	**RHC**	**AIB2**
SC	Acc (%):	91.67	90.17	89.00	98.33	98.67	**99.83**
	CC (M):	3.46	0.67	0.53	1.38	**0.09**	0.34
	RR (%):	-	80.50	84.67	60.08	**97.29**	90.13
	PC (M):	-	7.77	1.31	16.22	2.39	**1.14**
FA	Acc (%):	95.07	91.60	91.02	**95.46**	93.02	92.94
	CC (M):	106.11	19.87	18.38	51.65	**12.93**	16.08
	RR (%):	-	81.28	82.68	51.32	**87.81**	84.84
	PC (M):	-	216.36	48.96	533.70	140.41	**43.27**
TP	Acc (%):	**98.50**	94.68	93.60	98.10	93.72	97.06
	CC (M):	512.00	85.66	76.83	243.51	**55.50**	61.88
	RR (%):	-	83.27	85.00	52.44	**89.16**	87.92
	PC (M):	-	1169.75	205.95	2085.42	**150.49**	177.88
YG	Acc (%):	**93.76**	91.58	89.55	92.85	90.94	90.49
	CC (M):	742.26	138.56	108.92	229.82	**93.85**	100.26
	RR (%):	-	81.33	85.33	69.04	**87.36**	86.49
	PC (M):	-	1854.74	254.41	4072.30	**162.61**	240.73
WF	Acc (%):	**99.87**	99.69	99.62	99.82	99.55	99.65
	CC (M):	1248.30	13.59	11.72	26.88	**9.37**	9.71
	RR (%):	-	98.91	99.06	97.85	**99.25**	99.22
	PC (M):	-	165.88	31.42	7196.75	63.69	**25.78**
SL	Acc (%):	52.36	49.87	48.18	52.00	**52.80**	51.56
	CC (M):	25.92	15.94	14.80	19.00	**12.80**	14.65
	RR (%):	-	38.51	42.89	26.69	**50.60**	43.49
	PC (M):	-	112.17	31.39	1537.07	57.01	**31.02**
CBF	Acc (%):	98.39	98.17	97.63	**99.78**	98.60	99.68
	CC (M):	17.71	1.29	1.15	1.97	**0.40**	0.59
	RR (%):	-	92.74	93.49	88.87	**97.74**	96.67
	PC (M):	-	15.06	3.50	78.48	7.26	**2.01**
Avg	Acc (%):	89.94	87.97	86.94	**90.91**	89.62	90.17
	CC (M):	379.40	39.37	33.19	82.03	**26.42**	29.07
	RR (%):	-	79.51	81.87	63.76	**87.03**	84.11
	PC (M):	-	505.96	82.42	2217.13	83.37	7**4.55**

PC measurements are attributed to the costly procedure for finding the most distant items in each created subset (see Subsection 2.2 or [28] for details).

RHC, AIB2 and IB2 had much lower preprocessing cost than the other two methods. This happened because IB2 and AIB2 are one-pass algorithms and RHC is based on a version of *k*-means that is sped-up by the class mean initializations (see Subsection 2.2 or [24] for details). In addition, RHC builds the smallest CSs. In all cases, RHC achieved higher reduction rates than the other DRTs. Thus, the corresponding classifiers had the lowest classification costs.

Table 3 Experimental results on datasets with 12 dimensions

Dataset		12 dimensions					
		Conv. 1-NN	**CNN**	**IB2**	**RSP3**	**RHC**	**AIB2**
SC	Acc (%):	98.50	97.00	95.83	**98.83**	98.17	98.50
	CC (M):	0.69	0.06	0.05	0.12	**0.03**	**0.03**
	RR (%):	-	90.75	93.13	82.96	**95.75**	95.13
	PC (M):	-	0.89	0.13	3.45	0.52	**0.10**
FA	Acc (%):	**87.91**	83.78	82.31	87.07	84.49	84.36
	CC (M):	9.72	2.89	2.53	4.80	**2.08**	2.22
	RR (%):	-	70.23	74.01	50.58	**78.59**	77.21
	PC (M):	-	30.36	5.95	50.91	13.16	**5.30**
TP	Acc (%):	**97.56**	93.52	91.38	96.66	94.34	94.48
	CC (M):	48.00	8.22	6.86	20.42	6.69	**5.39**
	RR (%):	-	82.89	85.72	57.45	86.06	**88.77**
	PC (M):	-	103.86	17.34	196.00	17.63	**14.56**
YG	Acc (%):	**92.36**	90.39	88.03	91.03	90.03	89.67
	CC (M):	20.91	4.41	3.50	6.71	3.13	**3.12**
	RR (%):	-	78.91	83.26	67.90	85.02	**85.06**
	PC (M):	-	52.23	8.04	110.56	**4.26**	7.30
WF	Acc (%):	**99.79**	99.62	99.51	99.40	99.25	99.50
	CC (M):	98.55	1.21	1.01	1.86	1.01	**0.99**
	RR (%):	-	98.77	98.97	98.11	98.97	**99.00**
	PC (M):	-	15.63	2.57	495.63	4.64	**2.44**
SL	Acc (%):	**52.62**	49.07	48.62	51.20	51.20	49.78
	CC (M):	2.43	1.54	1.37	1.78	**1.32**	1.35
	RR (%):	-	36.76	43.67	26.69	**45.69**	44.40
	PC (M):	-	11.33	2.86	56.00	4.99	**2.84**
CBF	Acc (%):	**100.00**	99.57	99.35	99.68	99.57	99.46
	CC (M):	1.66	0.06	0.06	0.12	0.04	**0.04**
	RR (%):	-	96.34	96.56	92.63	97.47	**97.55**
	PC (M):	-	0.66	0.19	7.32	0.70	**0.14**
Avg	Acc (%):	**89.82**	87.57	86.43	89.12	88.15	87.96
	CC (M):	25.99	2.63	2.20	5.12	2.04	**1.88**
	RR (%):	-	79.24	82.19	68.05	**83.94**	83.87
	PC (M):	-	30.71	5.30	131.44	6.56	**4.67**

The classification accuracy achieved by RHC was usually higher than IB2 and CNN-rule and as high as AIB2. In some cases, RHC and AIB2 were more accurate than RSP3. Considering the above, one may conclude that, since RHC ans AIB2 deal with all comparison criteria, they are efficient speed-up methods for time-series data. Finally, the experimental results illustrate that AIB2 is an efficient variation of IB2. In all cases, AIB2 achieves higher performance than IB2.

No DRT can be said to comprise the best speed-up choice. If classification accuracy is the most critical criterion, RSP3 may be preferable. On the other hand, if fast classification and/or fast construction of the condensing set are more critical than accuracy, RHC or AIB2 may be a better choice.

4 Conclusions

Efficient time series classification is an open research issue that attracts the interest of the data mining community. This paper proposes the use of non-parametric state-of-the-art prototype selection and abstraction algorithms for efficient and effective time series classification.

The experimental study conducted demonstrates that by combining prototype selection or abstraction algorithms with dimensionality reduction, one can obtain accurate and very fast time series classifiers. In addition, the study reveals that prototype abstraction algorithms are preferable to prototype selection algorithms when applied on time series data. The prototype abstraction algorithms examined in the study can achieve even higher accuracy than the conventional 1-NN classifier.

References

1. Aguilar, J.S., Riquelme, J.C., Toro, M.: Data set editing by ordered projection. Intell. Data Anal. 5(5), 405–417 (2001), `http://dl.acm.org/citation.cfm?id=1294007.1294010`
2. Aha, D.W.: Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. Int. J. Man-Mach. Stud. 36(2), 267–287 (1992), `http://dx.doi.org/10.1016/0020-73739290018-G`, doi:10.1016/0020-7373(92)90018-G
3. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. Mach. Learn. 6(1), 37–66 (1991), `http://dx.doi.org/10.1023/A:1022689900470`, doi:10.1023/A:1022689900470
4. Angiulli, F.: Fast condensed nearest neighbor rule. In: Proceedings of the 22nd International Conference on Machine Learning, ICML 2005, pp. 25–32. ACM, New York (2005)
5. Angiulli, F.: Fast nearest neighbor condensation for large data sets classification. IEEE Trans. on Knowl. and Data Eng. 19(11), 1450–1464 (2007), `http://dx.doi.org/10.1109/TKDE.2007.190645`, doi:10.1109/TKDE.2007.190645
6. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intell. Data Anal. 11(6), 627–650 (2007), `http://dl.acm.org/citation.cfm?id=1368018.1368022`
7. Brighton, H., Mellish, C.: Advances in instance selection for instance-based learning algorithms. Data Min. Knowl. Discov. 6(2), 153–172 (2002), `http://dx.doi.org/10.1023/A:1014043630878`, doi:10.1023/A:1014043630878
8. Buza, K., Nanopoulos, A., Schmidt-Thieme, L.: Insight: efficient and effective instance selection for time-series classification. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS (LNAI), vol. 6635, pp. 149–160. Springer, Heidelberg (2011)

9. Chen, C.H., Jóźwik, A.: A sample set condensation algorithm for the class sensitive artificial neural network. Pattern Recogn. Lett. 17(8), 819–823 (1996), http://dx.doi.org/10.1016/0167-86559600041-4, doi:10.1016/0167-8655(96)00041-4
10. Chou, C.H., Kuo, B.H., Chang, F.: The generalized condensed nearest neighbor rule as a data reduction method. In: Proceedings of the 18th International Conference on Pattern Recognition, ICPR 2006, vol. 02, pp. 556–559. IEEE Computer Society, Washington, DC (2006), http://dx.doi.org/10.1109/ICPR.2006.1119, doi:10.1109/ICPR.2006.1119
11. Devi, V.S., Murty, M.N.: An incremental prototype set building technique. Pattern Recognition 35(2), 505–513 (2002)
12. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. Proc. VLDB Endow. 1(2), 1542–1552 (2008), http://dl.acm.org/citation.cfm?id=1454159.1454226
13. Fayed, H.A., Atiya, A.F.: A novel template reduction approach for the k-nearest neighbor method. Trans. Neur. Netw. 20(5), 890–896 (2009), http://dx.doi.org/10.1109/TNN.2009.2018547, doi:10.1109/TNN.2009.2018547
14. Garcia, S., Derrac, J., Cano, J., Herrera, F.: Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Trans. Pattern Anal. Mach. Intell. 34(3), 417–435 (2012), http://dx.doi.org/10.1109/TPAMI.2011.142, doi:10.1109/TPAMI.2011.142
15. Gates, G.W.: The reduced nearest neighbor rule. IEEE Transactions on Information Theory 18(3), 431–433 (1972)
16. Grochowski, M., Jankowski, N.: Comparison of instance selection algorithms ii. results and comments. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 580–585. Springer, Heidelberg (2004)
17. Hart, P.E.: The condensed nearest neighbor rule. IEEE Transactions on Information Theory 14(3), 515–516 (1968)
18. Jankowski, N., Grochowski, M.: Comparison of instances seletion algorithms i. algorithms survey. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 598–603. Springer, Heidelberg (2004)
19. Keogh, E.J., Pazzani, M.J.: A simple dimensionality reduction technique for fast similarity search in large time series databases. In: Terano, T., Liu, H., Chen, A.L.P. (eds.) PAKDD 2000. LNCS (LNAI), vol. 1805, pp. 122–133. Springer, Heidelberg (2000)
20. Lozano, M.: Data Reduction Techniques in Classification processes. (Phd Thesis). Universitat Jaume I (2007)
21. Olvera-López, J.A., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F., Kittler, J.: A review of instance selection methods. Artif. Intell. Rev. 34(2), 133–143 (2010), http://dx.doi.org/10.1007/s10462-010-9165-y, doi:10.1007/s10462-010-9165-y
22. Ougiaroglou, S., Evangelidis, G.: Efficient data abstraction using weighted IB2 prototypes. Computer Science and Information Systems 11(2), 665–678 (2014)
23. Ougiaroglou, S., Evangelidis, G.: RHC: Non-parametric cluster-based data reduction for efficient k-nn classification. Pattern Analysis and Applications (accepted, 2014)

24. Ougiaroglou, S., Evangelidis, G.: Efficient dataset size reduction by finding homogeneous clusters. In: Proceedings of the Fifth Balkan Conference in Informatics, BCI 2012, pp. 168–173. ACM, New York (2012), `http://doi.acm.org/10.1145/2371316.2371349`, doi:10.1145/2371316.2371349
25. Ougiaroglou, S., Evangelidis, G.: AIB2: An abstraction data reduction technique based on IB2. In: Proceedings of the 6th Balkan Conference in Informatics, BCI 2013, pp. 13–16. ACM, New York (2013), `http://doi.acm.org/10.1145/2490257.2490260`, doi:10.1145/2490257.2490260
26. Riquelme, J.C., Aguilar-Ruiz, J.S., Toro, M.: Finding representative patterns with ordered projections. Pattern Recognition 36(4), 1009–1018 (2003), `http://www.sciencedirect.com/science/article/pii/S003132030200119X`, doi:`http://dx.doi.org/10.1016/S0031-32030200119-X`
27. Ritter, G., Woodruff, H., Lowry, S., Isenhour, T.: An algorithm for a selective nearest neighbor decision rule. IEEE Trans. on Inf. Theory 21(6), 665–669 (1975)
28. Sánchez, J.S.: High training set size reduction by space partitioning and prototype abstraction. Pattern Recognition 37(7), 1561–1564 (2004)
29. Tomek, I.: Two modifications of cnn. IEEE Transactions on Systems, Man and Cybernetics SMC-6(11), 769–772 (1976), doi:10.1109/TSMC.1976.4309452
30. Toussaint, G.: Proximity graphs for nearest neighbor decision rules: Recent progress. In: 34th Symposium on the INTERFACE, pp. 17–20 (2002)
31. Triguero, I., Derrac, J., Garcia, S., Herrera, F.: A taxonomy and experimental study on prototype generation for nearest neighbor classification. Trans. Sys. Man Cyber Part C 42(1), 86–100 (2012), `http://dx.doi.org/10.1109/TSMCC.2010.2103939`, doi:10.1109/TSMCC.2010.2103939
32. Tsymbal, A.: The problem of concept drift: definitions and related work. Tech. Rep. TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland (2004)
33. Wilson, D.R., Martinez, T.R.: Reduction techniques for instance-basedlearning algorithms. Mach. Learn. 38(3), 257–286 (2000), `http://dx.doi.org/10.1023/A:1007626913721`, doi:10.1023/A:1007626913721
34. Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C.A.: Fast time series classification using numerosity reduction. In: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, pp. 1033–1040. ACM, New York (2006), `http://doi.acm.org/10.1145/1143844.1143974`, doi:10.1145/1143844.1143974
35. Yi, B.K., Faloutsos, C.: Fast time sequence indexing for arbitrary lp norms. In: Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 2000, pp. 385–394. Morgan Kaufmann Publishers Inc., San Francisco (2000), `http://dl.acm.org/citation.cfm?id=645926.671689`

Enforcing Group Structure through the Group Fused Lasso

Carlos M. Alaíz, Álvaro Barbero, and José R. Dorronsoro

Abstract. We introduce the Group Total Variation (GTV) regularizer, a modification of Total Variation that uses the $\ell_{2,1}$ norm instead of the ℓ_1 one to deal with multidimensional features. When used as the only regularizer, GTV can be applied jointly with iterative convex optimization algorithms such as FISTA. This requires to compute its proximal operator which we derive using a dual formulation. GTV can also be combined with a Group Lasso (GL) regularizer, leading to what we call Group Fused Lasso (GFL) whose proximal operator can now be computed combining the GTV and GL proximals through proximal Dykstra algorithm. We will illustrate how to apply GFL in strongly structured but ill-posed regression problems as well as the use of GTV to denoise colour images.

1 Introduction

The irruption of big data, i.e., the need to study problems having very large sample sizes or very large dimensions or both, has resulted in a renewed interest in linear models, either because processing large samples with non-linear models is computationally demanding, or because a large dimension yields rich enough patterns so that methods enlarging pattern dimension such as the kernel trick add marginal value. Among linear models, Mean Squared Error is the simplest fitting function although it is well known that, in order to ensure good generalization, some regularizer has to be added to impose some property or structure in the resultant model, or just because the initial problem may be ill-posed. Classic choices include $\|w\|_2^2$ (this leads to the ridge regression model, which does not impose any particular structure on the solution) and $\|w\|_1$ (resulting in the Lasso [12] model, which enforces sparsity). Recently, more ℓ_1-based regularizers such as Group Lasso [14] or Fused Lasso [13], have been introduced.

Carlos M. Alaíz · Álvaro Barbero · José R. Dorronsoro
Departamento de Ingeniería Informática & Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, 28049 Madrid, Spain
e-mail: {carlos.alaiz,alvaro.barbero,jose.dorronsoro}@uam.es

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_17

From a general point of view all these regularized models can be stated as the problem of finding a $w^* \in \mathbb{R}^M$ which minimizes a certain functional $f(w) = f_{\mathrm{L}}(w) + f_{\mathrm{R}}(w)$ of the weights, with f_{R} the regularization term which somehow bounds the complexity of the model and f_{L} the loss functional. In more detail, assume a training set composed by P input patterns, $\{x^p\}_{p=1}^P$, with $x^p \in \mathbb{R}^M$, and their corresponding targets $\{y^p\}_{p=1}^P$, $y^p \in \mathbb{R}$. If $X \in \mathbb{R}^{P \times M}$ is the matrix having input patterns as rows and $y \in \mathbb{R}^P$ is the target vector, the overall problem for square loss can be written as

$$\min_{w \in \mathbb{R}^M} \{f(w)\} = \min_{w \in \mathbb{R}^M} \{f_{\mathrm{L}}(w) + \lambda f_{\mathrm{R}}(w)\} = \min_{w \in \mathbb{R}^M} \left\{ \|Xw - y\|_2^2 + \lambda f_{\mathrm{R}}(w) \right\}, \quad (1)$$

where λ is a parameter to control the strength of the regularizer.

In what follows, we will explain different choices of f_{R} and the models to which they lead, and we will generalize the Fused Lasso model to a group framework, leading to the Group Fused Lasso model, in which the different groups of coefficients are forced to be sparse using a regularizer à la GL and piece-wise constant through what we call the Group Total Variation (GTV) regularizer, a generalization of the Total Variation of FL. Although this may seem as a strong regularization, such a structure can emerge naturally in problems where the features have some spatial location (and thus coefficients of nearby regions should be similar) and each feature has a multidimensional nature (for example, several signals of the same geographical source). As shown in the experiments, in these cases the GFL model can detect the underlying structure of the data and build a model capturing it.

As a particular case of a multidimensional signal with spatial structure we can consider colour images (or videos), where each pixel has a trivial 2-dimensional spatial position (or 3-dimensional considering the temporal dimension in a video) and a multidimensional nature (the pixels are composed by three variables corresponding to the RGB layers). A pixel is equal to the adjacent one if all the three RGB coefficients are equal, so it is natural to enforce constancy at group levels. Moreover, using the GTV and a distance term to a reference signal we can force a colour image to be near to the observed one but with said smooth structure, thus denoising the observed image.

The structure of this chapter is as follows. First, in Sect. 2 we will review some of the classical structured linear models and define our proposed regularizer, the Group Total Variation, and the corresponding linear model, the Group Fused Lasso. We will show how to solve this model using proximal methods in Sect. 3. We shall illustrate the behaviour of GFL over some examples in Sect. 4, and we will close the chapter in Sect. 5 with a discussion and pointers to further work.

2 Group Fused Lasso and Group Total Variation

A first selection of the regularizer is the ℓ_2 norm, i.e., to penalize the Euclidean norm of the weights with $f_{\mathrm{R}}(w) = \frac{1}{2}\|w\|_2^2$ (also known as Tikhonov regularization). The resultant model is called Regularized Least Squares (RLS), or, alternatively, Ridge Regression. The corresponding optimization problem is:

$$\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2} \|Xw - y\|_2^2 + \frac{\lambda}{2} \|w\|_2^2 \right\},$$

which has a closed-form solution given by:

$$w^* = (X^\top X + \lambda I)^{-1} X^\top y, \tag{2}$$

where $I \in \mathbb{R}^{M \times M}$ denotes the identity matrix.

Taking $f_\mathrm{R}(w) = \|w\|_1 = \sum_{i=1}^M |w_i|$ results in the Lasso approach (LA), which forces some of the coefficients w_i to be identically 0, thus producing an implicit feature selection since only those inputs corresponding to nonzero coefficients have an impact in the model. The optimization problem becomes then:

$$\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2} \|Xw - y\|_2^2 + \lambda \|w\|_1 \right\}.$$

In some problems the features can present a spatial structure which we may want the models to capture. One way to do this is to enforce similarity among the coefficients corresponding to nearby features. If we do not consider any multidimensional feature structure, this can be achieved using a Total Variation (TV) regularizer which penalizes the differences between consecutive coefficients:

$$\mathrm{TV}_1(w) = \sum_{i=2}^M |w_i - w_{i-1}| = \|Dw\|_1,$$

where $D \in \mathbb{R}^{(M-1) \times M}$ is the differencing matrix

$$D = \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix},$$

that is, $D_{i,i} = -1$, $D_{i,i+1} = 1$ and $D_{ij} = 0$ elsewhere. As we are using the ℓ_1 norm of the differences between weights, some of these differences will be identically zero, and thus the value of the weights across those entries will be constant. Such regularizer can be further extended by adding a Lasso-like term to enforce weight sparsity, resulting in the combined regularizer $\lambda f_\mathrm{R}(w) = \lambda_1 \|w\|_1 + \lambda_2 \|Dw\|_1$, whose minimization problem is:

$$\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2} \|Xw - y\|_2^2 + \lambda_1 \|w\|_1 + \lambda_2 \|Dw\|_1 \right\}.$$

The resulting model is called the Fused Lasso (FL).

Neither LA nor FL do consider any possible group structure on the problem features and, therefore, the resulting models will not reflect it even if it may be present. Assume, however, that the pattern features x have such a group structure. We may then see x as a collection of multidimensional features, that is, x has NV components that come in N groups with V features each and therefore has the form

$$x = (\overbrace{x_{1,1}, x_{1,2}, \dots, x_{1,V}}^{x_1}, \overbrace{x_{2,1}, x_{2,2}, \dots, x_{2,V}}^{x_2}, \dots \overbrace{x_{N,1}, x_{N,2}, \dots, x_{N,V}}^{x_N})^\top .$$

The first subscript in $x_{n,v}$ indicates the group (or the multidimensional feature) and the second subscript the group feature so x is decomposed in N blocks $x_n = (x_{n,1}, \dots, x_{n,V})^\top$ that contain V variables.

In this framework, the behaviour of the weights should reflect this structure. In particular, and looking for a similar effect to the one of LA, all the coefficients of a particular group should be zero, or nonzero, at the same time, so the sparsity is achieved at the group level (a multidimensional feature is considered as irrelevant or relevant as a whole, and not each component independently as in the traditional LA model). This behaviour is obtained using a regularization based on the $\ell_{2,1}$ norm, which is defined, for a vector w with the previous structure, as

$$\|w\|_{2,1} = \sum_{n=1}^{N} \|w_n\|_2 = \sum_{n=1}^{N} \sqrt{\sum_{v=1}^{V} w_{n,v}^2},$$

which is just the ℓ_1 norm of the ℓ_2 group norms. This leads to the Group Lasso model (GL) whose regularizer is then $f_{\mathrm{R}}(w) = \|w\|_{2,1}$, and which is solution of the problem:

$$\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2} \|Xw - y\|_2^2 + \lambda \|w\|_{2,1} \right\}.$$

In this work we will extend GL to a fused setting, introducing first a new Group Total Variation regularizer (GTV) defined as:

$$\mathrm{GTV}(w) = \sum_{n=2}^{N} \sqrt{\sum_{v=1}^{V} (w_{n,v} - w_{n-1,v})^2}.$$

This regularizer enforces the differences between consecutive groups to be identically zero, i.e, the coefficients will be piece-wise constant at group level, with $w_{n,v} = w_{n-1,v}$ for $v = 1, \dots, V$. We can consider now a full regularization functional that adds the GTV term defined above to the standard $\ell_{2,1}$ regularizer of GL, which can be written in compact notation using a group differencing matrix $\bar{D} \in \mathbb{R}^{(N-1)V \times NV}$ as:

$$\lambda f_{\mathrm{R}}(w) = \lambda_1 \|w\|_{2,1} + \lambda_2 \|\bar{D}w\|_{2,1}, \quad \text{with } \bar{D} = \begin{pmatrix} -I & I & & & \\ & -I & I & & \\ & & \ddots & \ddots & \\ & & & -I & I \end{pmatrix},$$

and where $I \in \mathbb{R}^{V \times V}$ stands for the identity matrix. Therefore, we arrive to the following optimization problem:

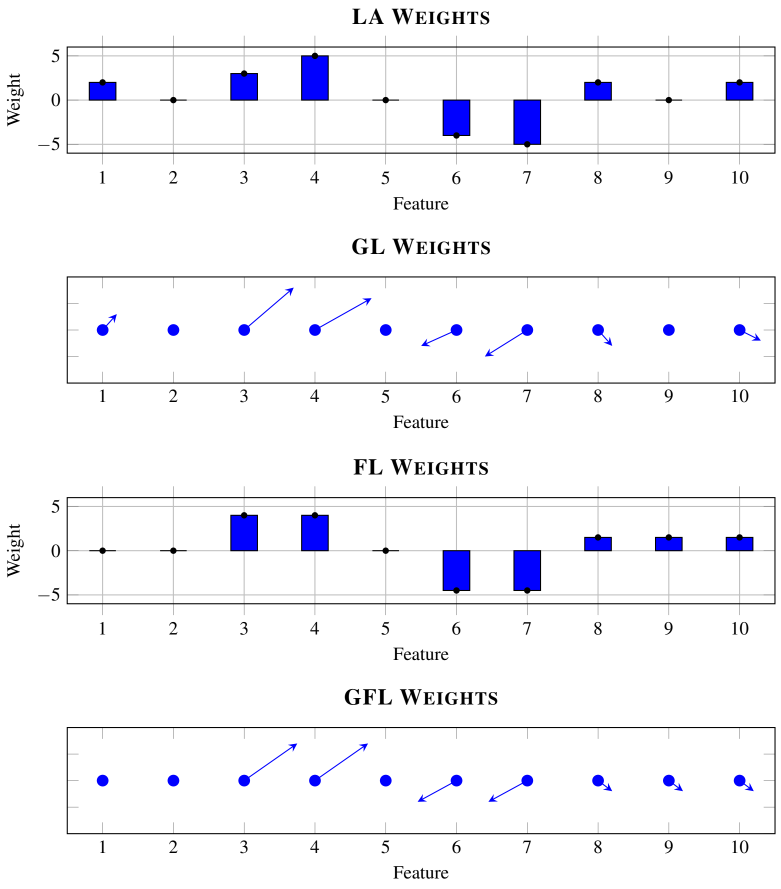


Fig. 1. Example of the structures induced by the different models

$$\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2} \|Xw - y\|_{2,1}^2 + \lambda_1 \|w\|_{2,1} + \lambda_2 \|\bar{D}w\|_{2,1} \right\}. \tag{3}$$

We call this model Group Fused Lasso (GFL). Notice that if $V = 1$ we recover FL, and if $V = M$, i.e., there is a single group with M variables, GFL boils down to a variant of FL using a TV_2 regularizer, also known as ℓ_2-Variable Fusion [3].

The difference between the models described above is illustrated in Fig. 1, with an example of the structures enforced by LA (some weights identically zero), GL (some groups of weights identically zero), FL (piece-wise constant weights) and the proposed GFL (piece-wise constant groups of weights).

We will solve the GFL optimization problem through convex proximal optimization techniques; we will essentially apply a variant of the FISTA algorithm which, in turn, requires that we can compute the proximal operator of the GFL regularizer, something we will do in next section. We point out that GFL with only the group

$\|\bar{D}w\|_{2,1}$ penalty has been introduced in [7]. However, its solution is different from ours, as it reduces this GFL to a GL model that is then solved by a group LARS algorithm. We believe our approach to be better suited to deal with the full general GFL case.

3 Solving Group Fused Lasso with Proximal Methods

All the ℓ_1 regularizers of Sect. 1 lead to convex but non-differentiable optimization problems, which prevents solving them by standard gradient-based methods. However, they fit very nicely under the paradigm of Proximal Methods (PMs), a set of techniques to optimize convex but possibly non-smooth functions via the splitting of the objective in several "easier" parts. We briefly review this paradigm next, and then we instantiate it for the particular case of GFL.

3.1 Proximal Methods

The proximal methods (see e.g. [9]) are a branch of techniques to minimize convex but possibly non-smooth functions by splitting the objective into easier terms and minimizing them independently via its proximal operator.

In our case, recall that the function to be minimized in (1) is $f_{\mathrm{L}}(w)+f_{\mathrm{R}}(w)$, where we include the penalty factor λ in $f_{\mathrm{R}}(w)$ for the sake of notation. As shown next, the intuitive idea of the PMs is to minimize this sum through the iterative minimization of f_{L} (using a gradient descent step) and f_{R} (using its proximal operator, defined below).

Denote by $\partial h(w)$ the subdifferential at w of a convex function h (i.e., the set of all the subgradients of h at w [10]); since both terms $f_{\mathrm{L}}(w)$ and $f_{\mathrm{R}}(w)$ are convex, w^* will be a minimum of $f_{\mathrm{L}}(w)+f_{\mathrm{R}}(w)$ iff zero belongs to the subdifferential [4]:

$$w^* = \arg\min_{w\in\mathbb{R}^M} \{f_{\mathrm{L}}(w)+f_{\mathrm{R}}(w)\} \quad\Longleftrightarrow\quad 0\in\partial(f_{\mathrm{L}}(w^*)+f_{\mathrm{R}}(w^*)). \tag{4}$$

By the Moreau–Rockafellar [11] theorem, the right-hand side subdifferential can be separated as $\partial f_{\mathrm{L}}(w^*)+\partial f_{\mathrm{R}}(w^*)$, and as $f_{\mathrm{L}}(w)$ is differentiable, the optimality condition can be rewritten in the following equivalent expressions:

$$\begin{aligned} 0\in\partial f_{\mathrm{L}}(w^*)+\partial f_{\mathrm{R}}(w^*) \quad &\Longleftrightarrow\quad 0\in\nabla f_{\mathrm{L}}(w^*)+\partial f_{\mathrm{R}}(w^*)\\ &\Longleftrightarrow\quad -\gamma\nabla f_{\mathrm{L}}(w^*)\in\gamma\lambda\partial f_{\mathrm{R}}(w^*)\\ &\Longleftrightarrow\quad w^*-\gamma\nabla f_{\mathrm{L}}(w^*)\in w^*+\gamma\partial f_{\mathrm{R}}(w^*)\\ &\Longleftrightarrow\quad w^*-\gamma\nabla f_{\mathrm{L}}(w^*)\in(I+\gamma\partial f_{\mathrm{R}})(w^*), \end{aligned}$$

which are satisfied for any $\gamma>0$. Thus, at an optimal w^* the set function $(I+\gamma\partial f_{\mathrm{R}})^{-1}$ verifies

$$w^*\in(I+\gamma\partial f_{\mathrm{R}})^{-1}\left(w^*-\gamma\nabla f_{\mathrm{L}}(w^*)\right). \tag{5}$$

Now, if F is a convex, lower semicontinuous function, its proximal operator at w with step $\gamma > 0$ is defined as

$$z_w = \text{prox}_{\gamma;F}(w) = \arg\min_{z\in\mathbb{R}^M} \left\{ \frac{1}{2}\|z-w\|_2^2 + \gamma F(z) \right\}. \tag{6}$$

Notice that the proximal operator can also be characterized in function of the subdifferential using the optimality condition of (4):

$$z_w = \text{prox}_{\gamma;F}(w) \quad \Longleftrightarrow \quad 0 \in z_w - w + \gamma\partial F(z_w) \quad \Longleftrightarrow \quad z_w \in (I+\partial F)^{-1}(w).$$

Moreover, for a general convex F, it can be shown [4] that ∂F is a monotone operator and, while in principle $(I+\partial F)^{-1}$ would be just a set-function, it is actually uniquely valued. Therefore, it defines a function for which $\text{prox}_{\gamma;F}(w) = z_w = (I+\partial F)^{-1}(w)$ holds, which is an equivalent definition of the proximal operator that justifies the iterative algorithm described next (nevertheless, the definition through problem (6) is still crucial at it provides a general way to compute the proximal operators for non-trivial functions). Thus, going back to (5), it follows that

$$w^* = \text{prox}_{\gamma;f_{\text{R}}}(w^* - \gamma\nabla f_{\text{L}}(w^*)).$$

This fixed-point equation immediately suggests an iterative algorithm of the form

$$w^{k+1} = \text{prox}_{\gamma;f_{\text{R}}}\left(w^k - \gamma\nabla f_{\text{L}}(w^k)\right),$$

which converges because the proximal operator of a convex function turns out to be firmly non-expansive [10]. This is at the heart of the well known proximal gradient method [9] and of its ISTA and FISTA (Fast Iterative Shrinkage–Thresholding Algorithm) extensions [5]. In particular, we will focus on FISTA, based on the pair of equations:

$$\begin{aligned} w^k &= \text{prox}_{\frac{1}{L};f_{\text{R}}}\left(z^k - \frac{1}{L}\nabla f_{\text{L}}(z^k)\right), \\ z^{k+1} &= w^k + \frac{t^k - 1}{t^{k+1}}(w^k - w^{k-1}), \quad \text{where } t^{k+1} = \frac{1}{2}\left(1+\sqrt{1+4t_k^2}\right) \end{aligned}$$

and L is a Lipschitz constant for ∇f_{L}. The main advantage of FISTA is its convergence rate, $O(1/k^2)$, in contrast with the $O(1/k)$ sublinear convergence of ISTA and the proximal gradient method [5].

Notice that all these algorithms require at each step the computation of the proximal operator at the current w^k. We discuss next these operators for GFL.

3.2 Proximal Operators for GFL

We turn now our attention to the particular application of FISTA to the GFL problem.

In order to solve (3) for the complete GFL regularizer, we need the proximal operator of the sum of the GTV and GL terms. Both regularizers are not separable, in the sense that their joint proximal operator cannot be built by the usual expedient of applying consecutively the proximal operators of GTV and GL (this strategy is possible, for example, in the case of the TV and the ℓ_1 norm regularizers). However, we can still solve the proximal problem by the Proximal Dykstra (PD) [9] algorithm, which allows to compute the proximal operator of the sum of several terms combining their individual proximal operators in an iterative fashion (see the scheme in Fig. 3). Therefore we will first focus on computing the proximal operators of GL and GTV separately, which will be later combined through PD.

In our case, the proximal operator of the GL regularizer is just the group soft-thresholding [2] defined as:

$$\operatorname{prox}_{\gamma;\|\cdot\|_{2,1}}(w_{n,v}) = w_{n,v}\left(1 - \frac{\gamma}{\|w_n\|_2}\right)^+,$$

which is a generalization of the soft-thresholding of the ℓ_1 regularizer. Indeed, any group w_n with a norm less than γ is zeroed.

Thus we have to derive now the proximal operator for the GTV regularizer. In particular, we will follow an analogous argument to the one in [3] for TV. We have to solve

$$\operatorname{prox}_{\gamma;\mathrm{GTV}}(w) = \arg\min_{z\in\mathbb{R}^M}\left\{\frac{1}{2}\|z-w\|_2^2 + \gamma\|\bar{D}z\|_{2,1}\right\}, \tag{7}$$

which is a particular case of the more general problem

$$\inf_{z\in\mathbb{R}^M}\ \{f(z) + \gamma r(Bz)\},$$

where $B \equiv \bar{D}$, $r(\cdot) \equiv \|\cdot\|_{2,1}$ and $f(y) \equiv \frac{1}{2}\|y-w\|_2^2$. In turn, this problem can be straightforwardly rewritten as

$$\inf_{z,v}\ \{f(z) + \gamma r(v)\} \quad \text{s.t. } v = Bz,$$

with $z \in \mathbb{R}^M$ and $v \in \mathbb{R}^{(N-1)V}$. Its Lagrangian is given by

$$\mathscr{L}(z,v;u) = f(z) + \gamma r(v) + u\cdot(Bz - v),$$

with $u \in \mathbb{R}^{(N-1)V}$. Therefore [11], we can transform the equivalent saddle point problem $\inf_{z,v}\ \{\sup_u \mathscr{L}(z,v,u)\}$ into the dual problem

$$\inf_u\left\{f^*(-B^\top u) + \gamma r^*\left(\frac{1}{\gamma}u\right)\right\}, \tag{8}$$

by means of the Fenchel Conjugate (FC [4]), which is in general defined for a function F as

$$F^*(\hat{x}) = -\inf \, \{F(x) - x \cdot \hat{x}\}.$$

Thus, going back to problem (7), we have to compute the FCs of f and r for the particular case of the proximal operator of GTV. For $f(z) = \frac{1}{2}\|z - w\|_2^2$ the problem for computing the FC of f becomes:

$$f^*(s) = -\inf_{z \in \mathbb{R}^M} \left\{ \frac{1}{2}\|z - w\|_2^2 - z \cdot s \right\}.$$

The optimum of this problem is clearly $z^* = w + s$, and substituting back we get

$$f^*(s) = -\left(\frac{1}{2}\|s\|_2^2 - (w + s) \cdot s \right) = \frac{1}{2} s \cdot s + s \cdot w.$$

The conjugate of $r(z)$, which in our case is the $\ell_{2,1}$ norm, can be derived using the definition of the FC and the conjugate of the ℓ_2 norm:

$$\begin{aligned} r^*(s) &= -\inf_{z \in \mathbb{R}^{(N-1)V}} \{\|z\|_{2,1} - z \cdot s\} = -\inf_{z \in \mathbb{R}^{(N-1)V}} \left\{ \sum_{n=1}^{N-1} \|z_n\|_2 - z_n \cdot s_n \right\} \\ &= \sum_{n=1}^{N-1} -\inf_{z_n \in \mathbb{R}^V} \{\|z_n\|_2 - z_n \cdot s_n\} = \sum_{n=1}^{N-1} \|s_n\|_2^* \\ &= \sum_{n=1}^{N-1} \iota_{\|s_n\|_2 \le 1} = \iota_{\bigwedge_{n=1}^{N-1} \|s_n\|_2 \le 1}, \end{aligned}$$

where when going from the second to the third row we have used that the FC of the ℓ_2 norm is the indicator function of the unitary ball (which takes the value of 0 inside the ball, and the value of ∞ otherwise), and therefore $r^*(s)$ is equal to ∞ if any group has a norm greater than 1, and equal to 0 if not.

We can now obtain the dual problem of (7) by substituting the FCs f^* and r^* computed above into (8):

$$\begin{aligned} &\min_{u \in \mathbb{R}^{(N-1)V}} \left\{ \frac{1}{2}\|\bar{D}^\top u\|_2^2 - u^\top \bar{D} w + \iota_{\bigwedge_{n=1}^{N-1} \|u_n\|_2 \le \gamma} \right\} \\ &\equiv \min_{u \in \mathbb{R}^{(N-1)V}} \left\{ \frac{1}{2}\|\bar{D}^\top u - w\|_2^2 \right\} \quad \text{s.t. } \|u_n\|_2 \le \gamma, \quad \text{for } n = 1, \dots, N-1, \qquad (9) \end{aligned}$$

where we have completed squares and changed the indicator function to a set of constraints. Since problem (9) is quadratic with simple convex constraints, it can be easily solved using Projected Gradient (PG), which basically consists on iterating a gradient descent step followed by a projection onto the feasible region.

After that, z_w (i.e., the result of the proximal operator) can be recovered from the dual solution u^* through the equality

$$z_w = w - \bar{D}^\top u^*,$$

which follows from the condition $0 = \nabla_z \mathscr{L} = z_w - w + B^\top u^*$.

To finish this section, we observe that the form of the GTV regularizer implicitly assumes a 1-dimensional spatial structure for the data, as we only consider differences between a group of features w_n and the two adjacent groups w_{n-1} and w_{n+1}. However, many problems of interest, such as image processing, present a natural multidimensional structure that cannot be captured by the $\ell_{2,1}$ penalty. Working only with the GTV penalty, and as in [3], a solution for this is to combine several 1-dimensional GTV penalties to obtain a multidimensional GTV. For example, for problems with a 2-dimensional structure, we penalize changes in both row and column-wise adjacent features. More precisely, denoting the i-th row by $w^{[i,\cdot]}$ and the j-th column by $w^{[\cdot,j]}$, we can define the 2-dimensional GTV regularizer as

$$\mathrm{GTV}^{2d}(w) = \sum_i \mathrm{GTV}(w^{[i,\cdot]}) + \sum_j \mathrm{GTV}(w^{[\cdot,j]}).$$

This can be easily extended to more than two dimensions but, again, notice that this multidimensional GTV regularizer is the sum of 1-dimensional GTVs.

With respect to how to compute the proximal operator of multidimensional GTV regularizers, notice that those terms corresponding to the same dimension (for instance, the terms $\mathrm{GTV}(w^{[i,\cdot]})$ corresponding to the different columns) apply over different variables, and are therefore trivially separable, so the proximal operator of the summation of a particular dimension can be computed just by composing the individual proximal operators. Nevertheless, each complete summation applies over all the variables, and they cannot be separated (in the 2-dimensional case, both $\sum_i \mathrm{GTV}(w^{[i,\cdot]})$ and $\sum_j \mathrm{GTV}(w^{[\cdot,j]})$ cover all the variables). In order to combine the proximal operators of these summations we can use once again the PD algorithm. This is illustrated in Fig. 2.

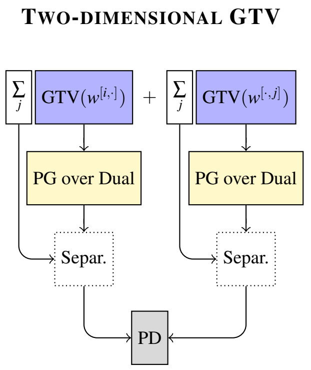


Fig. 2. Scheme of the computations of the proximal operator of the 2-dimensional GTV. The 1-dimensional proximal operators are solved using PG, and those corresponding to different rows (or columns) are just composed (the *Separ.* boxes have no cost). Finally, the proximal operator of both summations are combined with PD.

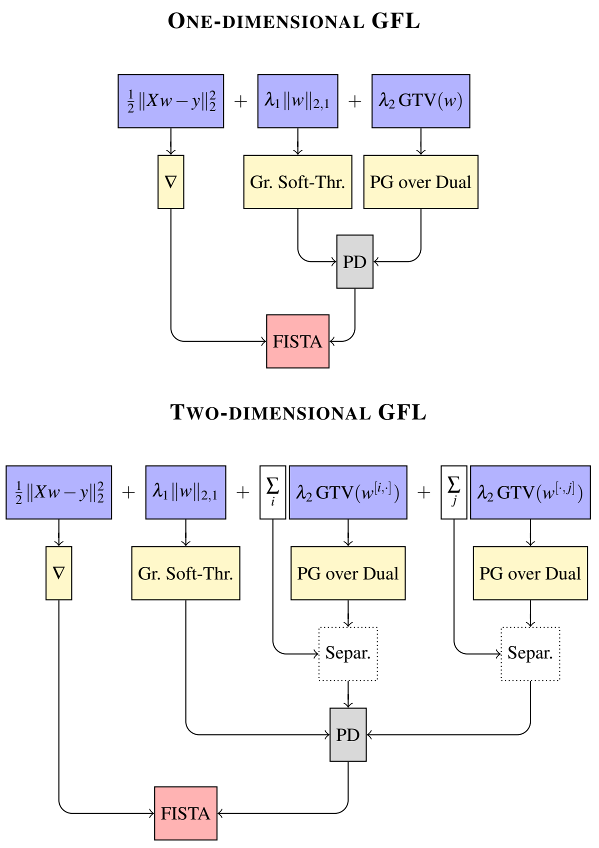

ONE-DIMENSIONAL GFL
$\frac{1}{2}\|Xw-y\|_2^2$ + $\lambda_1\|w\|_{2,1}$ + $\lambda_2\,\mathrm{GTV}(w)$
∇
Gr. Soft-Thr.
PG over Dual
PD
FISTA
TWO-DIMENSIONAL GFL
$\frac{1}{2}\|Xw-y\|_2^2$ + $\lambda_1\|w\|_{2,1}$ + $\sum_i$ $\lambda_2\,\mathrm{GTV}(w^{[i,\cdot]})$ + $\sum_j$ $\lambda_2\,\mathrm{GTV}(w^{[\cdot,j]})$
∇
Gr. Soft-Thr.
PG over Dual
PG over Dual
Separ.
Separ.
PD
FISTA

Fig. 3. Scheme of the computations of the one and 2-dimensional GFL model. First, the proximal operator of the individual 1-dimensional GTV terms (using PG) and of the $\ell_{2,1}$ norm (which is the group soft-thresholding) are computed. All these proximal operators are then merged applying PD. Finally, the proximal operator of f_{R} and the gradient of f_{L} are combined with FISTA.

Similarly, for the case of a complete multidimensional GFL linear model, we should use PD to combine the proximal operators of each 1-dimensional GTV and that of the GL term, as shown in Fig. 3.

4 Experiments

In this section we will illustrate through two synthetic experiments the differences between the structured linear models defined above; in particular, we shall see how GFL is able to detect structure both on the weights of an underlying linear model and also on the features of a structured problem, providing in some particular contexts a much simpler and meaningful model with a performance (in terms of test error) similar to that of RLS. Also, we will present an application of the GTV regularizer to RGB image denoising tasks.

4.1 Synthetic Example: Structured Weights

Data structure may have two main sources. A first one would be in the patterns themselves, in which there can exist a relationship between the features based on some similarity among their sources. We illustrate such a behaviour in Sect. 4.2, including how the different structured models can handle with it. A second possibility, which we consider here, is structure in the generative models that is reflected into pattern structure.

Therefore, this example corresponds to a synthetic structured linear problem where the generative model is defined by a structured vector of weights. Concretely, pattern features are divided into 100 3-dimensional groups, that is, $N = 100$ and $V = 3$. Total dimension is, thus, 300, and our goal is to check if the different linear models are able to detect this underlying structure.

The true model weights are generated randomly as four consecutive segments of 25 groups with constant values for the three group coordinates. This defines a weights vector

$$w^{\mathrm{tr}} = (\underbrace{w_1^{\mathrm{tr}}, \dots, w_{25}^{\mathrm{tr}}}_{\text{Block 1}}, \underbrace{w_{26}^{\mathrm{tr}}, \dots, w_{50}^{\mathrm{tr}}}_{\text{Block 2}}, \underbrace{w_{51}^{\mathrm{tr}}, \dots, w_{75}^{\mathrm{tr}}}_{\text{Block 3}}, \underbrace{w_{76}^{\mathrm{tr}}, \dots, w_{100}^{\mathrm{tr}}}_{\text{Block 4}})^{\top}, \tag{10}$$

where the three components of any two groups belonging to the same block i are equal,

$$w_n^{\mathrm{tr}} = w_m^{\mathrm{tr}} = c^i = \begin{pmatrix} c_1^i \\ c_2^i \\ c_3^i \end{pmatrix} \quad \forall n,m \in \text{Block } i;$$

c^i is either identically zero, or all the three coordinates are different from zero. Therefore, w^{tr} is built in such a way that it makes the features of a group to be simultaneously either active or inactive and in such a way that adjacent features have a block behaviour (the weights are included in Fig. 4a, where each colour represents a different feature within the group).

The true w^{tr} is then perturbed to obtain a weight vector $\tilde{w}$ of the form $\tilde{w}_{n,v} = w_{n,v}^{\mathrm{tr}} + \eta_{n,v}$ with $\eta \sim \mathcal{N}(0; 0.1)$ white Gaussian noise with a standard deviation of 0.1. Random independent patterns x^p are then generated by a $\mathcal{N}(0; 1)$ distribution, and the values $y^p = \tilde{w} \cdot x^p + \hat{\eta}^p$ with $\hat{\eta} \sim \mathcal{N}(0; 0.1)$ then define the targets to be fit by the regression models; we remark that these targets are generated using the perturbed vector $\tilde{w}$. The underlying spatial structure of the weights of (10) is reflected in the y^p values. Moreover, if the number of generated training patterns P is well below the number of features (300), the problem will become ill-posed.

We tackle this regression problem considering 600, 300, 100 and 50 training patterns and using five linear models: Regularized Least Squares (RLS), Lasso (LA), Group Lasso (GL), Fused Lasso (FL) and the proposed Group Fused Lasso (GFL). In the latter two cases, the linear models applied are the complete 1-dimensional FL and GFL, that is, including both regularization terms (the $\ell_1/\ell_{2,1}$ one, and the TV/GTV one). All the models are solved using FISTA with the proper proximal operator, except RLS, which has the closed-form solution (2).

The corresponding regularization parameters are chosen over an initial dataset so that the estimated weights are closest to the true (structured) weights in the ℓ_1 distance. Once the optimal parameters are fixed, 100 different experiment are generated (varying the random input patterns and the perturbed weights, and consequently the targets) in order to average the results.

Table 1 presents the results in terms of the distance between the recovered weights w and the true weights w^{tr}, both with the ℓ_1 distance, $\|w - w^{\mathrm{tr}}\|_1$ and the ℓ_2 one, $\|w - w^{\mathrm{tr}}\|_2$. The superscripts denote the ranking of the models; the same rank is repeated if the differences are not significant under a Wilcoxon signed rank test for zero median, with a significance level of 5%. As it can be seen, GFL achieves the lowest distances in all the cases for both measures. Only FL is comparable, whereas RLS, LA and GL values are clearly worse for the 600 and 300 pattern problems and markedly fail when used with few training samples. The advantage of GFL increases as the problem becomes more ill-posed. As reference values, the distances of the perturbed weights to the true ones are $\|\tilde{w} - w^{\mathrm{tr}}\|_1 = 23.93 \pm 1.04$ and $\|\tilde{w} - w^{\mathrm{tr}}\|_2 = 1.73 \pm 0.07$. When the number of patterns is large enough, RLS obtains comparable results as it has enough information to recover the perturbed weights. Meanwhile, FL and GFL improve this base distance because they do not consider only the error, but also the smoothness of the structure, attaining weights that are even closer to the true weights than the perturbed ones. When the number

Table 1. Distance between the original structured weights and the recovered ones, training with 600, 300, 100 and 50 patterns. The superscript denotes a model's ranking.

Model	$P = 600$	$P = 300$	$P = 100$	$P = 50$
		ℓ_1 **Distance**		
RLS	$23.91^{(4)} \pm 1.04$	$137.12^{(5)} \pm 60.13$	$1337.56^{(5)} \pm 22.40$	$1387.48^{(5)} \pm 17.89$
LA	$23.72^{(3)} \pm 1.03$	$43.58^{(3)} \pm 9.47$	$1155.96^{(4)} \pm 92.45$	$1304.73^{(3)} \pm 53.44$
GL	$26.77^{(5)} \pm 1.55$	$55.28^{(4)} \pm 12.36$	$1121.88^{(3)} \pm 94.29$	$1319.27^{(4)} \pm 56.74$
FL	$9.42^{(2)} \pm 1.45$	$11.16^{(2)} \pm 1.76$	$18.03^{(2)} \pm 3.72$	$76.67^{(2)} \pm 34.72$
GFL	$8.32^{(1)} \pm 1.36$	$10.10^{(1)} \pm 1.60$	$16.68^{(1)} \pm 3.36$	$27.20^{(1)} \pm 7.58$
		ℓ_2 **Distance**		
RLS	$1.73^{(4)} \pm 0.07$	$9.89^{(5)} \pm 4.32$	$111.97^{(4)} \pm 3.09$	$124.45^{(3)} \pm 2.15$
LA	$1.72^{(3)} \pm 0.07$	$3.22^{(3)} \pm 0.70$	$111.96^{(4)} \pm 9.29$	$129.73^{(5)} \pm 4.97$
GL	$2.06^{(5)} \pm 0.11$	$4.15^{(4)} \pm 0.90$	$103.96^{(3)} \pm 9.03$	$128.34^{(4)} \pm 4.97$
FL	$0.74^{(2)} \pm 0.10$	$0.88^{(2)} \pm 0.13$	$1.46^{(2)} \pm 0.25$	$6.20^{(2)} \pm 2.77$
GFL	$0.65^{(1)} \pm 0.10$	$0.77^{(1)} \pm 0.12$	$1.31^{(1)} \pm 0.23$	$2.11^{(1)} \pm 0.58$

of patterns decrease, the FL and GFL values are close to the reference ones, while those of RLS, LA and GL are far away.

Moreover, Fig. 4b shows how for a 600 pattern sample FL and GFL recover quite well the inherent structure of the problem, obtaining constant weights, while RLS, LA and GL tend to the perturbed weights. As the number of patterns gets down to 50 (Fig. 4c), RLS, LA and GL lose the structured reference, whereas FL and GFL always produce structured weights.

4.2 Synthetic Example: Structured Features

In the previous example, the structure of the problem is imposed through the structured lineal model given by the weights of (10). Therefore, the underlying generator of the target is, indeed, structured, and thus the GFL model can capture this nature. In the next synthetic experiment, the problem structure is given by the relationship between the features. This is a more plausible framework in real world settings, where some of the features may be highly correlated due, for instance, to some proximity on their location. Therefore, the structure is imposed through the inputs of the problem, instead of through the underlying weights (indeed, as explained below, the true weights are randomly generated, and consequently unstructured).

Here each synthetic input pattern x^p has the following block structure:

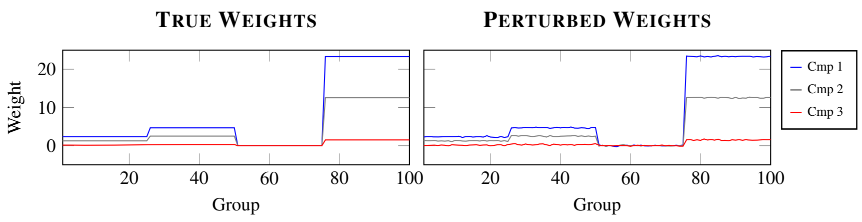
$$x^p = (\underbrace{x_1^p, \dots, x_{25}^p}_{\text{Block 1}}, \underbrace{x_{26}^p, \dots, x_{50}^p}_{\text{Block 2}}, \underbrace{x_{51}^p, \dots, x_{75}^p}_{\text{Block 3}}, \underbrace{x_{76}^p, \dots, x_{100}^p}_{\text{Block 4}})^\top, \tag{11}$$

with 3-dimensional blocks where all the multidimensional features of the same block are equal on their three coordinates. A generative vector of weights w^{tr} is randomly generated following a $\mathcal{N}(0,1)$ and it is kept fixed for the whole experiment. The random independent patterns x^p are generated using a $\mathcal{N}(0,1)$ distribution with the structure of (11) (indeed, only 4 features are generated, one per block, and they are repeated 25 times to compose the 4 blocks). Then they are perturbed with white noise $\mathcal{N}(0,0.01)$; thus the patterns are not perfectly piece-wise constant. Finally, the target values are computed as $y^p = w^{\text{tr}} \cdot x^p + \tilde{\eta}^p$ with $\tilde{\eta} \sim \mathcal{N}(0,0.25)$.

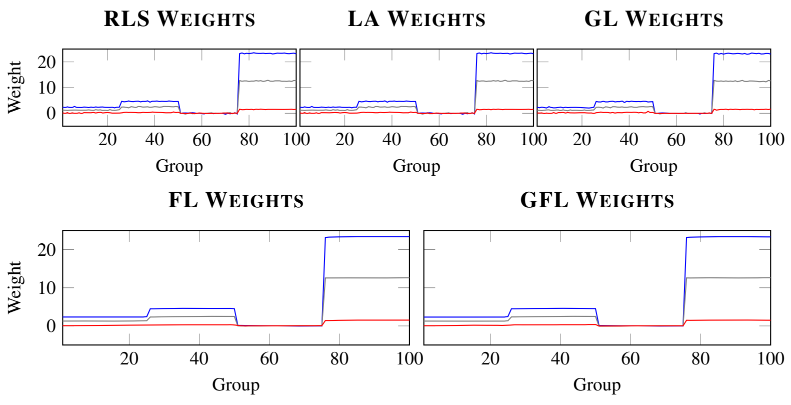
Since for this particular structured problem, all three coordinates in a block are essentially the same, a vector of weights $\bar{w}$ leading to the same targets can be defined averaging the true weights of each block. These weights, which we call structured, could provide a more robust prediction, as the resultant linear model would not rely in any particular representative feature of each group. Formally, $\bar{w}$ can be computed substituting all the true weights of a block by the average of the weights of that particular block:

$$\bar{w}_n = \frac{1}{25} \sum_{m \in \text{Block } i} w_m^{\text{tr}} \quad , \quad \forall n \in \text{Block } i.$$

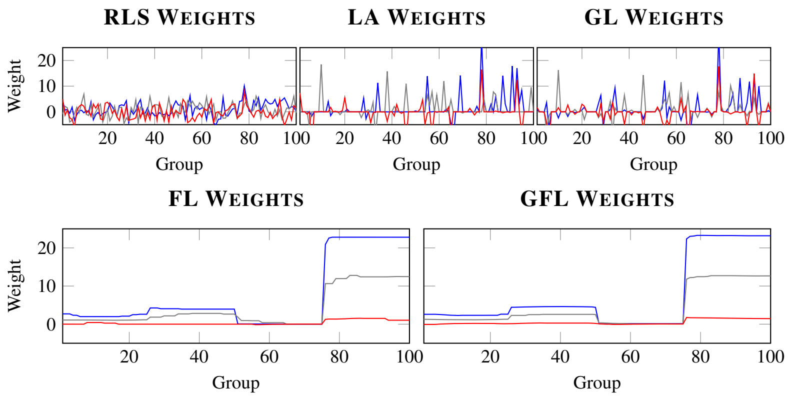
If there were no noise then these structured weights will lead to the same target as the original ones given the constancy of the features, and thus $\bar{w}$ minimizes the error while also providing valuable information about the structure of the problem.



(a) True and perturbed weights.



(b) Results with 600 patterns.



(c) Results with 50 patterns.

Fig. 4. Weights for the Structured Weights Problem

Once the problem is defined, the same five linear models of Sect. 4.1 are compared, again using 600, 300, 100 and 50 training patterns. The optimal regularization parameters are chosen over an initial dataset to minimize the Mean Absolute Error (MAE), as in this case there is not a unique reference vector (both w^{tr} and $\bar{w}$ are equivalent). A test set of 1000 patterns is also generated and kept fixed; the experiment is repeated 100 times changing the random input patterns.

An example of the weights obtained by each model is presented in Fig. 5. The GFL and FL models capture the underlying structure of the problem (converging to the structured weights $\bar{w}$), whereas LA and GL attain a noisy version of them. On the contrary, RLS fails to recover any structure, and it produces much more complex weights. Nevertheless, the RLS weights are nearer to the true ones w^{tr} and hence they have less test error, but this model ignores any relationship between the features. This is discussed in more detail below.

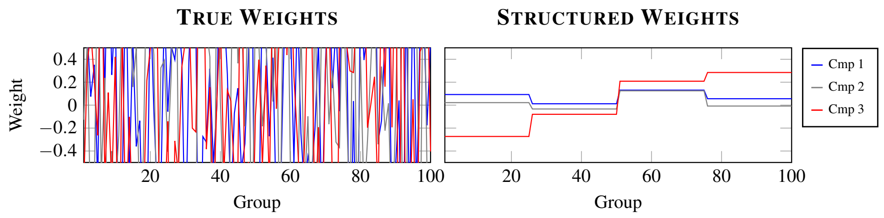
Table 2 quantifies the distance between the weights of each model and the structured weights $\bar{w}$. Using this measure[1], it is clear that GFL recovers the structure with a higher precision than the rest of the models; RLS performs worst when the number of training patterns is large, since this model does not consider any structure and when there is sufficient information it tends to recover w^{tr}. Nevertheless, with the small training size (50 patterns) it also gets structured weights as a collateral effect: if the problem is undetermined with redundant features (as in this case without taking into account the noise), for every linear model there exists an equivalent model that assigns the same weight to all the identical features, and thus it has the same quadratic error (the features are indistinguishable) but less ℓ_2 norm. This is why, with redundancy on the features, RLS also tends to assign the same weight to all of the identical features.

The results over the test set are shown in Table 3. As reference, a prediction using the generative true weights w^{tr} obtains a MAE of 1.95, and a Mean Squared Error (MSE) of 0.61, whereas using $\bar{w}$ the errors are 2.32 and 0.84 respectively (slightly larger because of the noise in the patterns). It is worth noting that, although the true and the structured weights are very different, as shown in Fig. 5a, they produce similar outputs since the underlying problem is ill-posed (the features of the same block are almost the same). The best error rates, for both the MAE and MSE, are attained by RLS when the number of patterns is large enough, because RLS tends to the unstructured but true weights w^{tr}, whereas the other models build a more structured solution, though with a slightly higher error. When the number of patterns gets smaller, the differences vanishes because RLS also obtains a structured solution, since there is not information to get a complex one. Nevertheless, the differences are small, and all the models perform quite similarly, but again notice that the FL and GFL models are the simpler and more meaningful ones, closer to the structured weights $\bar{w}$ from the beginning.

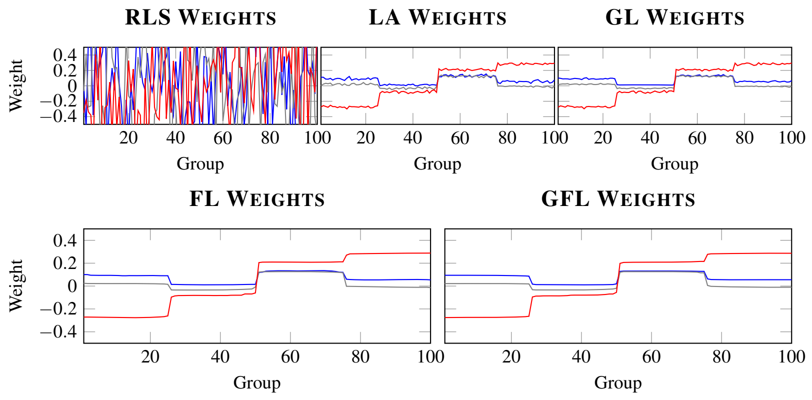
Finally, it is worth commenting that the piece-wise constant weights of GFL and FL suggest the use of a hierarchical model of just 12 features, namely the four 3-dimensional features $x^p_{b_i}$ that result from averaging each one of the blocks,

$$x^p_{b_i} = \frac{1}{25} \sum_{n \in \text{Block } i} x^p_n .$$

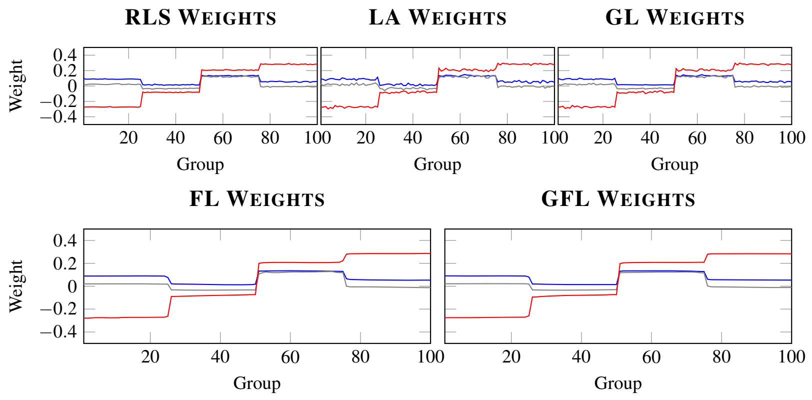
[1] The different scale with respect to Table 1 is because the entries of $\bar{w}$ are averages of the original weights, which are randomly generated with zero mean; therefore the entries of $\bar{w}$ are near zero.



(a) True and structured weights.



(b) Results with 600 patterns.



(c) Results with 50 patterns.

Fig. 5. Weights for the structured features problem

Table 2. Distance between the underlying structured weights and the recovered ones, training with 600, 300, 100 and 50 patterns. The superscript denotes the ranking, and the results are scaled by 10.

Model	$P = 600$	$P = 300$	$P = 100$	$P = 50$
		ℓ_1 **Distance** $(\cdot 10)$		
RLS	$1370.04^{(5)}\pm 53.61$	$1197.71^{(5)}\pm 61.02$	$589.82^{(5)}\pm 47.16$	$10.17^{(3)}\pm 1.25$
LA	$22.73^{(4)}\pm 1.04$	$22.83^{(3)}\pm 0.98$	$22.62^{(3)}\pm 1.06$	$24.40^{(5)}\pm 16.49$
GL	$15.26^{(3)}\pm 1.70$	$23.48^{(4)}\pm 1.06$	$23.74^{(4)}\pm 1.11$	$24.19^{(4)}\pm 25.09$
FL	$6.86^{(2)}\pm 0.82$	$7.01^{(2)}\pm 0.77$	$7.89^{(2)}\pm 0.90$	$8.28^{(2)}\pm 1.19$
GFL	$5.36^{(1)}\pm 0.75$	$2.11^{(1)}\pm 0.56$	$6.62^{(1)}\pm 0.87$	$7.11^{(1)}\pm 1.28$
		ℓ_2 **Distance** $(\cdot 10)$		
RLS	$99.21^{(5)}\pm 3.58$	$86.50^{(5)}\pm 4.29$	$42.68^{(5)}\pm 3.33$	$0.73^{(3)}\pm 0.09$
LA	$1.64^{(4)}\pm 0.08$	$1.65^{(3)}\pm 0.07$	$1.63^{(3)}\pm 0.07$	$1.79^{(4)}\pm 1.34$
GL	$1.18^{(3)}\pm 0.11$	$1.70^{(4)}\pm 0.08$	$1.71^{(4)}\pm 0.08$	$1.90^{(5)}\pm 2.26$
FL	$0.58^{(2)}\pm 0.07$	$0.58^{(2)}\pm 0.06$	$0.63^{(2)}\pm 0.07$	$0.66^{(2)}\pm 0.09$
GFL	$0.49^{(1)}\pm 0.07$	$0.17^{(1)}\pm 0.07$	$0.55^{(1)}\pm 0.06$	$0.56^{(1)}\pm 0.09$

Table 3. Test MAE and MSE for the structured data problem, training with 600, 300, 100 and 50 patterns. The superscript denotes the ranking, and the results are scaled by 10.

Model	$P = 600$	$P = 300$	$P = 100$	$P = 50$
		MAE $(\cdot 10)$		
RLS	$2.19^{(1)}\pm 0.03$	$2.28^{(1)}\pm 0.04$	$2.48^{(1)}\pm 0.09$	$2.82^{(3)}\pm 0.25$
LA	$2.38^{(4)}\pm 0.03$	$2.39^{(3)}\pm 0.03$	$2.57^{(3)}\pm 0.10$	$2.79^{(2)}\pm 0.26$
GL	$2.39^{(5)}\pm 0.03$	$2.38^{(2)}\pm 0.03$	$2.52^{(2)}\pm 0.09$	$2.80^{(2)}\pm 0.24$
FL	$2.35^{(2)}\pm 0.02$	$2.39^{(4)}\pm 0.03$	$2.52^{(2)}\pm 0.09$	$2.75^{(1)}\pm 0.23$
GFL	$2.35^{(3)}\pm 0.02$	$2.38^{(2)}\pm 0.03$	$2.51^{(2)}\pm 0.09$	$2.78^{(2)}\pm 0.24$
		MSE $(\cdot 10)$		
RLS	$0.75^{(1)}\pm 0.02$	$0.81^{(1)}\pm 0.03$	$0.96^{(1)}\pm 0.07$	$1.25^{(3)}\pm 0.24$
LA	$0.87^{(4)}\pm 0.02$	$0.89^{(3)}\pm 0.02$	$1.03^{(3)}\pm 0.08$	$1.23^{(2)}\pm 0.25$
GL	$0.88^{(5)}\pm 0.02$	$0.88^{(2)}\pm 0.02$	$0.99^{(2)}\pm 0.07$	$1.23^{(2)}\pm 0.22$
FL	$0.86^{(2)}\pm 0.01$	$0.89^{(4)}\pm 0.02$	$0.99^{(2)}\pm 0.07$	$1.19^{(1)}\pm 0.21$
GFL	$0.86^{(3)}\pm 0.01$	$0.88^{(2)}\pm 0.02$	$0.99^{(2)}\pm 0.07$	$1.21^{(2)}\pm 0.22$

In fact, this is exactly what these two models are doing, considering these meta-features instead of the original ones. Although the experiment of Sect. 4.1 also resulted into structured weights, in that case the structure was induced by the underlying weights, while in this experiment it is the features where the structure lies, and therefore a sensible approach is to average all those which are similar.

4.3 Image Denoising

In this section we will show how the GTV regularizer can be applied to denoise colour images. Notice that GTV is a natural choice for this, as images have a natural spatial structure where pixels change smoothly and can be considered nearly constant in nearby regions (except in objects borders).

In fact, TV regularization has been extensively used for this task [6] on gray level images, in the form of the denoising model

$$\min_{I^{\mathrm{rc}}} \left\{ \frac{1}{2} \|I^{\mathrm{rc}} - \tilde{I}\|_2^2 + \gamma \mathrm{TV}^{2d}(I^{\mathrm{rc}}) \right\}, \tag{12}$$

for a noisy image $\tilde{I}$ and some 2-dimensional form of TV, whose block structure permits abrupt changes and thus the preservation of the borders of the images. The parameter γ determines the balance between the similarity to the observed noisy image $\tilde{I}$ and the smoothness of the recovered image. By definition, the solution of (12) is just the proximal operator of the TV regularizer with step γ.

When dealing with colour images a possible option is to apply TV denoising independently to each of the three RGB layers, resulting in the problem:

$$\min_{I^{\mathrm{rc}}} \left\{ \frac{1}{2} \|I^{\mathrm{rc}} - \tilde{I}\|_2^2 + \gamma \sum_{n_3=1}^{3} \mathrm{TV}^{2d}(\tilde{I}^{[\cdot,\cdot,n_3]}) \right\}, \tag{13}$$

where $(\tilde{I}^{[\cdot,\cdot,1]}, \tilde{I}^{[\cdot,\cdot,2]}, \tilde{I}^{[\cdot,\cdot,3]})$ are the three colour layers of the noisy image $\tilde{I}$ (each layer $\tilde{I}^{[\cdot,\cdot,n_3]}$, obtained varying the first two indices and fixing the third one, is a matrix with the same dimensions as the image). Because the three regularizers apply to different coordinates, problem (13) can be solved applying the proximal operator of the 2-dimensional TV independently to each one of the colour layers, that is, denoising each layer like a gray level image. However, using this approach the relationship between the different layers is ignored, whereas in natural images the change in the three colours tend to occur at the same point.

A group approach is thus a natural strategy, as each pixel can also be considered a multi-valued 3-dimensional feature. Therefore, GTV fits in this problem as it will denoise an image using the entire problem structure. Specifically, the proximal operator of the 2-dimensional GTV can be employed, which is defined and solved as explained in Sect. 3:

$$\min_{I^{\mathrm{rc}}} \left\{ \frac{1}{2} \|I^{\mathrm{rc}} - \tilde{I}\|_2^2 + \gamma \mathrm{GTV}^{2d}(\tilde{I}) \right\}.$$

In the experiments that follow these denoising algorithms are applied to five different colour images with different noise models (which represent different effects, as discussed in [8]):

Peppers This image is perturbed with additive Gaussian noise, $\tilde{I} = I^{\mathrm{tr}} + \eta$, with $\eta \sim \mathcal{N}(0; 0.05)$. This type of noise usually models thermal noise and, also, the limiting behaviour of other noises that arise during acquisition of images and their transmission.

House Perturbed with speckle noise, that is, multiplicative uniform noise, with $\tilde{I} = I^{\text{tr}} + \eta I^{\text{tr}}$, where η is uniform with 0 mean and variance 0.25, $\eta \sim \mathscr{U}(;0.25)$. The speckle noise arises in coherent light imaging (like radar images) mainly due to random fluctuations in the return signal.

Lena Perturbed with Poisson noise, where each perturbed pixel is generated from a Poisson distribution with mean the original pixel, $\tilde{I} \sim \mathscr{P}(I^{\text{tr}})$. This noise represents variations in the number of photons sensed at a given exposure level.

Mandrill Perturbed with salt and pepper noise, which for grey images sets at random some of the pixel to either black or white. In this case, it is adapted so approximately the 10% of the pixels of each colour layer is set to the minimum or the maximum value. This type of noise is related with errors in the analogue to digital conversion and bit errors.

Squares Perturbed with both additive Gaussian and Poisson noises with the same distributions as before.

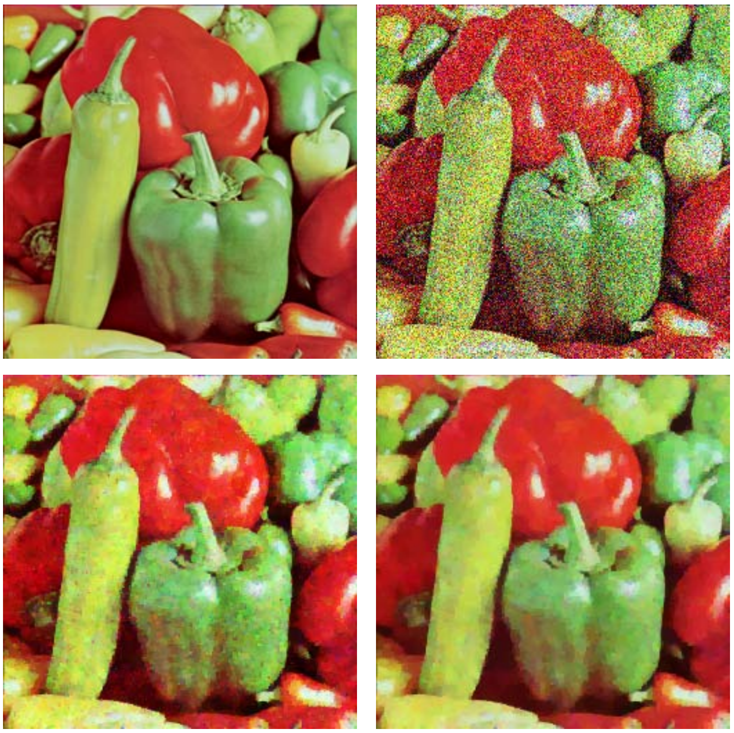


Fig. 6. Example of the image denoising results for *Peppers* (from left to right and top to bottom, the original image, the noisy version and the recovery images using TV and GTV)

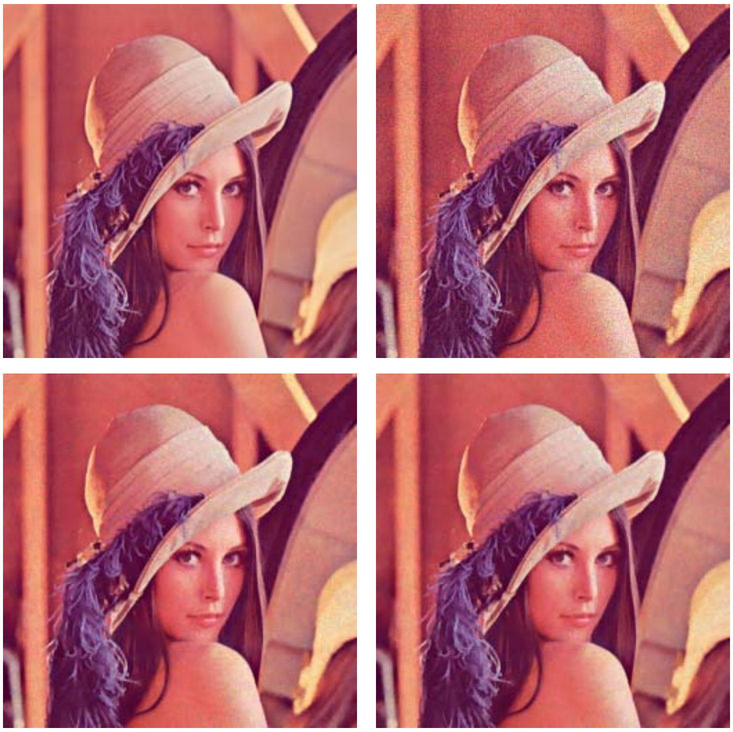


Fig. 7. Example of the image denoising results for *Lena* (from left to right and top to bottom, the original image, the noisy version and the recovery images using TV and GTV)

Table 4. SNR and ISNR of the recovered images for the five proposed examples with the respective noises, using TV and GTV as recovering algorithms. The superscript denotes the ranking considering significant differences between the means.

Model	**Orig** (SNR)	**TV** (SNR)	**GTV** (SNR)	**TV** (ISNR)	**GTV** (ISNR)
Peppers	8.11±0.01	15.80±0.37	18.13±0.37	$7.69^{(2)}$±0.37	$10.02^{(1)}$±0.37
House	7.94±0.01	16.74±0.07	17.20±0.07	$8.80^{(2)}$±0.06	$9.25^{(1)}$±0.07
Lena	21.99±0.01	25.48±0.11	26.63±0.25	$3.49^{(2)}$±0.11	$4.65^{(1)}$±0.25
Mandrill	9.94±0.04	15.51±0.19	16.81±0.12	$5.57^{(2)}$±0.21	$6.87^{(1)}$±0.13
Squares	8.35±0.01	22.87±0.15	23.93±0.16	$14.52^{(2)}$±0.15	$15.58^{(1)}$±0.16

The goal here is to compare the potential advantages of the 2-dimensional signal recovery using GTV over that of TV. Therefore, for each image the optimal TV and GTV penalties are selected as the ones that give the best Improvement in Signal to Noise Ratio (ISNR [1]) over a single perturbed sample, where the ISNR of a recovered image I^{rc}, from a noisy perturbation $\tilde{I}$ of an original image I^{tr}, is defined as:

$$\text{ISNR}(I^{\text{rc}};\tilde{I};I^{\text{tr}}) = 10\log_{10}\frac{\|I^{\text{rc}} - I^{\text{tr}}\|_2^2}{\|\tilde{I} - I^{\text{tr}}\|_2^2}.$$

This is equivalent to the difference between the signal to noise ratio (SNR) of the recovered image, $\text{SNR}(I^{\text{rc}};I^{\text{tr}})$ and the SNR of the original noisy image, $\text{SNR}(\tilde{I};I^{\text{tr}})$, where

$$\text{SNR}(I;I^{\text{tr}}) = 10\log_{10}\frac{\|I^{\text{tr}}\|_2^2}{\|I - I^{\text{tr}}\|_2^2}.$$

Once the optimal parameters are obtained, they are used to test TV and GTV denoising over 25 other different perturbations that follow the same distributions.

As an illustration, Figs. 6 and 7 include, for two of the five experiments, the original image, an example of the noisy image and the image recovered using both TV and GTV.

Numerically, in all cases GTV performed better than TV, as shown in Table 4, where the superscripts denote that all the means are significantly different (using again a Wilcoxon signed rank test for zero median, with a significance level of 5%).

5 Conclusions

In this work we have proposed the Group Total Variation (GTV) regularizer, combining the multidimensional group-sparse features of the Group Lasso regularizer with the block spatial structure of the Total Variation penalty used by Fused Lasso. This regularizer enforces a multidimensional feature to be piece-wise constant at group level, being adjacent groups equal in all the different variables at the same time. In addition, we have shown how to deal with this regularizer computing its proximal operator, which is the basic tool to apply it, both independently and with other regularizers. The GTV regularizer thus appears as a useful tool to reconstruct multidimensional patterns with a spatial structure that reflects smooth changes along the group features. Colour image denoising fits nicely in this framework and we have shown that, for a variety of noise models, GTV performs better than the simpler TV approach of applying 1-dimensional Total Variation independently on each colour. Moreover, and although not done in this work, image deconvolution can also be tackled using the proposed GTV regularizer. In particular, there are methods such as SALSA [1] and TwIST [6] which can generalize a denoising method to address also deconvolution problems.

Furthermore, this GTV regularizer can be merged with a Group Lasso (GL) term, leading to the linear model that we call Group Fused Lasso (GFL). The associated optimization problem for training this model can be solved using FISTA and the proximal operator of GTV and GL. We have illustrated over a synthetic example how GFL effectively captures block structure when present, and makes use of it to address linear ill-posed problems with a number of features much larger than the sample size.

This kind of spatial structure can be found in other real world problems, particularly those for which the underlying data features are associated to geographical locations. Any sensible linear regression models for such problems should assign similar weight values to spatially close features, which is exactly the behaviour that GFL enforces. As further work we intend to study the advantages of GFL in such a kind of problems, which will require the use of the complete 2-dimensional GFL model as explained at the end of Sect. 3, and also to analyse the numerical complexity of the proposed models and possible ways to improve it.

Acknowledgements. With partial support from Spain's grant TIN2010-21575-C02-01 and the UAM–ADIC Chair for Machine Learning.

References

1. Afonso, M.V., Bioucas-Dias, J.M., Figueiredo, M.A.T.: Fast image recovery using variable splitting and constrained optimization. IEEE Transactions on Image Processing 19(9), 2345–2356 (2010)
2. Bach, F., Jenatton, R., Mairal, J., Obozinski, G.: Convex Optimization with Sparsity-Inducing Norms (2011), `http://www.di.ens.fr/~fbach/opt_book.pdf`
3. Barbero, A., Sra, S.: Fast newton–type methods for total variation regularization. In: Proceedings of the 28th International Conference on Machine Learning (ICML 2011), New York, NY, USA, pp. 313–320 (2011)
4. Bauschke, H., Combettes, P.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces. Springer (2011)
5. Beck, A., Teboulle, M.: A fast iterative shrinkage–thresholding algorithm for linear inverse problems. SIAM Journal on Imaging Sciences 2(1), 183–202 (2009)
6. Bioucas-Dias, J.M., Figueiredo, M.A.T.: A new twist: Two-step iterative shrinkage/thresholding algorithms for image restoration. IEEE Transactions on Image Processing 16(12), 2992–3004 (2007)
7. Bleakley, K., Vert, J.P.: The group fused Lasso for multiple change-point detection. ArXiv e-prints (2011)
8. Bovik, A. (ed.): Handbook of Image and Video Processing. Academic Press, Inc., New York (2000)
9. Combettes, P., Pesquet, J.: Proximal splitting methods in signal processing, pp. 185–212. Springer (2011)
10. Combettes, P.L., Wajs, V.R.: Signal recovery by proximal forward-backward splitting. Multiscale Modeling & Simulation 4(4), 1168–1200 (2005)
11. Rockafellar, R.T.: Convex Analysis (Princeton Landmarks in Mathematics and Physics). Princeton University Press (1996)
12. Tibshirani, R.: Regression shrinkage and selection via the Lasso. J. Roy. Statist. Soc. Ser. B 58(1), 267–288 (1996)
13. Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., Knight, K.: Sparsity and smoothness via the fused lasso. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67(1), 91–108 (2005)
14. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society – Series B: Statistical Methodology 68(1), 49–67 (2006)

Incremental Anomaly Identification in Flight Data Analysis by Adapted One-Class SVM Method

Denis Kolev, Mikhail Suvorov, Evgeniy Morozov, Garegin Markarian, and Plamen Angelov

Abstract. In our work we used the capability of one-class support vector machine (SVM) method to develop a novel one-class classification approach. Algorithm is designed and tested, aimed for fault detection in complex technological systems, such as aircraft. The main objective of this project was to create an algorithm responsible for collecting and analyzing the data since the launch of an aircraft engine. Data can be transferred from a variety of sensors that are responsible for the speed, oil temperature and etc. In order to provide high generalization level and sufficient learning data sets an incremental algorithm is considered. The proposed method analyzes both “positive”/”normal” and “negative”/ ”abnormal” examples. However, overall model structure is based on one-class classification paradigm. Modified SVM-base outlier detection method is verified in comparison with several classifiers, including the traditional one-class SVM. This algorithm has been tested on real flight data from the Western European and Russia. The test results are presented in the final part of the article.

Keywords: Flight Data Analysis, Fault Detection and Identification, one class SVM.

1 Introduction

In this paper we present a novel one class classification technique based on support vector machines (SVM) principles [12]. One class classification methods

Denis Kolev · Mikhail Suvorov · Evgeniy Morozov
Moscow State University, Russia
e-mail: {denis.g.kolev,severe013}@gmail.com, morozov_msu@mail.ru

Garegin Markarian · Plamen Angelov
Lancaster University, LA14WA, UK
e-mail: {p.angelov,g.markarian}@lancs.ac.uk

represent a wide set of various machine learning approaches that assume a single class solution of a (fault detection or other) problem containing majority of the samples [15].

One-class classification methods are widely used [13] techniques for fault identification in complex systems [15]. The regular (non-faulty) system performance is presented like a “base” class under the assumption that the data produced by the normally functioning system have a more structured pattern as compared to a rather random pattern of anomalies. Getting the set of all possible system faults is a difficult task using traditional approaches. We needed to use all previous datasets or try to simulate all possible fault detection for best result. Only after that we could use our algorithm. Moreover we needed to obtain all possible combinations of alarms.

Often, conventional data analysis algorithms of the technical systems (for example, fault detection in flight data) are based on the set of predefined logical rules (that use thresholds and conditions). For example, in flight data analysis (FDA) systems such as Express Analysis (EA) used by SKAT [16] or Automated Ground Station (AGS) used by Sagem [17] each possible event/fault is listed and special logical condition is mapped to it. During the analysis process registered data is verified for the appearance of such logical conditions. If any condition is satisfied, the corresponding event is declared.

Such logical conditions use a limited subset of registered physical variables (features) and a large number of thresholds, predefined by the aircraft manufacturer (in the case of a FDA system). Usually predefined rules are verified over huge number of flights and used threshold are usually derived by the manufacturer based on technical details of the system considered. However, standard FDA techniques have several systemic disadvantages. The one of the most important is the fact that, that in the most of the cases provided rules and thresholds are describing general conditions, which may not fit properly for some given machine. Thus, sometimes it is required to adapt the rule-based detection algorithm for every machine/aircraft manually. Also, a limited event list does not allow the detection of any novel event. Due to the nature of rule-based algorithms, some interactions between features’ values are also not taken into account [18], as each rule takes into account only a relatively small subset of registered parameters.

Proposed method is related to the family of data-driven approaches. Usually, such type of techniques do not aim to describe the system using some deterministic approaches like differential equations, finite state automates and etc. In general, data-driven approaches just provide a parametric models which are fitted over some batch of data that describes the system behavior, which is usually called "learning set". Such type of modeling is useful in the cases when considered system is very complex , for instance aircraft. Most of the data-driven models have two main operation stages: "learning" and "application". Usually, these two stages are split in the time and learning stage is performed before application stage.

Described algorithm was developed with special condition that it should be able to utilize newly coming data without need to fit the algorithm over the full amount of learning set. This means, that any new batch of data can be added to the learning set and the parametric model will be updated without re-initialization. The parameters of the model are updated only using the newly coming data, but taking into account all previous learning batch. This includes recursive nature of the mathematical expressions with no need to store/memorize past data but to accumulate instead the statistically aggregated information, and the ability to update recursively the parameters of the algorithm itself [13].

One of the most popular approaches in one-class classification is support vector machine (SVM) algorithm [1]. Initially proposed as a linear classifier, it is possible to introduce non-linearity in the algorithm using the well-known “kernel trick” [12]. The algorithm provides solution of the given *empirical risk function* for which an incremental optimization procedure can also be developed. The standard one-class SVM method [7] is trained over the data set that contains only “base” class examples (“normal”, not faulty one).

One-class classification problem can also be used for solving outlier/anomaly detection problem [5,7]. That is why this type of technique is convenient in the cases, when most of the data from sensors can be associated with a single class. This subset of data is usually described as the “base” class (the class of “normality”). All other data have too complex pattern or are not in large amount. Typically, the one-class classifier [5,7,11] takes as an input a vector of the data samples and produces as an output the label of that data sample/input features vector. The output of the one-class classifier is trivial and represents a Boolean YES/NO (if the sample belongs to the “base” class or not) [7].

In an offline scenario there is a training stage that precedes the classification phase stage [13, chapter 8]. Experts or another system during the training phase provide correct labeled data. In an online scenario the correct labels may be provided one by one or all at once [13, chapter 8], e.g. after each flight of an aircraft if there was no any fault all samples are marked as “normal”. Also part of the training samples can be negative (knowingly “abnormal”). In the presented method we use not only “normal” samples in the recursive one-class SVM, but also the “abnormal”/”negative” ones. This is the main difference in comparison with the traditional systems.

In the next section we show the problem formulation, then, we describe the traditional SVM method as well as the incremental learning procedure and in last sections we describe the developed technique and illustrate the results of the performance of the algorithm. Finally, in the Conclusion, we discuss our future research directions.

2 Problem Specifics

In this paper we present a novel technique for a single class SVM which is particularly suitable to perform fault detection. (FD). Very important task is to find a right data set which contains information about all possible situations in the system. As we mentioned already, it is almost impossible to simulate a variety of all possible alarms and combinations of alarms especially in an plane during flight.

The one-class classification algorithms could be separated into two main groups:

– Support Vector Machine (SVM) -based one-class classification;
– One-class classification by the means of distribution density (membership) restoration.

In this work, the first group of methods will be investigated and tested on real flight data from various aircraft producers.

Incremental learning requirement to the algorithm, that was mentioned earlier is motivated by the nature of the considered problem. It is usually impossible to collect the data, containing all possible operation modes. For this reason, it is not acceptable to use the algorithms, which cannot be adapted to the new data sets after initial training. Ideally, it should be possible to adapt/evolve the classifier (fault detector) itself in terms of its structure and mechanism to fit the newly obtained data pattern especially if the changes are of a structural nature [13].

The standard representation of the data, processed by machine learning algorithms is in the form of multidimensional vectors [14] where the dimensionality is equal to the number of features (usually, physical variables such as air speed, pitch angle, altitude, etc.).

In mathematical terms, the data vector is represented as a set $X = \{x_1, \dots, x_T\}$, where T is the duration of the whole period of observation. A subset of the observed data is supposed to be used as training set although in incremental algorithms this training data set is updated by appending it with the new data vectors [13]. In an extreme case, some classifiers can start ‘from scratch’ with the very first data vector used as a model of the normality [3, 13], but the classification rate (and the precision, respectively) will only reach more meaningful and useful values accumulating a larger number of data vectors, as the parametric model converges. So, in this respect we assume that the initial number of training data set, especially when we describe a FDA system is more substantial.

For each time interval when a fault is detected and confirmed a label of “abnormal”/”faulty” can be added as soon as the confirmation is available.

The following quality measures are usually used to compare different fault detection and identification methods:

- False positives (FP). This is defined as the ratio of the data vectors that are marked as faulty by the method being interrogated, but in reality there is no fault or event.
- False negatives (FN). This is defined as a ratio of the data vectors that are supposed not to be faulty by the method that is being interrogated, but an event actually took place.
- Total error (TE). This is the ratio of data vectors in which the identification result of the method differs from the reality.

In this study a legacy FDA system used in Russia called SKAT implementing a method called Express Analysis (EA) which is based on a very large number of thresholds and expert rules [16] is used as ‘ground truth’. The use of experts is very expensive and this assumption means that in some cases a fault may have taken place but not being noticed by the experts or vice versa. Therefore, the comparison is with EA (used in SKAT and AGS) and not necessarily with the reality.

3 Support Vector Machines

The main idea of the popular two-class SVM method is to solve the classification problem by separating the data vectors in two groups by a hyper-plane [1]. This hyper-plane is selected in such a way to maximize the “gap” between the classes.

Formally speaking, if we consider a learning set $X = \{(x_1, y_1), \dots, (x_m, y_m)\}, \forall i = 1, \dots, m\ x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$, where x_i is the data object description (or just "object") and y_i is corresponding class label. Two-class SVM can be considered as a method for linear classifier fitting. Corresponding classification model is:

$$y(x) = sign\big(f(x, w, \rho)\big) = sign(\langle x, w\rangle - \rho)$$

where parameters to be fitted are $w \in \mathbb{R}^n$ - hyper-plane normal vector, and ρ is the bias. Model form provides simple geometrical explanation for the classification: class labeled as "-1" is situated in the negative semi-space, defined by $f(x, w, \rho)$ and class "1" is in the positive semi-space.

Hyper-plane fitting is performed with the idea to maximize the minimal distance to the closest points from each class in the way, that all the points are classified correctly. It can be shown, that such formulation can be expressed by following optimization problem:

$$\begin{cases} \min\limits_{w,\rho} \dfrac{1}{2} \|w\|^2 \\ y_i(\langle x_i, w\rangle - \rho) \geq 1\,, \quad i = 1 \dots m \end{cases}$$

However, such formulation is not applicable for the cases when the classes are not linearly-separable. In the cases, when it is impossible to distinguish the classes by linear function, constraints of the problem define an empty set.

Linear SVM can be applied to linearly non-separable classes by modification of the initial optimization problem statement. Constraints are changed by introduction of so called "*slack variables*", which allow misclassifications of the vectors in the learning set. Resulting optimization problem is as follows:

$$\begin{cases} \min\limits_{w,\rho} \dfrac{1}{2} \|w\|^2 + C \sum\limits_{i=1}^{m} \xi_i \\ y_i(\langle x_i, w\rangle - \rho) \geq 1 - \xi_i\,, \quad i = 1 \dots m \\ \xi_i \geq 0 \quad i = 1 \dots m \end{cases}$$

From mathematical perspective, introduction of the slack variables defines an unlimited set, so that feasible pair (w, ρ) always exist.

It can be shown, that $w = \sum_{i=1}^{m} \alpha_i x_i$, where α_i are Lagrange multipliers related to the limitation $y_i(\langle x_i, w\rangle - \rho) \geq 1 - \xi_i$. One of the most important properties of SVM is the sparsity of α, because most of the α_i are 0. Non-zero Lagrangian multipliers correspond to vectors for which $y_i(\langle x_i, w\rangle - \rho) \leq 1$. If the classifier is appropriate the amount of non-zero α_i should be relatively small.

The characteristics $M(x_i) = y_i(\langle x_i, w\rangle - \rho)$ measures the "confidence" of the classification of x_i. If $M(x_i) < 0$ then the misclassifies x_i. Usually $M(x_i)$ is referred as "margin".

However, in some cases linear model is too rough for classification rule estimation. In that cases a "kernel trick" is applied. The general idea is that vectors/samples are mapped into a higher dimensionality space and to solve the classification problem there. “Kernel trick” [13], is based on displacement of “linear” dot product by non-linear function, called kernel. The kernel is assumed to satisfy the Mercer’s conditions [5,12].

There are two main approaches to support vector one-class classification. The first one is called Support Vector Descriptor (SVD), which attempts to find a hyper-sphere of minimal radius so that all data points are within given sphere. The model is characterized by two main parameters: $(a, R), a \in \mathbb{R}^n, R \in \mathbb{R}$ - sphere center and radius respectively. Corresponding optimization problem is as follows:

$$\begin{cases} \min\limits_{w,\rho} \frac{1}{2} R^2 + C \sum\limits_{i=1}^{m} \xi_i \\ \|x_i - a\|^2 \leq R^2 + \xi_i\,, \quad i = 1 \dots m \\ \xi_i \geq 0 \quad i = 1 \dots m \end{cases} \tag{1}$$

In case of SVD the resulting hyper-sphere center can be presented as a sparse linear combination of vectors from the learning set.

Second approach proposed by Scholkopf is based on linear separation model. The idea is to "place" the base class data from the origin with maximal margin.The optimization problem of the "hyper-plane" one-class SVM is given as follows:

$$\begin{cases} \frac{1}{2} ||w||^2 + C \sum_{i=1}^{m} \xi_i - \rho \; \rightarrow min_{w\xi\rho} \\ < x_i, w > -\rho \geq -\xi_i\,, \quad i = 1 \dots m \\ \xi_i \;\geq 0 \end{cases} \tag{2}$$

w is the normal vector to the separating plane,

ρ is a free term of the hyper-plane;

In this work we focus on hyper-plane One-class SVM with optimization problem (2).

In most of the cases selection of "penalization" parameter C is performed by cross-validation. However, hyper-plane model has a significant property, that simplifies the selection of C. If we assume that $C = \frac{1}{m\nu}$, then ν is an upper bound of error fraction of the model over learning set.

This property was proven in [10], and is very useful in most of the practical cases as it gives information about the asymptotic FP rate in one-class SVM methods.

The easiest way to solve the main optimization problem is through the dual problem:

$$\begin{cases} W = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\langle x_i x_j\rangle + \rho\,(1 - \sum_{i=1}^{m}\alpha_i) \to min_{w\xi\rho} \\ 0 \le \alpha_i \le \frac{1}{vm} \quad i = 1 \ldots m \\ 1 - \sum_{i=1}^{m}\alpha_i = 0 \end{cases} \tag{3}$$

where α is the Lagrangian multiplier [8].

The optimal hyper-plane parameters that are solutions of (2) or, equivalently, (3) can be obtained as a linear combination of the data samples:

$$w = \sum_{i=1}^{n}\alpha_i x_i \tag{4}$$

All points x_i which are used for algorithm training and for which $f(x_i) \le 0$ are called support vectors (SV). Here $f(x) = sign\,(\langle x, w\rangle - \rho) = \text{sign}\,(\sum_{i=1}^{n}\alpha_i\,\langle x_i, x\rangle - \rho)$. The margin function for one-class classification is defined similarly, assuming that all $y_i = 1$:

$$M(x_i) = \langle x, w\rangle - \rho = \sum_{i=1}^{n}\alpha_i\,\langle x_i, x\rangle - \rho$$

However, proposed linear model may be over-simplified for the considered application domain. In order to introduce more flexible model, a kernel trick is used. As one-class SVM preserve the sparse solution property, the computational model is suitable for large learning datasets.

In this work flight data analysis is performed by applying one-class SVM with a Gaussian kernel that replace the dot product:

$$f(x) = sign\left(\sum_{i=1}^{m}\alpha_i \exp(-\gamma\|x_i - x\|^2) - \rho\right)$$

Hereafter we will present the algorithm for incremental one class SVM learning.

The optimization problem (3) is a quadratic one and it has a numerical solution which is described by *Karush-Kuhn-Tucker* (KKT) conditions [9], which provide both necessary and satisfactory conditions for the optimization problem (2) to have an optimal solution:

$$\frac{\partial W}{\partial \alpha_i} = M(x_i)\begin{cases} > 0, & \alpha_i = 0 \\ = 0, & 0 \le \alpha_i \le \frac{1}{vm} \\ < 0, & \alpha_i = \frac{1}{vm} \end{cases} \tag{5}$$

$$\frac{\partial W}{\partial \rho} = 1 - \sum_{i=1}^{m}\alpha_i = 0$$

One of the outcomes of (5) is formal division of the whole data set into the three separate (non-overlapping) data sub-sets:

1. Subset C (correct): Data vectors which were correctly classified by the algorithm, the corresponding margin function is positive
2. Subset M (marginal): Data vectors which are situated on the border line of the "base" class, the margin function is equal to 0.
3. Subset E (errors) Data vectors that are misclassified by the algorithm and corresponding margin function is negative.

Learning set decomposition is essential for SVM model (the hyper-plane) to accommodate newly obtained data vectors [8] (learn incrementally). Traditionally, SVM model is trained in an offline mode, but analytically optimal solution can also be obtained during online SVM parameters model update [10].

One can see that the model proposed can be characterized by 2 main parameters: $(\alpha, \rho), \alpha \in \mathbb{R}^m, \rho \in \mathbb{R}$ - vector of Lagrange multipliers and free term (for fixed learning set). The idea is to add a newly coming data into the learning set by assigning a weight α_c for new data vector x_c and modification of all weights related to all other vectors. This method updates the SVM parameters with every new data vector, x_c by additive update of the parameters of the model(for more details, please, see [10]):

$$\alpha^{new} = \begin{bmatrix} \alpha_1^{old} \\ \vdots \\ \alpha_n^{old} \\ \alpha_c^{old} \end{bmatrix} + \begin{bmatrix} \Delta\alpha_1 \\ \vdots \\ \Delta\alpha_n \\ \Delta\alpha_c \end{bmatrix} \tag{6}$$

$$\rho^{new} = \rho^{old} + \Delta\rho \tag{7}$$

The overall idea is to update the parameters of the model – the Lagrange multipliers for all the previous data and set new multiplier for the newly obtained data as well as change the free term value. As a result of the changes, the solution will still satisfy the KKT conditions.

When newly obtained learning data sample comes, two options are possible:

If the newly obtained data vector is correctly classified, than no update is needed, as setting corresponding α_C value to be *0* is enough for KKT conditions (5) to be fulfilled.

If the new data vector is misclassified by the algorithm, than it becomes a support vector (SV) and its α_C value is above *0*. Then so called "adiabatic" weight modification takes place. Initially α_C is set to 0, it is increased by a special procedure until KKT conditions (5) are met. Adiabatic procedure includes 2 iterative sub-steps: "increase of alpha" and "learning set reorganization".

The first one aims to increase α_C of x_c in the way that:

1. For all x_i in the subset M (marginal cases) the Margin $M(x_i)$ is situated at 0.
2. For all x_i in the subset C and E the Margin $M(x_i)$ is positive and negative correspondingly.

Along with the increase of α_C, all Lagrange multipliers of data vectors in subset M are changed as well. Free term ρ is changed too. Lagrange multipliers from sets C and E are supposed to be fixed. However, the margins of vectors from E and C change along with the change of α_C (and all Lagrange multipliers from M and ρ). Due to the change of margins of vectors from E and C changes and Lagrange multipliers from M, the increase of alpha is bounded. For instance, one of the vectors from E may get margin equal to 0 (which is the lower bound for the vectors from E), or Lagrange multiplier for the vector from M may hit 0 or $\frac{1}{vm}$ (limitations for each alpha). We will describe all possible situations when the step "increase of alpha" a bit further. Hereafter we will describe the increase procedure in more details [10]:.

Suppose initially we have a sample set $X = \{x_1, \ldots, x_m\}$ with corresponding values of Lagrange multipliers $(\alpha_1, \ldots, \alpha_m)$ and free term ρ, which satisfy KKT. We want to add a vector x_c into learning set with initially assigned value $\alpha_c = 0$. First of all, let's point that such value of α_c is acceptable for the procedure definitions, as it does not change the margin of any new vector. Lets' consider how shall we change the model if we change α_c by $\Delta\alpha_c$

Denote set $M \subseteq X$, $M = \{x_1^M, \ldots, x_{|M|}^M\}$ - vectors from margin set. $\alpha^M = (\alpha_1^M, \ldots, \alpha_{|M|}^M)$ - corresponding Lagrange multipliers. For sets C and E similar definitions are used.

Margin set equality equation can be expressed as follows:

$$K_{ME} * \alpha^E + K_{MC} * \alpha^C + K_{MM} * (\alpha^M + \Delta\alpha^M) + k_{M,c} * (\alpha_c + \Delta\alpha_c) - \rho - \Delta\rho = 0$$

Therefore, taking into account KKT:

$$Q_{MM} * \Delta\alpha^M + k_{M,c} * \Delta\alpha_c - \Delta\rho = 0$$

$$Q_{MM} \begin{bmatrix} -\Delta\rho \\ \Delta\alpha_M \end{bmatrix} = -\begin{bmatrix} 1 \\ k_{M,c} \end{bmatrix} \Delta\alpha_c \text{ where } Q_{MM} = \begin{bmatrix} 0 & 1^T \\ 1 & K_M \end{bmatrix} \qquad (8)$$

$K_{MM} \in \mathbb{R}^{|M|\times|M|}$ is the matrix of inner (dot) products between data vectors from the subset M, $(K_{MM})_{ij} = \langle x_i^M, x_j^M \rangle$. Matrices $K_{ME} \in \mathbb{R}^{|M|\times|E|}$ and $K_{MC} \in \mathbb{R}^{|M|\times|C|}$ denote inner product between sets M ans E and C correspondingly

$k_{M,c}$ - is the vector of inner products between the data vectors from the subset M and the new vector x_c;

$\Delta\alpha_M$ denotes the updates of Lagrange multipliers that correspond to the data vectors from the subset M.

$\Delta\rho$ denotes the update of the free-term of the hyper-plane

$$\Delta\rho = -\beta_\rho(x_c)\Delta\alpha_c, \quad \Delta\alpha_j^M = \beta_M(x_c)\Delta\alpha_c,$$

where we define:

$$\begin{bmatrix} \beta_\rho(x_c) \\ \beta_M(x_c) \end{bmatrix} = -Q_M^{-1} \begin{bmatrix} 1 \\ k_{M,c} \end{bmatrix},$$

Substituting this result in (8) we have following expression for the change of the margin for some given vector x_i:

$$\Delta M(x_i) = \tau(x_c, x_i)\Delta\alpha_c, \quad \forall\, i \in E \cup C \cup \{x_c\},$$

where:

$$\tau_i(x_c, x_i) = \langle x_i, x_c\rangle + \sum_{i=1}^{|M|} \langle x_i^M, x_c\rangle \beta_m(c) + \beta_\rho(c)$$

One can check that $\forall x_i \in X\colon x_i \in M \;\Rightarrow \tau_i(c) = 0$

Therefore, for each value of $\Delta\alpha_c$ we may obtain corresponding $\frac{\Delta\rho}{\Delta\alpha_M}$ so that the margins of the vectors from M-set are still 0. However, the margins of the vectors from E and C sets are not fixed, therefore if we change $\Delta\alpha_c$ too strong, we may get some vectors from C-set with negative margins or positive margins at E-set.

Lets' consider simple example. As it was mentioned, the change of α_c should not come to the situation when any of vectors from E set has a positive margin. Therefore:

$$\forall\, x_i^E \in E \;\; M(x_i^E) + \Delta M(x_i^E) < 0 \;\Rightarrow \Delta M(x_i^E) < -M(x_i^E) \;\Rightarrow$$

$$\tau(x_c, x_i)\Delta\alpha_c \le -M(x_i^E)$$

Taking into account that $\Delta\alpha_c > 0$ and $\forall\, x_i^E \in E\; M(x_i^E) < 0$ derived limitation should be taken into account only for the $x_i^E \in E\, :\, \tau(x_c, x_i) > 0$, i.e. the vectors, which margin is growing. Therefore, the final limitation is as follows:

$$E_+ : \{x_i \in E\; |\tau(x_c, x_i) > 0\}\;\; \Delta\alpha_c < \min_{E_+}\left[-\frac{M(x_i^E)}{\tau(x_c, x_i)}\right]$$

However, equality $\Delta\alpha_c = \min_{E_+}\left[-\frac{M(x_i^E)}{\tau(x_c, x_i)}\right]$ means that at least one vector from E_+ gets margin equal to 0.

Similar upper bound limitations for $\Delta\alpha_c$ could be derived for the limitations on M set and C. Then the minimal upper bound is taken as $\Delta\alpha_c$. However, increasing the α_c leads to structural change of the sets E, C and M, because at least one vector from the set related to the selected upper bound "hits" a bound of KKT conditions, i.e. to low\high margin or Lagrange multiplier. The increase of α_C proceeds until one of the following conditions is satisfied:

1) For data vector x_i in the subset C the function $M(x_i)$ obtains value *0*. Then this data vector/point is moved to the subset M with α set to *0* and α_C is increased with the new structure of the data subsets.
2) For data vector x_i in the subset E the function $M(x_i)$ obtains value *0*. Then this data vector/point is moved to the subset M with α set to *0* and α_C is increased with the new structure of the data subsets.

3) For data vector x_i in the subset M the value of α$_i$ gets value $\alpha_i = \frac{1}{vm}$. Then this data vector/point is moved to the subset E with α set to *0* and α$_C$ is increased with the new structure of the data subsets.
4) For data vector x_i in the subset M the value of α gets value *0*. Then this data vector/point is moved to the subset C with α$_C$ is increased with the new structure of the data subsets.
5) $\alpha_c = \frac{1}{vm}$. Then the new data vector is moved to the subset E and the algorithm terminates.
6) $M(x_c) = 0$. Then the new data vector is moved to data subset M with the current α$_C$ and the algorithm terminates.

If the cases 1)-4) take place, than the structure of the sets E,C,M is changed, therefore the matrix Q_{MM}^{-1} and matrices of inner products should be recalculated. All this operations could be performed in an optimized way, by moving replacing of the rows of the matrices and using Woodberry matrix inverse rule.

4 The Proposed Method

Existing one-class classification techniques mainly use their own settings to reduce the FP rate (false alarms). In addition, we can assume that the low level of FN (misses of real anomalies) is associated with the successful use of the model with a properly chosen classification system. But we have high level of negative reactions in practical application of this approach, because a vectors/points do not belong to the "base" class are classified incorrectly. At present day flight data processing using statistical approaches one of the most significant problems is the high FN rate (misses). Two-class classification methods are also used to determine for fault identification problem, but for the a successful operation of this method we have to use the training set, which in real life is an elusive goal for each aircraft. That is why we propose a model which contains information about all possible fault detections, which allows our algorithm to understand deviations, regardless of whether they were before or not. This also allows us to classify our model as one-class classification-type incremental SVM.

We propose a new technique of weight distribution which is applied for FP and FN errors separately. Also, we suggest the existence of two classes which are labeled by *1* and *-1* where *1* corresponds to the "base" class ((flight) data with no faults), and *-1* – to the "faulty" class.

The proposed method can be formulated as the following optimization problem:

$$\begin{cases} \frac{1}{2}\|w\|^2 + \\ + \frac{1}{v_1 m_1}\sum_{i=1}^{m} \xi_i [y_i = 1] + \frac{1}{v_2 m_2}\sum_{i=1}^{m} \xi_i [y_i = -1] - \rho \to min \\ y_i(< x_i, w > - \rho) \geq -\xi_i, \quad i = 1, \dots, m \\ \xi_i \geq 0, \quad i = 1, \dots, m \end{cases} \tag{9}$$

where m_1 is the number of positive examples in the learning sample, m_2 is the number of negative examples.

The Lagrangian of the problem (9) can be formulated as follows:

$$L(w,\xi,\rho,\lambda,\mu) =$$
$$\frac{1}{2}\|w\|^2 +$$
$$\frac{1}{v_1 m_1}\sum_{i=1}^{m}\xi_i[y_i = 1] + \frac{1}{v_2 m_2}\sum_{i=1}^{m}\xi_i[y_i = -1] - \rho + \sum_{i=1}^{m}(\alpha_i(y_i\rho - y_i < x_i, w > -\xi_i) - \mu_i\xi_i) \tag{10}$$
$$w.r.t.: \quad \alpha_i \ge 0, \mu_i \ge 0 \;\; i = 1 \dots m$$

Applying KKT conditions (in a differential form) we have the following optimality conditions:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{m}\alpha_i y_i x_i = 0 \tag{11}$$
$$\frac{\partial L}{\partial \rho} = 1 - \sum_{i=1}^{m}\alpha_i y_i = 0 \tag{12}$$
$$\frac{\partial L}{\partial \xi_i} = \frac{1}{v_1 m_1}[y_i = 1] + \frac{1}{v_2 m_2}[y_i = -1] - \alpha_i - \mu_i = 0 \tag{13}$$

For the proposed optimization problem the margin function of a vector x by $M(x) = -y_i\rho + y_i\langle x, w\rangle$.

The complementary slackness terms are represented as follows:

$$1) \quad \xi_i = 0 \;\; or \; \mu_i = 0 \tag{14}$$
$$2) \quad \alpha_i = 0 \; or \; y_i\rho - y_i < x_i, w > -\xi_i = 0 \tag{15}$$

So, all the samples could be divided into three subsets in the same way as in the regular SVM (Vapnik, 1998):

1) $\mu_i = \frac{1}{v_1 m_1}[y_i = 1] + \frac{1}{v_2 m_2}[y_i = -1], \alpha_i = 0, \xi_i = 0, M(x) \ge 0$ - correctly classified samples
2) $\mu_i > 0, \alpha_i > 0, \xi_i = 0, \;\; M(x) = -\xi_i$ – samples, that are situated on the border
3) $\mu_i = 0, \;\; \alpha_i = \frac{1}{v_1 m_1}[y_i = 1] + \frac{1}{v_2 m_2}[y_i = -1], \xi_i > 0, M(x) = -\xi_i$ – misclassified objects.

One can see that the listed sets are analogous of C, M and E respectively. Using all results from formulas above we have the corresponding dual optimization problem:

$$-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j < x_i, x_j > - \rho(1 - \sum_{i=1}^{m}\alpha_i y_i) \to max$$
$$w.r.t.$$
$$\begin{cases} 0 < \alpha_i < \dfrac{1}{v_1 m_1}[y_i = 1] + \dfrac{1}{v_2 m_2}[y_i = -1] \\ 1 - \displaystyle\sum_{i=1}^{m}\alpha_i y_i = 0 \end{cases} \tag{16}$$

From our results we can see that changes in the initial dual problem formulation give us a chance to use into account information about “negative” (“faulty”) class with various regularization weights. Based on information above we propose an algorithm for the incremental update of the SVM model to accommodate the newly obtained data vectors.

The idea of the incremental algorithm for the proposed algorithm is similar to the approach described in section III. The update is based on the two steps, described at previous section: "increase of alpha", and "learning set reorganization".

Lets’ consider the optimization problem (16), in order to derive the incremental algorithm. The functional can be represented as:

$$-\frac{1}{2}\alpha^T K\alpha - \rho(1 - < y, \alpha >)$$

If a new vector x_c is added, than the Lagrange multipliers should be updated. The changes in α and ρ are performed in the same way, as described previously: changes are performed only to the Lagrange multipliers related to M-set, preserving the margin sign for all the rest sets and KKT conditions. Thus:

$$K_{ME}\alpha^E + K_{MC}\alpha^C + K_{MM}(\alpha^M + \Delta\alpha^M) + k_{M,c}(\alpha_c + \Delta\alpha_c) - (\rho + \Delta\rho)\boldsymbol{y_M} = 0$$
$$\langle \boldsymbol{y_M}, (\boldsymbol{\Delta\alpha_M} + \alpha_M)\rangle + \boldsymbol{y_c}(\boldsymbol{\Delta\alpha_c} + \alpha_c) + \langle y_E, \alpha_E\rangle + \langle y_C, \alpha_C\rangle = 0$$

Therefore, following conditions on the change of parameters

$$\langle \boldsymbol{y_M}, \boldsymbol{\Delta\alpha_M}\rangle + \boldsymbol{y_c\Delta\alpha_c} = \boldsymbol{0}$$
$$\boldsymbol{K_{MM}\Delta\alpha_M - \Delta\rho y_M + k_{M,c}\Delta\alpha_c = 0}$$

K$_{MM}$ is a matrix of $\left(y_i y_j \langle x_i^M, x_j^M\rangle\right)$ – inner products of Margin set elements, multiplied by their class labels.

k$_{M,c}$ is a vector of $\left(y_c y_j \langle x_c, x_j^M\rangle\right)$. Vector form of the conditions 1) and 2) could be represented as follows:

$$Q'_{MM}\begin{bmatrix}-\Delta\rho \\ \Delta\alpha_M\end{bmatrix} = -\begin{bmatrix}y_c \\ k_{M,c}\end{bmatrix}\Delta\alpha_c, \quad Q'_{MM} = \begin{bmatrix}0 & y_M^T \\ y_M & K_{MM}\end{bmatrix}$$

Therefore, similarly to the standard one-class SVM case

$$\Delta\rho = -\beta_\rho(x_c)\Delta\alpha_c, \quad \Delta\alpha_j = \beta_{M,j}(x_c)\Delta\alpha_c$$

Where

$$\begin{bmatrix}\beta_\rho(x_c) \\ \beta_M(x_c)\end{bmatrix} = -Q'^{-1}_{MM}\begin{bmatrix}y_c \\ k_{M,c}\end{bmatrix} \tag{17}$$

The change of the margin value for the learning set vectors is expressed as follows:

$$\Delta M = \left(\begin{bmatrix}y_C, K_{MC} \\ y_M, K_{MM} \\ y_E, K_{ME}\end{bmatrix} * \begin{bmatrix}\beta_\rho(x_c) \\ \beta_M(x_c)\end{bmatrix} + \begin{bmatrix}k_{C,c} \\ k_{M,c} \\ k_{E,c}\end{bmatrix}\right) * \Delta\alpha_c$$

The formulas provides a rule for the changes of the Lagrange multipliers of the M-set depending on the changes of α_c. One can see, that the change is proportional to the $\Delta\alpha_c$,where proportion coefficient is denoted as

$$\tau(x_c, x_i) = \left[y_i, k_{M,i}^T\right] * \begin{bmatrix}\beta_\rho(x_c)\\ \beta_M(x_c)\end{bmatrix} + k_{i,c}$$

Here $k_{M,i}$ and $k_{i,c}$ denotes a vector(value) of inner product(s) between x_i and M or x_c. The limitations over $\Delta\alpha_c$ are as follows:

1) All vectors in the Error Set have a negative margin:

$$E_+ : \{x_i \in E \mid \tau(x_c, x_i) > 0\}\ \Delta\alpha_c \leq \min_{x_i \in E_+}\left[-\frac{M(x_i)}{\tau(x_c, x_i)}\right]$$

2) All vectors in the Correct Set have a positive margin:

$$C_-: \{x_i \in C \mid \tau(x_c, x_i) < 0\}\ \Delta\alpha_c \leq \min_{x_i \in C_-}\left[-\frac{M(x_i)}{\tau(x_c, x_i)}\right]$$

3) All components of α_M satisfy the limitations of (16):

$$M_+: \{x_i \in M \mid \beta_{M,i}(c) > 0\}\ \Delta\alpha_c \leq \min_{x_i \in M_+}\left[\frac{C(y_i) - \alpha_i}{\beta_{M,i}(x_c)}\right]$$

$$M_-: \{x_i \in M \mid \beta_{M,i}(c) < 0\}\ \Delta\alpha_c \leq \min_{x_i \in M_-}\left[\frac{-\alpha_i}{\beta_{M,i}(x_c)}\right]$$

4) The α_c satisfies the limitations of (16):

$$\Delta\alpha_c \leq C(y_c) - \alpha_c$$

According to the given limitations, maximal possible $\Delta\alpha_c$ is chosen. When it is found and the values of Lagrange multipliers are changed, one of the events 1)-6) described in section III occurs, the actions of the OSA algorithm are similar to the standard SVM. The updates of the parameters of the model proceeds until conditions 5) or 6) are satisfied.

5 Tests on Benchmark Data Sets

To validate the proposed algorithm we applied it to a number of well-known benchmark datasets comparing it against both offline (SVM, C4.5, kNN) and online (eClass0 and eClass1) classifiers. The chosen classification problems are, in fact, multiclass problems as datasets for one-class classification problems are not widely represented. But these problems can be easily turned into one-class with one of the classes assigned as base and the remaining classes considered as outliers.

Each dataset was split into training set and control set. Training sets were used at the learning step while classifiers' performance was evaluated by the control set. We will not discuss tuning of the known algorithms concentrating on the proposed algorithm instead. For the incremental one-class SVM a Gaussian kernel was chosen which is a common practice. Its single parameter γ, determining the spread of the base class, was determined by a cross-validation procedure. The Pareto-optimal γ, bringing minima to FN and FP rates, was chosen for each dataset. SVM parameter ν was set to 0.05 for all datasets, thus limiting FP rate on the training set. Table I shows total error of the compared algorithms on different datasets.

Table 1 Total error on benchmark datasets, in %

	Offline (batch mode)			**Online/incremental**		
	SVM	**C4.5**	**kNN**	**eClass0**	**eClass1**	**The proposed method**
Pima	31.03	26.37	25.55	30.62	23.36	26.33
Iris	10.53					10.53
Wine	5.77	7.87	3.06	7.56	2.78	5.77

On the well separable datasets Wine and Iris all the algorithms performed quiet similarly. On the Pima dataset the proposed incremental one-class SVM perfumed slightly better than the standard SVM. C4.5, kNN and both eClass methods were not applied to the Iris dataset, the corresponding fields are left blank. Table 1 shows that the proposed method provides competitive with standard methods results being, at the same time, online and incremental.

6 Tests on Real Flight Data

The proposed incremental one-class SVM method was applied to anomaly detection in flight data collected from 138 real Boeing 737-800 flights. Flight data, recorded in a binary format, can be hardly processed by standard offline methods as there are hundreds of aircraft parameters registered at frequencies up to 1000Hz. At the same time, as an aircraft operates in a changing environment 'normal' parameter values are changing as well. This makes online and incremental methods favorable.

Flight data, that is a set of parameter values registered at different frequencies, was preprocessed, in order to represent every flight as a multidimensional time series. All parameters were linearly interpolated to the same frequency. Traditionally Flight Data Analysis (FDA) is performed by legacy systems, such as AGS [17], determining periods of abnormal aircraft functioning (events). Their markup was used in this study for two purposes: to provide a bootstrap training set for the proposed algorithm, as well as to evaluate its performance on the full flight dataset. It should be mentioned that most of the events detected by the legacy systems are different kinds of pilot errors, hardly separable in the data space.

Two parameters of the proposed method, kernel spread γ and SVM ν-parameter, were determined by a cross-validation procedure identifying the dependence of different error rates from these parameters. Cross-validation was performed over flight data registered during "approach" and "final approach" phases, split into training and control sets. FP and FN rates on the control set are shown in Figure 1 and Figure 2, respectively, representing these rates as functions of parameters γ and ν.

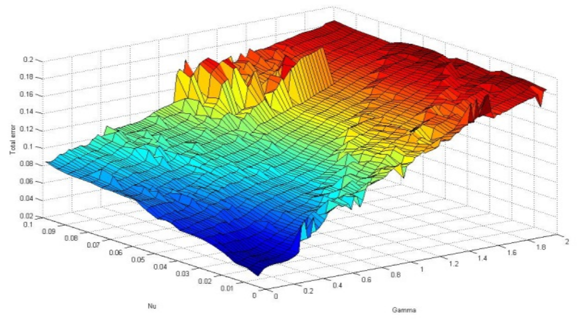


Fig. 1 FP rate as a function of $\boldsymbol{\nu}$ and $\boldsymbol{\gamma}$

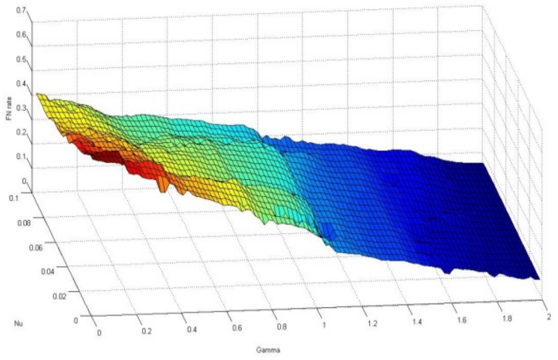


Fig. 2 FN rate as a function of $\boldsymbol{\nu}$ and $\boldsymbol{\gamma}$

Cross-validation showed, and it can be seen in Figure 1 and Figure 2, that the dependence of FP and FN rates on the algorithm's parameters is close to linear. But FN rate can be decreased only by the cost of FP rate growth. The impact of the change of the γ-parameter is stronger in comparison to the ν-parameter.

The maximum acceptable FP rate was defined by flight safety experts. Thus algorithm's parameters were adopted to decrease the FN rate with the limited FP rate.

To compare standard SVM approach [6] with the proposed approach twenty flights were chosen as training set and both SVM models (offline and the proposed one) were fitted to this data. The cross-validation procedure provided the following parameter setting:

$$\nu_1 = 2, \nu_2 = 0.1, \gamma = 0.3$$

The algorithms' performance was evaluated by the resting 118 flights.

The proposed incremental SVM approach is computationally demanding, as the Marginal, Error and Correct sets are increasing at every iteration of the algorithm. To reduce this computational complexity, the size of the set, containing correctly classified samples was limited by 1000 vectors. Thus the model, fitted over several million vectors, contained only 3000 support vectors, which is acceptable for the available computational resources.

SVM and the proposed method performance is represented by ROC-curves in Figure 3. The curves are built by varying the threshold parameter ρ of each of the models.

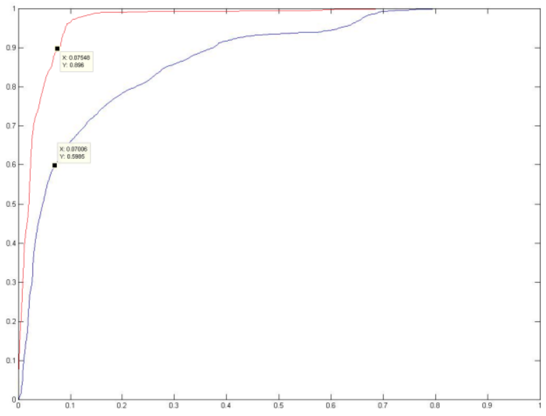


Fig. 3 ROC curves, SVM (blue) and the proposed approach (red)

The horizontal axis represents the FP rate; the vertical axis represents the true-positive rate, which can be calculated as $1 - FN$. The red line indicates the performance of the proposed method, the blue line – of the standard SVM.

ROC-curves show that the results for the proposed incremental one-class SVM are significantly better comparing to the standard SVM. An acceptable 5% of FP for the standard SVM approach allows achieving only 60% of correct alarms, therefore 40% of FN is obtained. The proposed method allows, for the same 5% of FP, getting 85% of true-negative rate, or 15% of FN.

7 Conclusion

In this work a new type of incremental, online SVM approach is proposed. It builds upon recently reported online recursive one-class SVM approach [10] by generalizing to also include in considerations the "negative" (faulty) data samples as well as the "normal" ones, but is still one-class classification type. It was tested on a number of benchmark data sets as well as on real flight data. Cross-validation procedure was used to identify the parameters of the SVM algorithm. The results presented in this paper show that the developed algorithm is superior to existing analogues. The next step in the improvement of the results will be a reduction FN rate. As the results indicate a very promising approach to achieve this aim is to use autonomously learning classifiers such as eClass [3,13,19].

Acknowledgements. The research led in this project has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° ACPO-GA-2010-265940, “SVETLANA”.

References

[1] Vapnik, V., Golowich, S.E., Smola, A.J.: Support Vector Method for Function Approximation, Regression Estimation and Signal Processing. In: Paper presented at the meeting of the Advances in Neural Information Processing Systems 9 — Proceedings of the 1996 Neural Information Processing Systems Conference (NIPS 1996), Dever, CO, USA (1997)

[2] TU-204-100 Aircraft Operations Manual / Tupolev ANTK (1998)

[3] Angelov, P.: Anomalous System State Identification, GB1208542.9 (priority date May 15, 2012)

[4] Koppel, M., Schler, J.: Authorship verification as a one-class classification problem. In: Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004), p. 62. ACM, New York (2004)

[5] Gretton, A., Desobry, F.: On-Line One-Class Support Vector Machines. An Application to Signal Segmentation. In: IEEE ICASSP, vol. 2, pp. 709–712 (2003)

[6] Gâlmeanu, H., Andonie, R.: Implementation Issues of an Incremental and Decremental SVM. In: Kůrková, V., Neruda, R., Koutník, J. (eds.) ICANN 2008, Part I. LNCS, vol. 5163, pp. 325–335. Springer, Heidelberg (2008)

[7] Das, S., Oza, N.C.: Sparse Solutions for Single Class SVMs: A Bi-Criterion Approach. In: Proc. SDM, pp. 816–827. SIAM, Omnipress (2011)

[8] Jensen, S.: An Introduction to Lagrange Multipliers, `http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html`

[9] Gershwin, S.B.: KKT - Examples. MIT OpenCourseWare (2010)

[10] Tax, D.M.J., Laskov, P.: Online SVM learning: from classification to data description and back. Journal of Machine Learning Research 7, 1909–1936 (2006)

[11] Hill, S.I., Doucet, A.: Adapting Two-Class Support Vector Classification Methods to Many Class Problems. In: Proc. ICML 2005 Proc. 22nd Intern. Conf. on Machine Learning, pp. 313–320 (2005)

[12] Scholkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C.: Estimating the support of a high dimensional distribution. Neural Computation 13(7), 1443–1471 (2001)

[13] Angelov, P.: Autonomous Learning Systems: From Data Streams to Knowledge in Real time. Willey (December 2012) ISBN: 978-1-1199-5152-0

[14] Bishop, C.: Machine Learning and Pattern Classification, 2nd edn. Springer (2009)

[15] Stibor, T., Timmis, J.I., Eckert, C.: A Comparative study of real-valued negative selection to statistical anomaly detection techniques. In: Jacob, C., Pilat, M.L., Bentley, P.J., Timmis, J.I. (eds.) ICARIS 2005. LNCS, vol. 3627, pp. 262–275. Springer, Heidelberg (2005)

[16] TU-204-100 Aircraft Operations Manual / Tupolev ANTK (1998)

[17] Analysis Ground Station (AGS), Sagem/SAFRAN, France, `http://www.sagem-ds.com/spip.php?rubrique230` (accessed November 8, 2012)

[18] Kolev, D., Zvikhachevskiy, D., Suvorov, M., Angelov, P.: Safety (and maintenance) improvement Through automated flight data analysis March 19, 2012, Scale Focused Research Project for project SVETLANA, Grant Agreement: ACPO-GA-2010-265940 (2012)

[19] Angelov, P., Zhou, X.: Evolving Fuzzy-Rule-based Classifiers from Data Streams. IEEE Trans. on Fuzzy Systems 16(6), 1462–1475 (2008)

Inertial Gesture Recognition with BLSTM-RNN

Grégoire Lefebvre, Samuel Berlemont, Franck Mamalet, and Christophe Garcia

Abstract. This chapter presents a new robust method for inertial MEM (MicroElectroMechanical systems) based 3D gesture recognition. The linear acceleration and the angular velocity, respectively provided by the accelerometer and the gyrometer, are sampled in time resulting in 6D values at each timestep, which are used as inputs for our gesture recognition system. We propose to build a system based on Bidirectional Long Short-Term Memory Recurrent Neural Networks (BLSTM-RNN) for gesture classification from raw MEM data. We compare this system to a statistical method based on HMM (Hidden Markov Model), to a geometric approach using DTW (Dynamic Time Warping), and to a specific classifier FDSVM (Frame-based Descriptor and multi-class Support Vector Machine) using filtered and denoised MEM data. Experimental results, on a dataset composed of 22 individuals producing 14 gestures, show that the proposed approach outperforms classical methods with an average classification rate of 95.57% and a standard deviation of 0.50 for 616 test gestures. Furthermore, these experiments underline that combining accelerometer and gyrometer data gives better results than using a single inertial description.

Grégoire Lefebvre · Samuel Berlemont
Orange Labs, R&D, Meylan, France
e-mail: {gregoire.lefebvre,samuel.berlemont}@orange.com

Franck Mamalet
Orange Labs, R&D, Rennes, France
e-mail: franck.mamalet@orange.com

Christophe Garcia
LIRIS, UMR 5205 CNRS, INSA-Lyon, F-69621, France
e-mail: christophe.garcia@liris.cnrs.fr

P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_19

1 Introduction

Accelerometers and gyrometers are nowadays present in our Smartphones. These sensors capture hand movements when users grasp their devices. We can consider two main issues: posture recognition and symbolic gesture recognition. In the first case, the user maintains a posture during a certain period of time, keeping for instance the device upside down. In the second situation, the user may produce a gesture to execute a system command, like drawing a *heart* symbol in 3D space in order to call its favorite phone number. Dynamic gesture recognition based on inertial sensors is a very challenging task. Algorithms are confronted to numerous factors causing errors in the recognition process: dynamical differences (intense versus phlegmatic gestures), temporal variations (slow versus fast movements), physical constraints (device weight, human body elasticity, left or right-handed users, seated or standing up users, users on the move, etc.), classification constraints (mono versus multi users, open or closed world paradigm, etc.). Moreover, inertial MEM introduce some noise in the data. For example, the accelerometer, as a 3-axis capacitive mass spring system, interpret gravity as an acceleration in the opposite direction. This constant bias is difficult to extract due to the reference frame of the sensor, which is fixed in relation to the device (see [23] for more details).

Classically, several steps operate from signal data processing to gesture classification with some intermediate steps like data clustering and gesture model learning. The processing steps aim at building a more compact representation of the input signals that characterize the corresponding gestures. Different methods can then be applied: calibration, filtering, normalization or thresholding. Data clustering is often applied to reduce the input space dimension and find class referent gesture vectors. A learning phase of a gesture model follows this clustering step and finally a decision rule or a specific classifier is built in order to label the input data as a recognized gesture or an unknown gesture.

In this chapter, we propose to learn an efficient gesture classifier without any preprocessing method (*i.e.* from raw MEM data) using a BLSTM-RNN model. This chapter is organized as follows. Section 2 presents a survey of sensor-based gesture recognition. In section 3, we describe in detail the proposed gesture recognition method. We present in depth experimental results in section 4. Finally, conclusions and perspectives are drawn in section 5.

2 Accelerometer Based 3D Gesture Recognition

3D gesture recognition using accelerometers has been studied in recent years, and for gesture classification three main strategies stand out which are based on statistics, on geometry or on boosting classifiers.

The first strategy has been deeply studied in the last decade, the methods being based on Hidden Markov Models (HMM) [12, 13, 14, 21] and Bayesian networks [4]. Hofmann *et al.* [12] propose to use discrete HMM (dHMM) for recognizing dynamic gestures thanks to their velocity profile. This approach consists of two levels and stages of recognition: a low-level stage essentially dividing the input data space into different regions and assigning each of them to a representative codebook, and a high-level stage taking the sequences of vector indexes from the first stage and classifying them with discrete HMM. The experiments are performed using a training set of 500 gestures with 10 samples per gesture realized by two persons. Each sample represents hand orientations, acceleration data and finger joint angles. A vector codebook is obtained by an input space clustering method (*i.e.* the K-means algorithm). The clustering essentially serves as an unsupervised learning procedure to model the shape of the feature vector distribution in the input data space. Here, the observation alphabet size equals 120. The comparison between ergodic HMM and left-to-right HMM shows similar results with 95.6% of correct recognition rate for 100 gestures from two users describing German Sign Language.

Hand gesture recognition based on HMM is also proposed by Mantyla *et al.* [18]. Their approach differs by a uniform vector quantization strategy based on 512 3D codewords. Here, minimum and maximum values of each acceleration component were evaluated using the training data in order to define the bounding box of the activity in the feature space. The experimental results show an average correct recognition rate of 96% on a small dataset of six gesture categories. The input acceleration data is low-pass Butterworth filtered after $100Hz$ sampling.

Similar results are presented in [13, 14]. Kallio *et al.* [13] use five HMM states and a codebook size of eight for 16 gestures. The authors highlight that the performances decrease when using four sequences for training the system instead of 20 sequences. The recognition rate falls from 95% to 75% even for this mono-user case study. In [14], a 37 multi-user case is studied for 8 gestures, evaluating the effect of vector quantization and sampling. A rate of 96.1% of correct classification is obtained with five HMM states and a codebook size of height. However, this study can be considered as biased since the K-means clustering is performed from all the available data set and not only the training database.

In opposition to the previous studies, and taking into consideration that gesture data are correlated in time, Pylvänäinen proposes in [21] to build a system based on continuous HMM (cHMM). Again, the results are convincing, with 96.76% on a dataset providing 20 samples for 10 gestures performed by 7 persons. An evaluation of the quantization and sampling effects is demonstrated, and the best performances are obtained by using $7-12$ bits per sample and a frequency from 20 to $30Hz$.

In [4], Cho *et al.* build Bayesian networks models based on the extraction, within the acceleration signals, of primitives which correspond to portions of the signals whose values increase or decrease monotonously. These primitives are recursively decomposed in mid-points models, represented by the Gaussian distributions of the samples whose time indexes are the mean of the extracted primitive end-points. Gestures are classified based on a maximum likelihood matching to the models created. Cho *et al.* show a mean correct recognition rate of 96.3% with a 4-fold test, with 11 gestures carried out 3 times by 100 users.

The second strategy for recognizing 3D gestures is based on distances between geometric models. The goal is to provide a gallery of some gesture references to model each gesture into class with time warping strategies and design a decision rule for a test gesture regarding the respective distance to these referent instances. On the contrary to the HMM strategy, no learning phase is needed but the computational time is high as a test gesture has to be compared to all referent instances. Consequently, the main drawback of this approach is the necessity to find the most relevant samples to represent a gesture class while keeping the number of these referents low in order to minimize the final evaluation processing time. Wilson *et al.* in [22] compare Linear Time Warping (LTW) and Dynamic Time Warping (DTW) to the HMM based strategy. Their experiments with 7 types of gesture from 6 users shows an advantage for HMM with 90% in opposition to the score of LTW and DTW of respectively 40% and 71% of correct recognition rate.

Liu *et al.* experiment with more success the DTW strategy in [17]. Gesture recognition and user identification are performed with good recognition rates of respectively 93.5% and 88%. The gesture data, performed over multiple days, consists of 30 samples of 8 gestures for 8 individuals and the user recognition results are obtained from 25 participants. The authors introduce an averaging window of 50 ms for reducing noise and erratic moves. It appears that preprocessing methods are here crucial to increase the classification results. Likewise, in [1], Akl *et al.* use DTW and affinity propagation for dimension reduction for recognizing 3D gestures. 7 subjects participated producing 3700 gesture traces for a good classification rate of 90%.

The third strategy for recognizing 3D gestures is to learn a specific classifier. Hoffman *et al.* in [11] improve 3D gesture recognition with a linear classifier and the Adaboost method, inspired by the method proposed in [15] for 2D symbol writer recognition. The experiments show an accuracy of 98% for 13 gestures made by 17 participants. Other studies focus on Support Vector Machine (SVM) like in [24]. This study uses Frame-based Descriptor and multi-class SVM (FDSVM). Each gesture is divided into segments, from which are extracted statistical descriptors such as: mean, energy, entropy, standard deviation and correlation. These descriptors form the feature vectors to be classified by a multi-class SVM. The obtained results achieve 95.21% of good recognition for 12 gestures made by 10 individuals.

In [4], Cho *et al.* also use a confusion pair discrimination strategy, enhancing their 96.3% average recognition rate to 96.9% with a bi-class SVM (*i.e.* letter *O* versus number 6), in order to better separate similar gestures, which generate most of the recognition errors.

Consequently, many strategies are explored with different paradigms and specific data processing methods. However, theses approaches depend heavily on the choice of descriptors which may not be optimal in general. Moreover, it is challenging to compare these methods on different databases. To cope with these issues, we develop hereafter our 3D gesture recognition method based on BLSTM-RNN that learns to automatically extract relevant features from raw input data and compare it with the main state-of-the-art methods on a common database.

3 The Proposed Gesture Recognition Method

3.1 Bidirectional Long Short-Term Memory Recurrent Neural Networks

Classical Recurrent Neural Networks (RNNs) are a common learning technique for temporal sequence analysis. It can be seen as a particular neural network, which can remember previous inputs and use them to influence the network output, using recurrent connections in the hidden layers to store some information about the context. Nevertheless, even if they are able to learn tasks which involve short time lags between inputs and corresponding teacher signals, Hochreiter and Schmidhuber in [10] have shown that this short-term memory becomes insufficient when dealing with "real world" sequence processing, such as inertial gesture sequences.

In order to alleviate this problem, they introduced the Long Short Term Memory RNNs (LSTM-RNN), which neurons contain a constant "memory cell" – namely constant error carousel (CEC) –. This allows for constant error signal propagation through time, and thus, provides remedies for the RNN's problem of *exponential error decay* [10]. Figure 1 presents in detail a LSTM neuron.

Improved versions of the LSTM-RNN were proposed by Graves *et al.* [9], adding some multiplicative gates to control the access to the CEC. These gates are neurons that can set (input gate), reset (forget gate) or hide (output gate) the internal value of the CEC according to neuron input values and context. Additional direct connections – namely peephole – have also been introduced to achieve finer gate control using the CEC value as input.

Furthermore, since in sequence classification, at a given timestep, past and future context may have similar importance, Graves and Schmidhuber [9] also introduced a so-called bidirectional LSTM-RNN model (BLSTM-RNN), that consists in two separate hidden layers, the forward and backward layers able to deal respectively

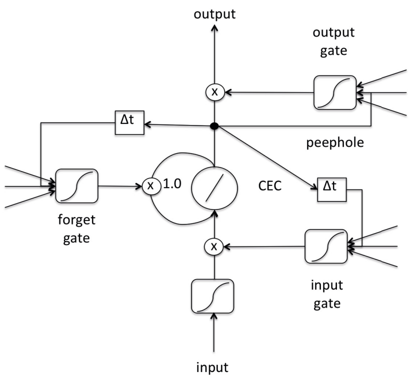


Fig. 1 A LSTM neuron: the Constant Error Carousel (CEC) is controlled by three multiplicative gates (input, forget and output). Peepholes enable to transmit directly the CEC value to the gate, at a given timestep.

with past and future context. As shown in Figure 2, the output layer is connected to both hidden layers in order to fuse past and future contexts.

Let's give the following notations:

- $G = \{G_0, ..., G_{T-1}\}$ is a gesture with T denoting the size of the sequence;
- $G_t = (x_0(t), ..., x_{N-1}(t))$ is a vector at timestep t and N being the sensor number;
- $(o_0(t), ..., o_{n-1}(t))$ is the BLSTM-RNN output set at timestep t with n is the number of gestures to be classified.

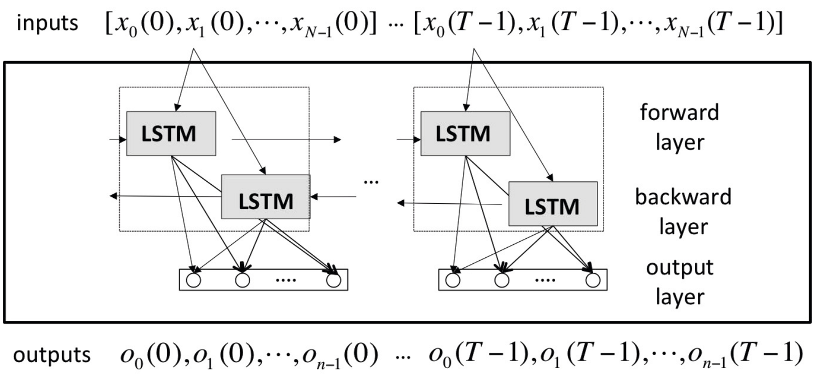

inputs $[x_0(0), x_1(0), \cdots, x_{N-1}(0)]$ ··· $[x_0(T-1), x_1(T-1), \cdots, x_{N-1}(T-1)]$
LSTM
LSTM
LSTM
LSTM
forward layer
backward layer
output layer
outputs $o_0(0), o_1(0), \cdots, o_{n-1}(0)$ ··· $o_0(T-1), o_1(T-1), \cdots, o_{n-1}(T-1)$

Fig. 2 A BLSTM-RNN architecture: the forward (resp. backward) layer processes the input sequence in the (resp. reverse) order. Output layer concatenates hidden layers values at each timestep to make a decision by considering both past and future contexts.

(B)LSTM-RNNs have proven their great ability to deal with temporal data in many applications such as: phoneme classification [9], action classification [2], facial expression recognition [3], rhythm or timed event recognition [6], robot trajectory learning [19], handwriting recognition [8, 7], or text recognition [5].

In this chapter, we consider gesture data using 6D input vectors through sampling timestep. These data are correlated during the user gestural production, and time lags between the beginning and the end of gesture can be long. Furthermore, past and future context are both essential for gesture recognition. For these reasons, BLSTM-RNN is chosen to classify the input MEM data sequence.

3.2 BLSTM-RNN Architecture, Training and Decision Rule

The proposed gesture classification scheme based on BLSTM-RNN is described in Figure 3. First, the input layer consists in the concatenation of 3-axis accelerometer and 3-axis gyrometer information synchronized in time (*i.e.* $N = 6$). Notice that our system relies only on the raw MEMs data, without any preprocessing in opposition to most of state-of-the-art methods. These data are only rescaled between -1 and $+1$ according to the maximum value that sensors can provide.

The forward and backward LSTM hidden layers are fully connected to the input layer and consist in multiple LSTM neurons each with full recurrent connections. Several experiments have been conducted with different hidden layer sizes and 100 neurons lead to the best results (cf. section 4.2).

The output layer has a size equals the number of gesture to classify (*i.e.* $n = 14$ in our experiments) . The SoftMax activation function is used for this layer to give network responses $o_i(t)$ between 0 and 1, and a sum $\sum_{i\in[0,n[} o_i(t)$ equals one, at every timestep t. Classically, these outputs can be considered as posterior probabilities of the input sequence to belong to a specific gesture class.

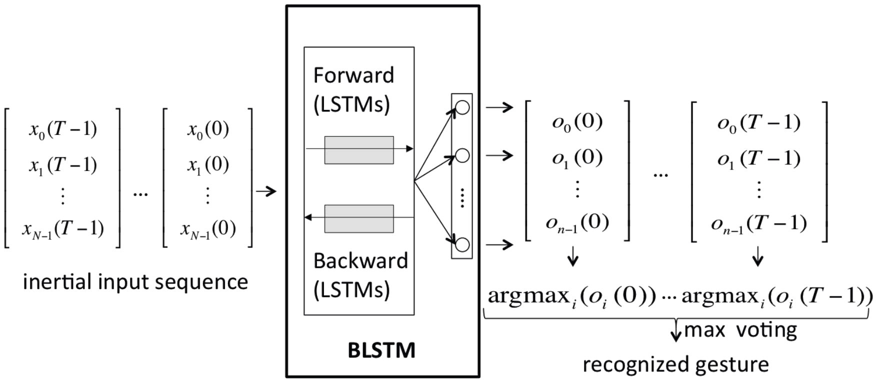

Forward (LSTMs)
Backward (LSTMs)
BLSTM
inertial input sequence
$x_0(T-1)$ $x_1(T-1)$ $x_{N-1}(T-1)$
$x_0(0)$ $x_1(0)$ $x_{N-1}(0)$
$o_0(0)$ $o_1(0)$ $o_{n-1}(0)$
$o_0(T-1)$ $o_1(T-1)$ $o_{n-1}(T-1)$
$\text{argmax}_i(o_i(0)) \cdots \text{argmax}_i(o_i(T-1))$
max voting
recognized gesture

Fig. 3 BLSTM-RNN gesture classification architecture: input layer is a 6D inertial sequence. The largest BLSTM-RNN output is computed at each timestep and a Max voting scheme enables to determine the recognized gesture class.

This network is learned using classical on-line backpropagation through time with momentum (*i.e.* learning rate $5e{-}4$, momentum 0.2, experimented in section 4.2). As described in [9], on a training set, we target the same corresponding gesture class at each timestep for each input example. For classifying a new gesture sequence, we use a majority voting rule over the outputs along the sequence (*i.e.* keeping only the most *probable* class $argmax_{i\in[0,n[} o_i(t)$ at each timestep) to determine the final gesture class.

4 Experimental Results

4.1 Inertial Gesture Dataset

To the best of our knowledge, the community does not provide yet any public dataset for benchmarking inertial symbolic gesture recognition methods. Therefore, we collected our own 3D gesture dataset to compare our classification method with classical approaches in this research field. Our dataset has been captured on an Android

Nexus S Samsung device. The sampling time is $40ms$ during the accelerometer and gyrometer capture (*i.e.* a frequency of $25Hz$). 22 novice participants, from 20 to 55 years old, all right-handed, performed five times each of the 14 symbolic gestures that we designed. This corresponds to 1540 gestures. As shown in Figure 4, the 14 symbolic gestures are divided into two families: linear gestures (*e.g. north*, *east*, *west* and *south flicks*, and *down*, *up*, *throw* and *pick* gestures) and curvilinear gestures (*e.g. clockwise*, *counter-clockwise*, *letter Z*, *letter N*, *alpha* and *heart*). These choices make the dataset particularly difficult. For instance, there classically are confusions between *flick* gestures and *letter N* and *Z*. Likewise, the *clockwise* movement is often confused with *alpha* or *heart* symbols.

In order to locate and use only the informative part of the input sequence for performing gesture classification, an automatic temporal segmentation is performed. As described in [16], this temporal segmentation is based on a signal energy indicator obtained by the difference of two instant power estimators. A gesture indicator $I(t)$ can be defined from two estimators $E_0(t)$ and $E_1(t)$ of the instant signal power $P(t)$ (see Equation 1). Some heuristics on this gesture indicator identify the beginning and the end of a symbolic gesture before processing the recognition phase.

$$\begin{cases} P(t) &= \frac{1}{2}\sqrt{\sum_{i=0}^{N-1} x_i(t)} \\ E_0(t) &= \delta_0 E_0(t-1) + (1-\delta_0)P(t), \delta_0 \in [0;1] \\ E_1(t) &= \delta_1 E_1(t-1) + (1-\delta_1)P(t), \delta_1 \in [0;1], \delta_0 > \delta_1 \\ I(t) &= |E_0(t) - E_1(t))| \end{cases} \tag{1}$$

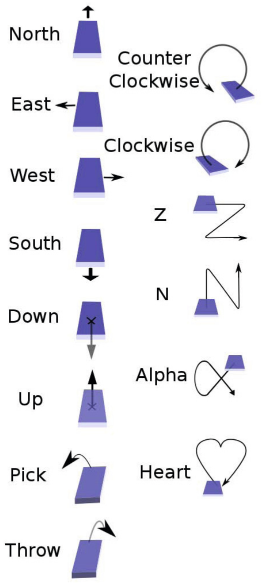
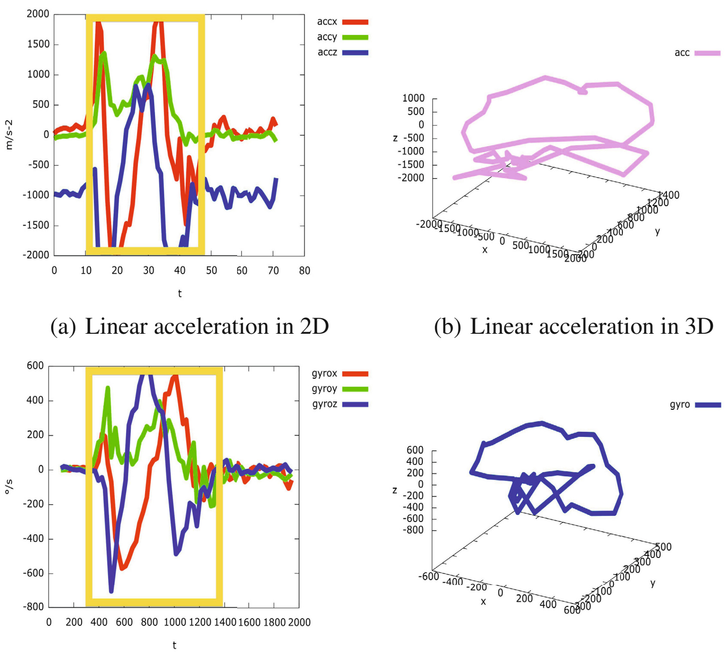


Fig. 4 Gesture dataset : the 14 symbolic gestures to be classified.

For instance, Figure 5 shows raw input signals to be classified as a clockwise gesture and the yellow parts correspond to segmented gestures. Six degrees of freedom are then provided by the accelerometer and the gyrometer with respectively linear accelerations and angular velocities. In Figure 6, we show the 3D gesture trajectory reconstruction using Simpson's 3/8 rule for numerical integration.



(a) Linear acceleration in 2D (b) Linear acceleration in 3D

(c) Angular velocity in 2D (d) Angular velocity in 3D

Fig. 5 Clockwise gesture sample : accelerometer and gyrometer features to be classified. The yellow parts correspond to segmented gestures.

We then use three different configurations of our dataset to compare several state-of-the-art methods with our solution. The first configuration (DB1) corresponds to the personalization paradigm, where only one user is considered with few learning examples. For this configuration we have used the 70 gestures of a participant in the learning phase, and ask him to process 16 more instances of each gesture for test (*i.e.* 224 gestures). The second configuration (DB2) uses three instances of each gesture per user for the learning phase: 924 gestures (*i.e.* 60% of all data) are used for the learning phase and 616 gestures (*i.e.* 40%) for the test phase. This case corresponds to a multi-user system and a closed world paradigm. The third configuration (DB3) is composed of all samples from 17 users (*i.e.* 1190 gestures) and the test data uses the other available gestures (*i.e.* 350 gestures from five unknown users). This case is close to a real system trained with a several users and having to generalize to new

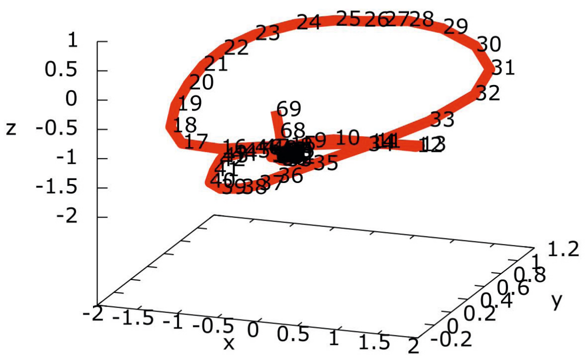


Fig. 6 Clockwise trajectory reconstruction using Simpson's 3/8 rule for numerical integration

users who want to use it without any training phase. Here, the third configuration represents the open world paradigm.

Our experimental results aim at assessing the method based on BLSTM-RNN and compare it to three state-of-the-art solutions: cHMM[21], DTW[20] and FDSVM [24] methods. These three state-of-the-art solutions represent three main strategies which are based on statistics, on geometry or on boosting classifier approaches. We also compare our results obtained with BLSTM-RNN and LSTM-RNN methods.

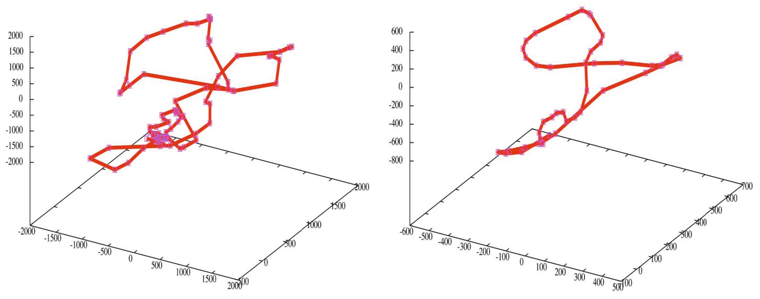
For the three state-of-the-art solutions and in all experiments, we use normalized, filtered and thresholded gesture signals. In opposition, LSTM-RNN and BLSTM-RNN based solutions use raw MEM data. As shown in Figure 7, the main objective for classical gesture recognition methods is to operate on preprocessed input signal for facilitating gesture recognition. From the raw data, some signal processings are operated to filter, normalize and threshold the gesture data.

We define (*cf.* Section 3 for notations) the normalized signal $x_i^{norm}(t)$ by Equation 2 being a weighting by the inverse of the maximal Euclidean norm for every gesture sequence element $x_i(t)$. Equation 3 gives the filtered signal $x_i^{filter}(t)$ with a low-pass filter of smoothing parameter β. Equation 4 is the condition to keep an original data in a thresholding process (*i.e.* two consecutive data differ from a measure Δ).

$$\forall i \in \{0,...,n-1\}, x_i^{norm}(t) = \frac{x_i(t)}{max_{t=0}^{T-1} \|G_t\|}, \tag{2}$$

$$\forall i \in \{0,...,n-1\}, x_i^{filter}(t) = \beta x_i^{filter}(t-1) + (1-\beta) x_i^{norm}(t), \beta \in [0,1], \tag{3}$$

$$\forall t \in \{0,...,T-2\}, \|G_t - G_{t+1}\| > \Delta, \Delta \in \Re. \tag{4}$$



(a) Raw acceleration input signal (b) Normalized, filtered and thresholded acceleration signal

Fig. 7 An *alpha* gesture with raw MEM data (a) and after preprocessing methods (b)

4.2 Preliminary Classification Results

We have conducted some experiments to explore the BLSTM architecture and find the best configuration and parameters (*i.e.* learning rate and momentum). Figure 8 gives some primarily results with different BLSTM-RNN architectures and parameters on DB3. Here, the inertial gesture recognition task is performed with the better results obtained for a configuration of 100 LSTM neurons with a learning rate of $5e{-}4$ and a momentum of 0.2.

In order to obtain the best results for the state-of-the-art methods, we have also made some primarily experimentations to find the best parameters:

- The filtering process use a low-pass filter with a smoothing parameter β of 0.8 ;
- The thresholding process approximates a minimal local error Δ of 0.1 ;
- The cHMM based method is left-to-right and uses the maximum of likelihood as a decision rule and is composed of nine states using Gaussian distribution ;
- The DTW based method uses a 5-nearest-neighbor classification ;
- The FDSVM based method applies polynomial kernel SVMs on 19-dimensional feature descriptors (mean, energy, entropy, standard deviation and correlation) for nine time segments with six signal dimensions, *i.e.* feature vectors of 1026 dimensions.

4.3 Classification Results

In the following, we use a three fold cross validation to obtain mean and standard deviation, as shown in Table 1, which outlines the global performances of each classifier for configurations DB1, DB2 and DB3 using either accelerometer or gyrometer data or both.

Firstly, when comparing these methods using a single input MEM sensor (accelerometer or gyrometer), we can see that using only the gyrometer data is less

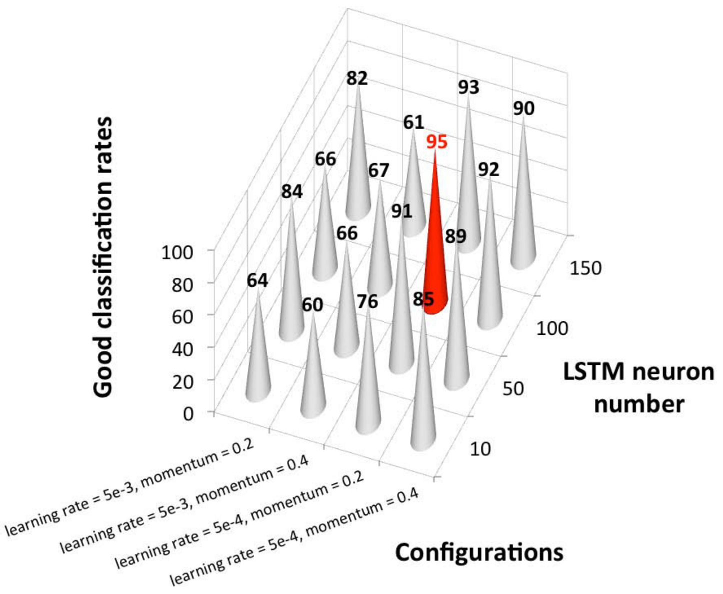


Fig. 8 Recognition rates for different BLSTM-RNN architectures on DB3 with multiple learning configurations.

efficient than using the single accelerometer data. Moreover, when these two information are combined, the global performances increase.

Secondly, considering coupled input data, this table shows that our BLSTM-RNN based classifier gives the best results on DB2 and DB3, with respectively $95.57\% \pm 0.50$ and $92.57\% \pm 2.85$.

On DB1 configuration, the DTW and cHMM based methods achieve the equivalent best performances (*i.e.* the mean recognition rates are upper than 99%, due to the strong similarity between test and training gestures provided by the same user) while our BLSTM-RNN approach is less efficient with $86.75\% \pm 0.75$. This is mainly due to the reduced number of training data which leads to the classical neural network over-fitting issue. The attempts made with smaller BLSTM-RNN did not allow any improvement on generalization.

In comparison, the FDSVM solution provides good and stable performances with an average recognition rate greater than 92% on the three datasets, but does not achieve the best results.

The main conclusions of a deep analysis of confusion matrices on DB3 (*i.e.* open world paradigm) are the following. The main drawback for the cHMM based method in this context is the incorrect classification of some linear gestures as *down*, *throw* and *up* gestures. 48% of the *down* and *throw* gestures are confused with *N* gestures and 52% of the *up* gestures are indeed confused with *pick* gestures. These gestures share indeed some features that proved to be difficult to distinguish for this method.

Table 1 Classification rates on DB1, DB2 and DB3 using accelerometer (acc), gyrometer (gyr), and both sensors (acc+gyro)

Databases	DB1	DB2	DB3
Methods	Mean & Standard Deviation		
cHMM acc	99.02% ± 0.81%	83.99% ± 1.09%	80.09% ± 2.82%
cHMM gyro	95.05% ± 2.62%	70.92% ± 0.74%	70.76% ± 0.58%
cHMM acc+gyro	**99.86% ± 0.20%**	85.79% ± 0.67%	82.76% ± 1.41%
DTW acc	99.40% ± 0.21%	92.59% ± 0.20%	90.29% ± 2.07%
DTW gyro	95.39% ± 0.56%	80.63% ± 2.39%	79.81% ± 1.72%
DTW acc+gyro	99.70% ± 0.42%	94.04% ± 0.15%	91.71% ± 1.46%
FDSVM acc	94.94% ± 2.42%	93.07% ± 0.88%	91.56% ± 0.28%
FDSVM gyro	92.26% ± 1.38%	85.66% ± 0.47%	84.52% ± 0.48%
FDSVM acc+gyro	96.38% ± 1.89%	95.39% ± 0.57%	92.38% ± 1.27%
LSTM acc	86.46% ± 5.83%	94.29% ± 1.07%	89.14% ± 2.03%
LSTM gyro	77.53% ± 5.26%	81.54% ± 1.79%	72.29% ± 1.30%
LSTM acc+gyro	88.10% ± 3.72%	95.18% ± 0.88%	92.47% ± 1.34%
BLSTM-RNN acc	84.15% ± 0.67%	94.86% ± 1.23%	89.42% ± 2.45%
BLSTM-RNN gyro	68.90% ± 4.85%	83.39% ± 0.65%	74.19% ± 1.55%
BLSTM-RNN acc+gyro	86.75% ± 0.75%	**95.57% ± 0.50%**	**92.57% ± 2.85%**

On the contrary, the DTW based method provides a good solution to classify linear gestures except for *throw* gestures which are often recognized as *east*, *north flick* and *pick* gestures, which can be explained by the similar nature of production of these three gestures.

The FDSVM method achieves good performances in general. Nevertheless, some misclassifications appear for instance between opposite gestures as *pick* and *throw* or *down* and *up* gestures. Opposite gestures may be misclassified for instance when some users anticipate a *throw* gesture by slightly moving back the device in the beginning of the production as in a *pick* gesture.

Our BLSTM-RNN approach (see Table 2) have some issues distinguishing respectively the *letter N* and the *up* gesture from the *up* and the *pick* gesture. This may be due to the uniform learning target chosen (*i.e.* same class at each timestep), or the majority voting scheme in the recognition phase.

4.4 Computing Times

Table 3 presents the computing times for all methods for the three configurations in recognition phase executed on an Intel Core i5 CPU at 2.67*GHz* with 3.42*Go* of RAM. These experimental results show that the computing time for the HMM, FDSVM, LSTM-RNN and BLSTM-RNN based solutions is quite constant on the

Table 2 Best Confusion Matrix on DB3 for the BLSTM based method.

BLSTM	alpha	ccw	cw	flickE	flickN	flickS	flickW	heart	N	pick	down	throw	up	Z	recognition rate
alpha	**25**														1
ccw		**25**													1
cw			**24**											1	0.96
flickE				**24**			1								0.96
flickN					**23**	1						1			0.92
flickS						**25**									1
flickW				1			**24**								0.96
heart			1					**24**							0.96
N									**21**	1			3		0.84
pick										**25**					1
down											**25**				1
throw										1	1	**23**			0.92
up										2	1		**21**		0.88
Z														**25**	1
error rate	0	0	0.04	0.04	0	0.04	0.042	0	0	0.16	0.08	0.04	0.12	0.04	**0.96**

Table 3 Computing times (in *ms*) to classify one unknown gesture.

Databases	DB1	DB2	DB3
Training examples	70	924	1190
Test examples	224	616	350
cHMM accgyro	42.53±1.97	23.89±2.74	30.19±1.65
DTW accgyro	11.93±0.02	34.57±0.47	44.58±0.38
FDSVM accgyro	23.81±2.10	28.14±0.77	30.82 ±0.47
LSTM accgyro	27.98±0.90	26.52±0.77	25.42±0.84
BLSTM-RNN accgyro	30.47±0.23	31.12±0.57	29.56±0.48

different datasets (*i.e.* for instance around 30 ms for BLSTM-RNN and 43*ms* for cHMM to classify one input gesture on DB1). The learning process is indeed built off-line and, consequently, the recognition process from a learnt model is quite fast. On the contrary, the DTW based method requires to compare the test gesture with all training reference samples. That is why the computing time increases in average from 11.93*ms* for 70 learning samples to 44.58*ms* for 1190 learning samples. The DTW based method requires then a small number of reference gestures which makes it hard to cover all user gesture variations. This trade-off may reduce the performances.

Consequently, our proposed system based on BLSTM-RNN, achieving the best performances in multi-user configuration with a recognition computing time independent of training dataset size, is a very challenging solution.

4.5 Final Results

In order to assess more precisely our BLSTM-RNN based method, we investigate seven performance criteria which we quantified from to 0 to 5 in Figure 9 (*e.g.* the number 5 denotes a good performance regarding the specific criterion):

1. Classification: the overall performance regarding the classification rates over the three datasets (cf. Table 1);
2. Training time: the average time requested to learn all reference gestures (*i.e.* lower the training time, higher the grade);
3. Test time: the average time to test all unknown gestures (cf. Table 3);
4. Parametrization: the number of parameters to be specified to get a valuable gesture model (*i.e.* fewer parameters equals higher grade);
5. Fast recognition: the ability to classify an unknown gesture as soon as possible (*i.e.* before the gesture ending);
6. Personalization: the ability to add new user gestures to rebuild gesture models;
7. Universality: the ability to generalize gesture models without adding new user samples.

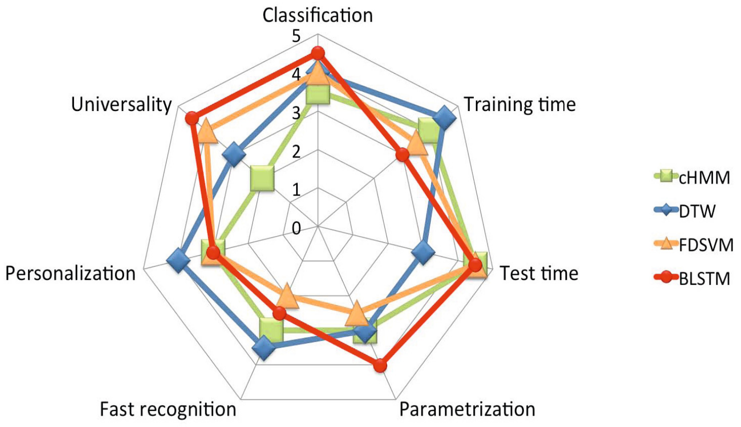


Fig. 9 Benchmark of the four main methods in regard to the seven criteria.

The BLSTM-RNN based method gives the best overall performance with the biggest covered area in the radar chart (cf. Figure 9). Therefore, with the best classification rates on two datasets and a competitive computing time for testing unknown gestures, this method offers good perspectives for universality issues, extracting gesture features from the raw signal data and dealing with strong gesture variations. The parametrization is also simplified by the choice of three criteria: the number of LSTM neurons, the learning rate and the momentum. Nevertheless, some difficulties remain when gestures should be classified as soon as possible (*i.e.* fast recognition). The temporal gesture segmentation must be indeed properly provided to maximize the classification results. The main drawbacks are when new gestures are added to the model for personalization purposes, given that the neural network must be learnt once again with a substantial computing time.

In opposition to the BLSTM based method, the three other methods perform better in term of training speed, but with less success on classification results. As previously exposed in section 4.4, the DTW based method requires also more time during test. Concerning the parametrization criterion, these three methods are more complex to tune with a need of a preprocessing step with specific parameters for filtering, normalizing and vectorizing input signals. Additional parameters are also inherent to each method. A fast recognition may be better realized with the DTW based method, computing a similarity between training samples and an unknown gesture even if the last one is incomplete. A fast recognition for the FDSVM based method is difficult because the whole input gesture must be segmented into frames in order to compute the final feature vector to be classified. For personalizing the system with new user gestures, the DTW based method seems preferable because only a selection of these instances is needed to be designed as new gesture references. Finally, Table 1 presents the universality criterion on DB3 with an open world paradigm. We show then that the best performance is obtained respectively in order by the BLSTM, FDSVM, DTW, and cHMM based method.

5 Conclusion and Perspectives

In this chapter, we have presented a contribution based on BLSTM-RNN for inertial MEM based gesture recognition. This study about symbolic gesture recognition compares our contribution to three classical pattern recognition methods: the geometric approach using DTW and the statistical method based on cHMM and a specific FDSVM classifier. We have shown that for a multi-user configuration our approach achieves the best average classification rates, up to 95.57%, in a closed world configuration, and up to 92.57%, in a open world configuration. Computing times are also very challenging for an industrial solution, being independent to the learning sample number. The main remaining confusions with the proposed solution are when two 3D trajectories are similar or share some initial movements. A new approach, ,height=5cmusing a modified objective function, such as a Connectionist Temporal Classification [9], that permits to jointly learn to localize and classify events in input sequences, might be used to overcome this issue or to classify non segmented gestures. Another interesting extension of this work will be to study how Deep Learning models like Convolutional Neural Networks can be used in our scheme to enhance the feature extraction stage by automatically learning sparse discriminant features from the raw data.

References

1. Akl, A., Valaee, S.: Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In: ICASSP (2010)
2. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Sequential Deep Learning for Human Action Recognition. In: Salah, A.A., Lepri, B. (eds.) HBU 2011. LNCS, vol. 7065, pp. 29–39. Springer, Heidelberg (2011)

3. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Spatio-Temporal Convolutional Sparse Auto-Encoder for Sequence Classification. In: BMVC (2012)
4. Cho, S.J., Choi, E., Bang, W.C., Yang, J., Sohn, J., Kim, D.Y., Lee, Y.B., Kim, S.: Two-stage Recognition of Raw Acceleration Signals for 3-D Gesture-Understanding Cell Phones. In: Lorette, G. (ed.) Tenth International Workshop on Frontiers in Handwriting Recognition. Université de Rennes 1, Suvisoft, La Baule, France (2006), `http://hal.inria.fr/inria-00103854`, `http://www.suvisoft.com`
5. Elagouni, K., Garcia, C., Mamalet, F., Sébillot, P.: Text Recognition in Videos using a Recurrent Connectionist Approach. In: Villa, A.E.P., Duch, W., Érdi, P., Masulli, F., Palm, G. (eds.) ICANN 2012, Part II. LNCS, vol. 7553, pp. 172–179. Springer, Heidelberg (2012)
6. Gers, F.A., Schraudolph, N.N., Schmidhuber, J.: Learning precise timing with lstm recurrent networks. J. Mach. Learn. Res. 3, 115–143 (2003)
7. Graves, A.: Offline Arabic Handwriting Recognition with Multidimensional Neural Networks. Springer (2012)
8. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE TPAMI 31(5), 855–868 (2009)
9. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks (18), 5–6 (2005)
10. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation (9), 1735–1780 (1997)
11. Hoffman, M., Varcholik, P., La Viola, J.: Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices. In: Virtual Reality Conference (VR), pp. 59–66 (2010), doi:10.1109/VR.2010.5444813
12. Hofmann, F.G., Heyer, P., Hommel, G.: Velocity profile based recognition of dynamic gestures with discrete hidden markov models. In: Wachsmuth, I., Fröhlich, M. (eds.) GW 1997. LNCS (LNAI), vol. 1371, pp. 81–95. Springer, Heidelberg (1998), `http://dx.doi.org/10.1007/BFb0052991`
13. Kallio, S., Kela, J., Mantyjarvi, J.: Online gesture recognition system for mobile interaction. In: Systems, Man and Cybernetics, vol. 3, pp. 2070–2076 (2003)
14. Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L., Marca, D.: Accelerometer-based gesture control for a design environment. Personal Ubiquitous Comput. 10(5), 285–299 (2006), `http://dx.doi.org/10.1007/s00779-005-0033-8`, doi:10.1007/s00779-005-0033-8
15. LaViola Jr., J.J., Zeleznik, R.C.: A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. IEEE Trans. PAMI 29(11), 1917–1926 (2007), `http://dx.doi.org/10.1109/TPAMI.2007.1109`
16. Lefebvre, G., Roux, S., Petit, E.: Automatic temporal segmentation method for instrumented gesture, devices and associated terminal. Patent FR2013/51320 (2013)
17. Liu, J., Wang, Z., Zhong, L., Wickramasuriya, J., Vasudevan, V.: uwave: Accelerometer-based personalized gesture recognition and its applications. In: IEEE PerCom, pp. 1–9 (2009)
18. Mantyla, V.M., Mantyjarvi, J., Seppanen, T., Tuulari, E.: Hand gesture recognition of a mobile device user. In: IEEE ICME, vol. 1, pp. 281–284 (2000)
19. Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A., Schmidhuber, J.: A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (2006)

20. Petit, E.: Grasp: Moteur de reconnaissance de gestes. Inter Desposit Digital Number IDDNFR.001.030023.000.S.P.2010.000.31500 (2010)
21. Pylvänäinen, T.: Accelerometer Based Gesture Recognition Using Continuous HMMs. In: Marques, J.S., Pérez de la Blanca, N., Pina, P. (eds.) IbPRIA 2005. LNCS, vol. 3522, pp. 639–646. Springer, Heidelberg (2005)
22. Wilson, D.H., Wilson, A.: Gesture recognition using the xwand. Tech. Rep. CMU-RI-TR-04-57, Robotics Institute (2004)
23. Woodman, O.J.: An introduction to inertial navigation. Tech. rep., University of Cambridge (2007), `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.63.7402`
24. Wu, J., Pan, G., Zhang, D., Qi, G., Li, S.: Gesture recognition with a 3-d acceleromete. In: Zhang, D., Portmann, M., Tan, A.-H., Indulska, J. (eds.) UIC 2009. LNCS, vol. 5585, pp. 25–38. Springer, Heidelberg (2009)

Online Recognition of Fixations, Saccades, and Smooth Pursuits for Automated Analysis of Traffic Hazard Perception

Enkelejda Kasneci, Gjergji Kasneci, Thomas C. Kübler, and Wolfgang Rosenstiel

Abstract. Complex and hazardous driving situations often arise with the delayed perception of traffic objects. To automatically detect whether such objects have been perceived by the driver, there is a need for techniques that can reliably recognize whether the driver's eyes have fixated or are pursuing the hazardous object. A prerequisite for such techniques is the reliable recognition of fixations, saccades, and smooth pursuits from raw eye tracking data. This chapter addresses the challenge of analyzing the driver's visual behavior in an adaptive and online fashion to automatically distinguish between fixation clusters, saccades, and smooth pursuits.

1 Introduction

Driving is a complex task requiring proper visual functioning. According to Nagayama [Nag78], more than 50% of collisions in road traffic occur due to missed or delayed hazard perception [VRK$^+$02]. Therefore, numerous studies over the last two decades have investigated eye movements of drivers to identify deficits in visual search patterns or types of hazardous situations that may cause accidents. According to the scanpath theory by Noton and Stark [NS71], a top-down internal cognitive model of what we see drives our eyes efficiently over a scene [PS05] involving six types of eye movements: *fixations, saccades, smooth pursuits, optokinetic reflex, vestibulo-ocular reflex*, and *vergence* [LZ06]. Among these, fixations, saccades, and smooth pursuits are the most studied in driving scenarios. During a

Enkelejda Kasneci · Thomas C. Kübler · Wolfgang Rosenstiel
Department of Computer Engineering, University of Tübingen, Germany
e-mail: {enkelejda.kasneci,thomas.kuebler,
wolfgang.rosenstiel}@uni-tuebingen.de

Gjergji Kasneci
Hasso Plattner Institute, Germany
e-mail: gjergji.kasneci@hpi.uni-potsdam.de

© Springer International Publishing Switzerland 2015
P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_20

fixation the eye is kept relatively stable on an area of interest (AOI), whereas saccades correspond to rapid eye movements enabling the retinal part of sharpest vision (fovea) to fixate different areas of the scene [PS05]. Smooth pursuits occur whenever the eye follows a moving target [Duc07]. Thus, eye movements during driving result in a sequence of fixations, smooth pursuits, and saccades. Such a sequence is also known as *visual scanpath*.

The visual search strategy of a driver seems to be crucial for driving safety. Several studies have reported changes in the viewing patterns of drivers with increasing driving experience [CU98, CUR02, MS99, PHD$^+$05]. Chapman et al. [CUR02] have reported that differences between viewing patterns of novices and experienced drivers are particularly noticeable in demanding or hazardous situations. These changes, however, do not seem to be related to the cognitive resources but rather to the envisioned model of what is likely to happen during the task [UCBC02]. A further issue addressed by several studies concerns changes in viewing patterns with age. While some studies could not find any age-related decline in the subject's visual search behavior in hazardous situations [UPW$^+$05, HDBK$^+$13], others have reported a decline in search efficiency with age [MS99, HSCG01]. At least there is evidence of a significant increase of response times with age [HMM$^+$08].

Based on the above reports, several training techniques have been proposed to improve the visual scanning of drivers – with limited success [KCC12]. The main problem is that viewing patterns are highly individual and vary with the task and scene. Therefore, we hypothesize that the most effective way of hazard avoidance in driving scenarios would be to provide automated and adaptive means for continuously monitoring and analyzing the driver's visual behavior in alignment with entities on the scene; the driver should be warned well in time about upcoming hazards and, in cases where the driver's reaction time would not suffice to avoid an accident, the system (e.g., the automated breaking system) should completely take over to avoid the worst.

Indeed, today's driving assistance systems build on numerous sensors to provide assistance for specific tasks, such as automatic parking, lane keeping, intelligent speed control, emergency braking assistance, and many more. However, many of these tasks are independent of the driver's abilities, let alone her visual deficits. Moreover, the corresponding systems are geared towards reliable, deterministic performance; they do neither adapt to new traffic situations and nor to individual driving capabilities. For example, emergency braking assistance systems are relatively crude in nature, as they are typically applied as the only mean to avoid an accident. In contrast, we are interested in more fine-grained methods, which take the driver's visual search behavior into account to identify overlooked hazardous objects in real-time, possibly without distracting or patronizing the driver.

Going beyond driving assistance systems, efforts toward autonomous driving [Mar10, UAB$^+$08] have come a long way since their beginnings [Pom89]. For a safe navigation, modern self-driving cars use numerous sensors to analyze position, speed, traffic situation, road conditions, etc. The goal of related projects is to take away the burden of driving from human drivers and shift it to the inbuilt autonomous car system. However, despite the reported driving safety of such systems [Mar10], there are still many unanswered questions, especially concerning the legislation of driving, e.g., who is responsible in case of accidents, which insurance company covers the costs, etc. Our approach aims at improving the driving performance and leaves the burden of driving, and thus the whole responsibility, to the human driver. The overarching vision of our work is the synergistic combination of methods that analyze the visual search behavior of the driver as a first step towards the development of driving assistance systems that smoothly guide the driver's visual attention towards potential traffic hazards.

The most important cornerstone for the implementation of such systems is the development of methods for the analysis of the driver's visual behavior in an online fashion. This problem can be reduced to the reliable detection of and distinction between fixation clusters, saccades, and smooth pursuits from eye-tracking data. Most prior research on the detection of fixations and saccades has primarily focused on the offline detection of these types of eye movements. Various approaches such as position, velocity or acceleration based algorithms, methods based on minimum spanning trees, Hidden Markov Models or Kalman filters have been proposed [BWLH12, CNTN08, Git02, KJKG10, MSP08, PS00, SG00, SD04, TGB03, Woo02]. Yet, these approaches show two main drawbacks: (i) they either require several clustering parameters as input, which makes them inadequate for online usage or (ii) they show poor performance in the detection of fixations and saccades in dynamic scenes. A further challenge arises when smooth pursuits have to be distinguished from saccades and fixations. Although new methods have been proposed [KK13, VBG12], their applicability to the detection of smooth pursuits in dynamic scenes is unclear.

In order to identify whether a hazard was perceived by the driver, the driver's visual scanpath has to be analyzed in real time while considering all entities that appear on the visual scene. If a relatively stable target was perceived, we would expect the driver's eye movements to be focused on that target, thus yielding a cluster of fixation points. Any algorithm for clustering such fixation points needs to work in an online fashion and be unparameterized with respect to the number of clusters, as new entities may appear on the scene. Note that the system has to know the driver's AOIs at any point in time. Furthermore, as the viewing behavior differs from person to person, an adaptive algorithm is needed.

In the following, we present a work-flow for the online analysis of hazard perception based on an adaptive online algorithm for the identification of and distinction between fixations, saccades and smooth pursuits.

2 Visual Perception and Eye Movements

The human eye is a foveated system. This means that optimal visual perception (i.e., at highest spatial resolution) is only possible at a small, central area of the retina, which is known as the fovea. The spatial resolution falls by an order of magnitude within a few degrees from the fovea. Thus, when we want to look at an object, we will move our eyes until the image of the object falls on the fovea of each eye [Cor12]. Visual perception would therefore not be possible without eye movements. Although we are mostly unaware of them, when viewing a scene, our eyes are constantly moving to enable the fovea to fixate different parts of the scene. These foveal fixations are also known as *Areas of Interest*, AOIs for short. Although it is possible to deploy attention without an accompanying fixation, under natural viewing conditions this is quite rare. Thus, eye movements are assumed to be preceded by a shift in visual attention to a subsequent target. However, the whole process of visual perception involves not only the sensory input mechanisms, but also memory, attention, cognition, and decision making [HB05, Lan09, LT09].

The question, what drives our eyes over a scene, has been raised in early works of Buswell [Bus35] and Yarbus [Yar67] and, since then, has been in the scope of research in different areas. Especially in recent years, since eye-tracking devices have improved and become broadly available, the number of investigations on eye movements has increased. Several approaches modeling the process of visual attention have been proposed and follow mainly two basic paradigms: the *top-down* and *bottom-up mechanism*. The scanpath theory by Noton and Stark [NS71] suggests that an internal cognitive model of what we see not only controls our vision, but also drives the sequences of rapid eye movements and fixations efficiently over a scene [PS05], e.g., when we are asked to identify a specific object in a scene. This is called the top-down mechanism.

The bottom-up mechanism is based on the idea that an object might attract our attention due to particular features, e.g., a person wearing a red shirt among others wearing white. In this example, the red color is the salient feature. Computer-based models of visual attention have mainly focused on the modeling of bottom-up visual attention by automatically identifying possible fixation targets based on image properties [IK00, IKN98]. The detailed description of these models is beyond the scope of this work. A review of visual attention models can be found in [LT09, SBG11, THLB11]. Although intuitively comprehensible, visual salience as determined by such models can not predict saccade sequences any better than random models [HBCM07].

Despite extensive research in this area, so far, it has not been possible to predict the sequence of fixations of an observer looking at an arbitrary scene [SBG11]. The difficulty lies in the complexity of visual processing, for which eye movements are essential. Six types of eye movements are involved in visual processing of a scene: *fixations*, *saccades*, *smooth pursuits*, *optokinetic reflex*, *vestibulo-ocular reflex*, and *vergence* [LZ06]. In this work we focus on fixations, saccades, and smooth pursuits. Hence, these types of eye movements will be briefly described in the following subsections. A detailed description of the physiology and characteristics of eye movements can be found in [Duc07, LT09].

2.1 Fixation

During a fixation the eye is kept relatively stable on an AOI. It can be assumed that during a fixation, the AOI imaged on the fovea is being visually attended to by the viewer. Fixations usually have a duration of about $200-300ms$, although much longer fixations are possible. The duration of fixations varies depending on the visual search task. Furthermore, fixation durations show inter- and intra-subject variability. Therefore, the duration of fixations is an important research topic in itself, as it relates to the visual information to which the observer is attending as well as to her cognitive state [LT09].

The general assumption while driving is that the visual perception of scene features, such as signs, pedestrians, and obstacles, requires foveated vision (i.e., explicit fixation of the object of interest) [FZ09]. Although peripheral vision is considered as sufficient for some subtasks, such as keeping the vehicle centered in the lane [SNP96], in [MS04] was reported that peripheral vision is insufficient for the detection of traffic hazards. This means, that to have seen a traffic hazard, the driver must have fixated it.

2.2 Saccade

A saccade represents a rapid eye movement to reposition the retinal part of sharpest vision (fovea) to fixate different areas of a scene [PS05]. A saccade occurs at maximum frequency of about $4s^{-1}$ (e.g., during reading) and maximum velocity of approximately $500°/s$ [LT09]. Saccades are performed by both eyes conjugately. As introduced above, a saccade can be triggered in a bottom-up manner, e.g., by a suddenly moving target or a new stimulus, or in a top-down way. Saccade duration varies between $10ms$ and $100ms$ [Duc07]. The reaction time for externally driven saccades is usually $150-200ms$. Furthermore, saccade latency increases with increasing eccentricity [LT09].

2.3 Smooth Pursuit

A smooth pursuit occurs whenever the eye follows a moving target [Duc07]. Similarly to saccades, smooth pursuits are conjugate eye movements that are performed at lower velocities of approximately $15°/s$ [LT09]. They are elicited by moving targets and cannot be executed voluntarily without a target that is visually pursued.

3 Online Recognition of Fixations, Saccades, and Smooth Pursuits from Eye-Tracking Data

The reliable detection of and distinction between the above types of eye movements in an online fashion is a crucial step towards the automated identification of objects that might be overlooked by the driver, i.e., potential traffic hazards. While performing such a detection on eye-tracking data in alignment with information from the visual scene is reliably feasible by us humans, reliable automated clustering of eye movements is still challenging; even more so, when the visual scene changes continuously, such as in driving scenarios.

In the scope of this section will be an online, adaptive algorithm for the reliable detection of the above eye movement types. After discussing related work on the classification of eye movement data, we present a two-step approach to the online detection of the type of eye movements from eye-tracking data.

3.1 State of the Art Methods

Since the first works on classification of eye movements [McC81, Wid84], several approaches have been proposed that fall into three main categories: (i) dispersion-based methods, (ii) velocity- and acceleration-based methods, and (iii) probabilistic methods based on statistical Markov models. All groups of algorithms will be briefly discussed in the following subsections. A detailed review can be found in [HNA$^+$11].

3.1.1 Dispersion-Based Methods

Dispersion-based algorithms distinguish between different types of eye movements (mostly between fixations and saccades) based on the distance between subsequent eye position points. Usually, dispersion-based algorithms aim at detecting fixation clusters by identifying data points that are close to each other within a predefined time window, whereas all other data points are not considered [HNA$^+$11]. A method that is representative of this group, is the *Dispersion Threshold Identification (I-DT)* algorithm [SG00], where the separation of fixation clusters from saccades is based on the dispersion of consecutive data points within a temporal window. In I-DT, the dispersion D is defined as the sum of the maximum and minimum differences between the x and y coordinates of the points within the temporal window, i.e., $D = [max(x) - min(x)] + [max(y) - min(y)]$. If D is below a predefined threshold, the data points within the window belong to a fixation. Otherwise, a saccade is found. Similar approaches that use a temporal threshold as in I-DT, but differ in the way the dispersion is calculated have also been presented in [Bli09, HNA$^+$11, SG00, SSC08].

Another prominent algorithm from this group is the *Identification of eye movements based on Minimum Spanning Trees (I-MST)* [HNA$^+$11, SG00]. As presented in [SG00], this algorithm first builds a minimum spanning tree (MST) on a predefined number of eye position points that fall within a temporal window. Then, from each node v of the MST, all other MST nodes are visited in depth-first search up to a predefined depth threshold. The mean and the variance of the length of the edges that were visited during the depth-first search are assigned as meta information to v. Based on the meta information of the end nodes of an edge, a decision can be taken concerning the type of eye movement it represents, i.e., a saccade or fixation [SG00].

Other dispersion-based approaches are based on clustering algorithms. For example, in [SD04] a mean shift clustering algorithm for the detection of fixation clusters was used. In another approach [ULIS07], fixation clusters have been detected based on projection clustering. Typical clustering algorithms, such as the well-known and widely used k-means algorithm, are only applicable to the problem of identifying fixations clusters, when the number of fixation clusters is known a priori. Hence, such algorithms cannot be applied to the analysis of eye movements when viewing dynamic scenes, e.g., such as those occurring while driving, where the number of resulting clusters is not known.

Other approaches in this realm have divided the viewing area into a regular grid and recorded the time spent inside each square [SG00]. While such approaches are well-suited for static scenes, e.g., reading a page, they are not applicable to dynamic scenarios, where no a priori information about the viewing area is given. The same holds for techniques that provide visualizations of the areas of interest, i.e., fixation clusters, by adapting the original image [Woo02].

Over the last years, dispersion-based approaches have been implemented in both academic and commercial tools [HNA$^+$11], e.g., faceLab [See], SMI BeGaze [Sen], Gazetracker [Eye], etc. Although they offer several useful features, their main drawback is that they come as black-box solutions and can hardly be integrated in self-designed applications. Moreover, in most cases, commercial analysis software provides only offline analysis of eye movements. Academic tools such as the MATLAB-toolbox GazeAlyze [BWLH12] based on ILab [Git02], or Astef [CNTN08] can be easily integrated in self-designed applications but, unfortunately, only for the offline analysis. Another major drawback of dispersion-based methods is that they often rely on thresholds (e.g., length of the temporal window, dispersion threshold, etc.) that have to be empirically adjusted to the individual viewing behavior, the viewing area, and the specific task, thus being inadequate for the task of adaptive, online scanpath analysis [HNA$^+$11].

3.1.2 Velocity- and Acceleration-Based Methods

Velocity- and acceleration-based algorithms distinguish between different types of eye movements based on velocities or accelerations between subsequent eye data points [HNA$^+$11]. An approach that is representative of this group is the *Velocity-Threshold Identification (I-VT)* algorithm. In I-VT, a point is identified as a saccade point, if the implicit velocity along the distance from the previous data point to that point exceeds a predefined threshold (e.g., $>$ 300 deg/sec). Otherwise the data point is assigned to a fixation cluster [SG00]. Approaches based on acceleration thresholds work similarly. Due to their simplicity, algorithms of this group have been implemented in several commercial software packages, e.g., Tobii [Tob], SMI [Sen], EyeLink [SR]. Several recommendations for task specific settings of velocity thresholds have also been made [HNA$^+$11].

In general, this group of algorithms is best suited for the analysis of eye data tracked at high sampling frequency ($>$ 200 Hz) [HNA$^+$11]. However, as with the dispersion-based methods, a major drawback of the I-VT algorithm and methods related to it is that the applied threshold values need to be empirically adjusted to the eye data at hand. For this reason and because of the fact that velocity profiles are strongly physically- and physiologically- dependent, such methods are not reliable, especially when real-time analysis of eye-tracking data is needed. An additional issue concerns the detection of smooth pursuits. Most velocity-based methods have primarily focused on the separation of fixation clusters from saccades without considering smooth pursuits as an additional class. In [Itt05] a velocity-based approach, which classifies fixation and smooth pursuit clusters into a general "intake" category has been presented. Although saccades and fixations could be separated based on threshold values, smooth pursuits could not be distinguished from fixation clusters. Commercial implementations, such as the Tobii Fixation Algorithms [Tob] and the EyeLink parser [SR], work similarly [HNA+11]. Two-step approaches that use two velocity thresholds to first detect saccades and then separate smooth pursuits from fixations, were presented in [Fer00] and [KK13]. Offline Eigenvector analysis on data points and velocities within a temporal window has been proposed as an approach to distinguish fixations from smooth pursuits in [BBM+09]. Although reliable, the static nature of such offline methods makes them inadequate for the application to dynamically changing scenes. In [KK13], a combination of velocity and dispersion thresholds was used with the I-VT algorithm (coined I-VDT in [KK13]) to classify saccades, smooth pursuits, and fixations. As above, following a two-step approach, first saccades are separated from the other two types of eye movements based on a velocity threshold; then, a dispersion threshold is used to separate fixations from smooth pursuits [KK13]. As with all the presented threshold-based methods, the thresholds of I-VDT have to be chosen carefully, based on thorough data analysis and the specific task.

In summary, most of the dispersion- and velocity-based approaches are based on a considerable number of input parameters that can have significant influence on the classification result. Although several recommendations regarding thresholds for specific tasks have been made, they mostly consider eye-tracking data from viewing behavior on static images.

3.1.3 Probabilistic Methods

The most prominent methods that are representative of this realm are Hidden Markov Models (HMM). Such models are simple dynamic Bayesian networks with variables representing values from a discrete state and observation space. Because of their sequential nature, such models are a popular choice for the analysis of successively arising data points (i.e., observations). For the detection of fixations and saccades from eye data, HMMs have been used with velocity observations between successive data points, thus allowing the adaptation of the model to the physiological viewing behavior [SG00]. In the model of [SG00] (coined I-HMM), the two states used represent velocity distributions over fixations and saccades. Transition probabilities between the states represent the probability of the next sample belonging to a fixation cluster or a saccade, given the current state [HNA+11]. Due to the probabilistic representation of velocities (i.e., no thresholds are needed), the I-HMM is reported to perform better than fixed-threshold methods, such as I-VT [SG00]. The dynamic and probabilistic nature of HMMs makes them an adequate choice for sequential data arising in an online fashion and containing variability in its features.

Another similar probabilistic approach was presented in [KJKG10, SMDRV91] and was based on a Kalman-Filter that models and predicts the velocity of eye movements based on previous eye data points. The proposed model could distinguish saccades from fixations.

In summary, with respect to their application to the online analysis of eye data generated in the context of driving, many of the related approaches reviewed in this section suffer from one or more of the following problems: (i) they require several static input parameters, which makes them inadequate for online usage, (ii) they do not adapt to changing scene information or to the subject's viewing behavior, or (iii) they do not successfully generalize to the detection of smooth pursuits in addition to fixations and saccades [KK13].

In contrast, our method, which can be assigned to the family of probabilistic models, performs reliably on the task of online classification of fixations, saccades, and smooth pursuits.

3.2 Method Overview

The method for classifying eye-tracking data consists of two steps: In the first step, a Bayesian online mixture model is employed to distinguish saccades from data points that might represent fixations and smooth pursuits. In a second step, a fine-grained Principal Component Analysis (PCA) is conducted to distinguish fixations from smooth pursuits. Both steps are presented in the following subsections.

3.2.1 Bayesian Mixture Model for the Online Recognition of Fixations and Saccades

In [TKRB12], we presented an effective online clustering algorithm, that could distinguish between fixations and saccades by considering only the Euclidean distance between subsequent data points recorded by the eye tracker. The underlying model was based on the intuition that distances between subsequent fixation points will in general be shorter than distances between subsequent saccade points; that is, distances between subsequent fixation points would be generated from a specific Gaussian distribution and those between subsequent saccade points from another. Imagine a temporally ordered sequence of two-dimensional data points, $\mathbf{S} = \{s_i \mid 1 \leq i \leq T\}$, e.g., recorded by an eye tracker and representing the visual scanpath of an observer over time. In such a representation, a dense region of successive points (i.e., successive points that are close to each-other in terms of the Euclidean distance) might reflect an AOI (or, more specifically, an object that attracts the observer's attention). According to [NH10], a fixation location is manifested in eye-tracking recordings as a cloud of points that are normally distributed around the center of the object of interest. Assuming such a normal distribution of fixation points, one could leverage the covariances derived from the coordinates of the data points to approximate the Gaussian distribution that governs them. However, when the number of observation points is rather moderate, such an approximation typically leads to a poor estimation of the underlying distribution. Hence, the proposed algorithm makes use of the intuition that the distances between successive fixation points in an AOI and the distances between successive saccade points come from two distinct Gaussian distributions; thus, following the idea that the structure of two-dimensional data points can be implicitly understood by looking at the distances between them. The resulting (reduced) dimensionality allows the algorithm to effectively deal with a moderate number of observations. The parameters of the two Gaussian distributions (i.e., means and variances) that govern the two different types of distances are learned through a generative mixture model. The Bayesian network in Figure 1 depicts the mixture model for the two Gaussian distributions.

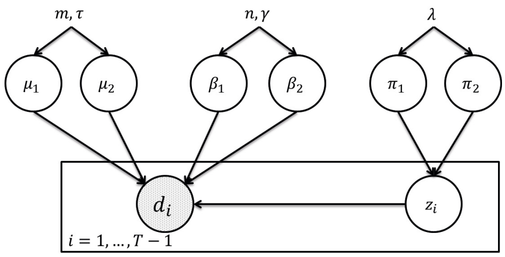


Fig. 1 Bayesian mixture model for the online detection of fixation clusters and saccades

Let $\mathbf{D} = \{d_i \mid 1 \leq i \leq T-1\}$ be the set of distance variables between points $s_i, s_{i+1} \in \mathbf{S}$. $\boldsymbol{\Theta} = \{\mu_1, \beta_1, \mu_2, \beta_2\, \pi_1, \pi_2\}$ denotes the complete parameter set of the mixture model that is depicted in Figure 1. The mixture component is denoted by the variable z_i and the observed distances by the observed variables d_i. The simplifying assumption here is that the distances are generated sequentially in an independent and identically distributed fashion. More specifically, each distance between two successive points is generated independently by the most likely Gaussian distribution.

The joint probability distribution of the model is given by:

$$\begin{aligned} p(\mathbf{D}, \mathbf{z} | \boldsymbol{\Theta}) &= \prod_{i=1}^{T-1} p(z_i | \boldsymbol{\pi}) p(d_i | \mu_{z_i}, \beta_{z_i}) \\ &= \prod_{i=1}^{T-1} \pi_{z_i} N(d_i; \mu_{z_i}, \beta_{z_i}) \end{aligned}$$

where $\mathbf{z} = \{z_1, \ldots, z_{T-1}\}$, with $z_i \in \{1, 2\}$ being the index of the mixture component chosen for distance d_i, and $\boldsymbol{\pi} = \{\pi_1, \pi_2\}$ denotes the set of mixture parameters.

We have used Infer.NET [MWGK12] to specify the model with the following distributions.

(1) The factorized distribution over the probabilities of each mixture component:

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{i-1}^{T-1} \pi_{z_i}$$

(2) The factorized prior distribution over the model parameters:

$$p(\boldsymbol{\Theta}) = p(\boldsymbol{\pi})p(\boldsymbol{\mu})p(\boldsymbol{\beta})$$

Furthermore, for the online version of the model, the following definitions are needed.

(3) The prior distribution generating the π_1 and π_2 (i.e., the mixture parts) is defined as a symmetric Dirichlet distribution:

$$p(\boldsymbol{\pi}) = Dir(\boldsymbol{\pi}; \lambda) \tag{1}$$

Note that the Dirichlet distribution is the so-called conjugate prior of the Multinomial distribution governing π_1 and π_2. This means that the posterior distribution on $\boldsymbol{\pi}$ has a similar mathematical form as the Dirichlet distribution, allowing the Dirichlet prior to be updated by the posterior distribution as more observations are made [Bis06]. Similar reasoning holds for the following definitions.

(4) The prior distribution over the means as a product of Gaussians:

$$p(\boldsymbol{\mu}) = N(\mu_1; m, \tau)N(\mu_2; m, \tau) \tag{2}$$

(5) The prior distribution over the precisions as a product of Gammas:

$$p(\boldsymbol{\beta}) = Gam(\beta_1; n, \gamma)Gam(\beta_2; n, \gamma) \tag{3}$$

Figure 2 represents the Gaussian distributions for fixation clusters and sccades, learned by the above model on a real-world data set.

The above Bayesian model comes with the great benefit that it can be easily turned into an online learning model. In general, for given model parameters $\boldsymbol{\Theta}$ and observations $\mathbf{D}$, after applying Bayes' rule follows that the probability of the parameters $\boldsymbol{\Theta}$ in light of the data $\mathbf{D}$ is:

$$p(\boldsymbol{\Theta}|\mathbf{D}) = \frac{p(\mathbf{D}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})}{p(\mathbf{D})}$$

More generally, we can write:

$$p(\boldsymbol{\Theta}|\mathbf{D}) \propto p(\mathbf{D}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})$$

The above formula suggests that in an online setting the prior of the parameters $p(\boldsymbol{\Theta})$ can be iteratively substituted with the posterior $p(\boldsymbol{\Theta}|\mathbf{D})$, while the likelihood on the parameters $p(\mathbf{D}|\boldsymbol{\Theta})$ helps readjust the model, as more and more observations are made (see also [Bis06]).

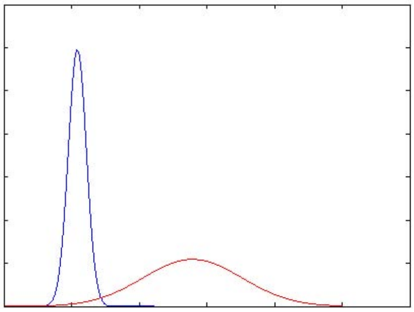


Fig. 2 Two Gaussian distributions learned by the Bayesian mixture model. The left distribution represents distances between successive eye-tracking points that belong to fixation clusters, and the right distribution reflects distances between successive saccadic data points [Kas13].

The above formulation allows the model to readjust the learned parameters $\boldsymbol{\mu},\boldsymbol{\beta},\boldsymbol{\pi}$ as more and more eye movement data is available. To this end, we use Gaussian distributions as conjugate priors for the means μ_1,μ_2 (see Equation 2), Gamma distributions as conjugate priors for the precisions β_1,β_2 (see Equation 3), and Dirichlet distributions as conjugate priors for π_1,π_2 (see Equation 1). All these distributions belong to the so-called exponential family of distributions, meaning that they have the same abstract mathematical form. This allows the above prior distributions to be iteratively updated by the posterior distributions on the corresponding variables of the model as new data points are observed.

The whole model was implemented in C# and Infer.NET. The latter not only allows the declarative definition of models such as the above, it also provides various methods for approximate inference; we have used Variational Message Passing as implemented by Infer.NET. In the following we show the fragment of the C# code that is responsible for the update of the priors of the parameters with their posteriors. Initially, we let the inference engine infer all parameters of the model and their posteriors based on a small number (numTrainingPoints) of training points (see lines 1-4). For every other subsequent data point, the priors of the parameters are set to their inferred posterior values (see while loop). The subsequent data point is clustered based on these updated parameters.

```
    ...
 1. engine.InferAll(weights, means, precs);
 2. var meanPost = engine.Infer<Gaussian[]>(means);
 3. var precPost = engine.Infer<Gamma[]>(precs);
 4. var weightPost = engine.Infer<Dirichlet>(weights);
 5. int idxDataPoints = numTrainingPoints+1;
 6. while (iterDataPoints.MoveNext()){
 7.    var newObsData = new double[]{iterDataPoints.Current};
 8.    numData.ObservedValue = newObsData.Length;
 9.    data.ObservedValue = newObsData;
10.    meanPrior.ObservedValue = meanPost;
```

```
    precPrior.ObservedValue = precPost;
    weightPriors.ObservedValue = weightPost;
    ...
    idxDataPoints++;}
```

3.2.2 Principal Component Analysis for the Detection of Smooth Pursuits

Reliably distinguishing fixations from saccades is an important first step towards automated driving support and the prediction of hazardous situations. However, in driving scenarios, objects are typically in motion relative to the driver. Therefore it is crucial to automatically recognize objects that are being pursued by the driver's gaze and others that are not. To address this issue, in [TKK$^+$13] we have extended the above method to also recognize smooth pursuits. We first describe the general idea and then the details of the algorithm.

Let us assume that the last k gaze points were labeled by the above mixture model as fixation points. The key question is whether these points are centered around a relatively stable target or correspond to a moving object that is being pursued by the driver's gaze. In the former case, the vector that represents the highest variability in the k data points and the one representing the second highest variability will have approximately similar lengths. In the latter case though, there will be a notable difference in the lengths of the two vectors. The direction of highest variability corresponds the direction of movement of the followed object relative to the observer. Note that these vectors correspond to the first and the second eigenvectors of the covariance matrix of the data points. We rely on Principal Component Analysis [Jol86] to efficiently retrieve these vectors.

More specifically, let $\mathbf{M}$ be the matrix holding the last k successive data points that were all labeled as fixation points, such that each row of $\mathbf{M}$ contains the coordinates of a point after subtracting the (empirical) mean of the distribution of all points. Through singular value decomposition, $\mathbf{M}$ can be decomposed into $\mathbf{U\Sigma V}^T$, where $\mathbf{U}$ contains the orthonormal eigenvectors of the covariance matrix $\mathbf{MM}^T$, $\mathbf{\Sigma}$ is a diagonal matrix containing the positive roots of the eigenvalues of $\mathbf{MM}^T$, and $\mathbf{V}$ contains the orthonormal eigenvectors of $\mathbf{M}^T\mathbf{M}$. This decomposition is unique up to different orderings of the values in $\mathbf{\Sigma}$. This means that if the values are ordered decreasingly (with the largest value in the upper-left corner of the matrix) in $\mathbf{\Sigma}$, then the first and the second column of $\mathbf{U}$ correspond to the first and second eigenvector of $\mathbf{MM}^T$, respectively.

In order to decide whether the last k data points describe a smooth pursuit, we compute:

$$\frac{\sigma_2^2 \cdot \|u_2\|}{\sigma_1^2 \cdot \|u_1\|} = \frac{\sigma_2^2}{\sigma_1^2} \leq t$$

where t is an empirically established threshold, σ_1 and σ_2 are the largest and the second largest values in $\mathbf{\Sigma}$, respectively, and u_1 and u_2 are the corresponding eigenvectors.

An example scenario of an online smooth pursuit detection is depicted in Figure 3, where a hazardous situation arises from the white car cutting into the lane from the right. The figure shows video frames from a driving scene that was generated in a driving simulator. The driver's eye movements were tracked by means of a mobile Dikablis eye tracker at a sampling rate of 25Hz.

Figure 3(a) shows the moment when the driver's attention is caught for the first time by the white car. The black arrow shows the shift of visual attention in the most recent time frame of 1s. Figure 3(b) depicts the situation 400 ms later, when the driver has come closer to the white car. During this time, the driver's gaze has pursued the relative movement of the white car. According to the spread of the gaze points, which were labeled as a fixation cluster by the online Bayesian mixture model, the PCA-based analysis has classified them as belonging to a smooth pursuit.

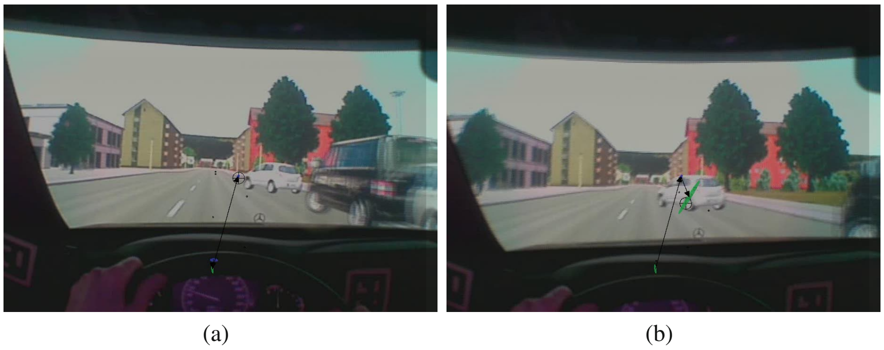


Fig. 3 An example scenario of a smooth pursuit [TKK$^+$13]

4 Experimental Evaluation

In this section, we first demonstrate the superior quality of the online Bayesian mixture model over a state-of-the-art model for detecting fixations and saccades and then showcase the quality of the PCA-based detection of fixations, saccades, and smooth pursuits based on a real-world, hand-labeled data set.

4.1 Evaluation of the Bayesian Mixture Model in Comparison with a Hidden Markov Model

For the quality evaluation of the Bayesian Mixture Model (BMM), in [KKKR14] we compared its prediction performance to that of a Hidden Markov Model (HMM), such as the one presented in [KJKG10, SG00]. These types of models come with several advantages over threshold-based methods: (1) no fixed thresholds are needed, instead the parameters of the model (i.e., state transition probabilities and label emission probabilities) are learned from labeled data, (2) in consequence, HMMs can adapt to the individual (i.e., physiological) viewing behavior of a subject and to the specific task, and (3) given their dynamic sequential structure, they are naturally suited for sequentially arising data points, such as eye-tracking data. Note also that both models, HMM and BMM, belong to the family of probabilistic models.

We implemented a two-state HMM according to the description of the I-HMM in [SG00]. However, the observed sequences for the I-HMM were velocities between the eye-tracking data points, whereas in the HMM version that we have implemented, the sequences consist of distances between successive data points [KKKR14]. Based on training data (i.e., manually labeled data points) such distance observations can be mapped to a discrete set of observations, which in our context correspond to the IDs of two estimated Gaussian distributions, i.e., one representing the distribution of distances between saccades and another one standing for the distribution of distances between fixations. These distributions, the emission probabilities of their IDs, as well as the transition probabilities between the HMM states are learned from labeled data, by computing the corresponding maximum likelihood estimations. A Viterbi-based, forward-backward algorithm [FJ73] was implemented to compute the most probable state sequence of the HMM corresponding to an observation sequence.

To evaluate the performance of the HMM and BMM, we employed nine real-world, eye-tracking data sets from nine different subjects who took part in our driving experiments [KSA$^+$14]. Each data set consisted of 750 data points recorded while driving, at a sampling rate of 25Hz by a mobile Dikablis eye tracker. The data was analyzed frame-wise by two PhD students. An eye-tracking data point was thereby labeled as belonging to a saccade or fixation only if both of the judges agreed. Blinks and disagreements were excluded from the data. Each of the data sets corresponds to a driving sequence of 30 seconds, resulting in a total of 4.5 minutes.

Both models were applied post-experimentally, but in real-time on sequentially arising raw eye-tracking data points from nine different data sets. Both the HMM and the BMM were trained on the first 300 eye-tracking points of the each data sets. From such training data, the HMM derives the distance distributions (i.e., for distances between consecutive saccades and consecutive fixations) as well as transition and emission probabilities. In contrast, the BMM, updates the learned parameters in an online fashion as new data points are observed. Once the parameters were learned for both models, their prediction quality was tested on each data set.

Table 1 shows a summary of the evaluation of both algorithms with respect to the detection of saccades and fixations from [KKKR14]. The quality results present average values from the evaluation of the models on the above nine data sets in terms of the following measures: *Precision* ($\frac{TP}{TP+FP}$), *Recall* ($\frac{TP}{TP+FN}$), *F1-measure* ($\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$), which represents the harmonic mean of precision and recall, and *Miss-Classification-Rate (MCR)* ($\frac{FP+FN}{TP+FP+TN+FN}$).

Table 1 Quality comparison of the Hidden Markov Model (HMM) and the Bayesian Mixture Model (BMM) with respect to the detection of fixations and saccades from raw eye-tracking data

	Data points	Model	Precision	Recall	F1	MCR
Saccade	**910**	**HMM**	0.774	0.803	0.763	0.068
		BMM	0.997	0.955	0.975	0.009
Fixation	**5840**	**HMM**	0.974	0.943	0.958	0.068
		BMM	0.989	1.000	0.994	0.009

As shown in Table 1, the BMM clearly outperformed the HMM, especially in the detection of saccades where an astounding average precision of 99.7% is achieved. Note that the reliable recognition of saccade points is in the context of driving very critical, since a correct detection of saccades implies a correct separation of fixation clusters. In general, the proportion of saccade points is much smaller than that of fixation points. For a model such as the HMM, which aims at maximizing the joint probability of a sequence of states and corresponding observations, it is safer to focus on the most probable states and observations; these are fixations and the corresponding distance distributions. These findings are also in line with the findings presented in [KJKG10].

A much better performance of the HMM is shown in the detection of fixation points, with an average precision of 97.5%. The precision achieved by the BMM is even higher, i.e., 98.9%. The recall of the HMM with respect to the detection of fixation points is remarkably lower than that of the BMM, because the HMM does not manage to adapt well enough to varying distances between fixation points. This is different for the BMM, which can adapt to varying distances in an online fashion. Considering the MCR, which is very low for both models, we found a superior performance of the BMM with values smaller than 1%. In summary, the HMM was outperformed by the BMM with respect to all measures.

These results highlight the superior performance of the online Bayesian mixture model in comparison to an HMM. Especially, on the difficult task of reliably detecting saccades, which is crucial for the correct separation of fixation clusters, the Bayesian model achieves highly satisfiable precision, and recall values.

Beyond the context of driving, the BMM can be integrated into vision research tools (e.g., Vishnoo [TKP$^+$11]) to analyze the viewing behavior during visual search tasks presented on a screen. Furthermore, the BMM can be used in the context of medical testing, e.g., for advanced visual field testing involving online analysis of fixations (e.g., EFOV [THH$^+$13]). In summary, whenever fixations and saccades have to be detected in an online fashion, the BMM is a highly reliable choice.

4.2 Overall Evaluation of the Classification Technique

The PCA-based extension of the Bayesian mixture model was evaluated on data from a driving simulator experiment with 27 subjects. The results were presented in [TKK$^+$13].

The driving experiment was conducted in the moving-base driving simulator [Zee10] shown in Figure 4 at the Mercedes-Benz Technology Center in Sindelfingen, Germany. The facility allows simulating acceleration forces in all directions, with up to 1g into the direction of a twelve-meter long rail. The cabin, Figure 4(b), contained a real car body (Mercedes S class with automatic transmission) amidst a 360° projected virtual reality, Figure 4(a). The car body was oriented perpendicular to the rail. Thus curve and lane changing maneuvers appear most realistic, while acceleration and braking result in a movement often described as “diving”. All in all, acceleration, sound effects, and car environment contributed to a near-to-reality driving experience.

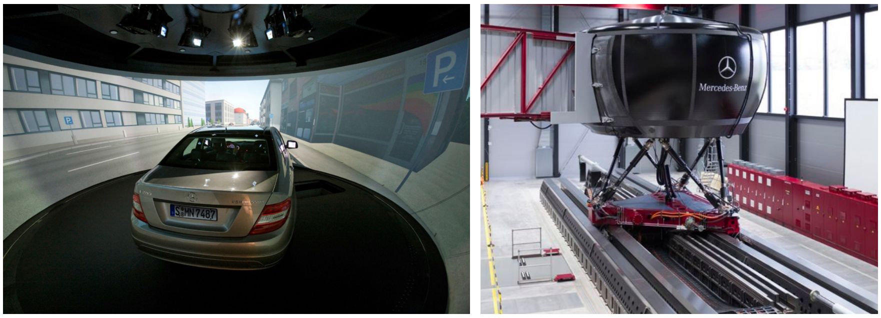


Fig. 4 Moving base driving simulator. (a) View from inside the cabin onto the environment projection. (b) The entire cabin is mounted on a hexapod, moving along the 12m rail resulting in up to 1g acceleration force. Figures were provided by Daimler AG.

Nine hazardous situations were placed along the 37.5 km long driving route. These traffic hazards, e.g., pedestrians suddenly appearing behind parking cars and trying to cross the road (Figure 5) and risky overtaking maneuvers, were the same for each driver. In case of an overseen hazard, the participants did not experience a crash. For example, it was not possible to run over pedestrians. Instead pedestrians would leap backwards and overtakers would return to their original lane to avoid crashes and subsequent psychological stress. The course contained rural as well as urban areas with different speed limits up to 100 km/h. For the 27 study subjects this would result in a total of 243 hazardous situations. However, some of the study participants aborted the session due to motion sickness or technical problems, resulting in a total of 184 hazardous situations.

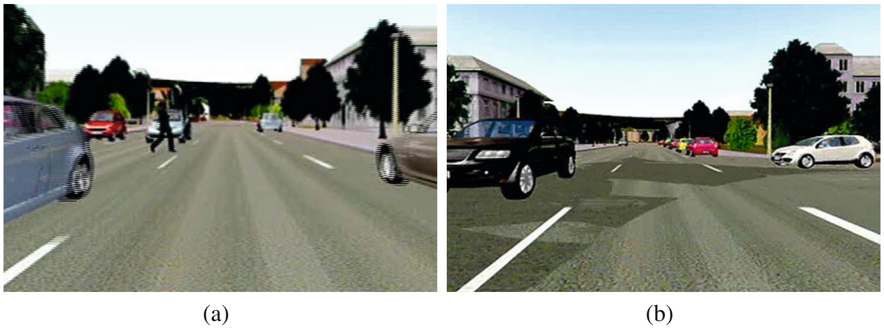


Fig. 5 Scenes from the virtual reality [TKK$^+$13]: (a) a pedestrian intending to cross the road, (b) and a white car coming from the right side.

Driver's eye movements were recorded at 25Hz using a Dikablis mobile eye tracker, whereas head movements were recorded by a LaserBird head tracker. Prior to the driving session each subject underwent a brief training session of 5 km length, in order to adjust to the car and the driving environment. This session was also used to learn an initial visual behavior model with parameters adjusted to the current driver. The training session began with a straight road and became more complex as oncoming traffic became successively denser and traffic signs more frequent.

We evaluated our method on 184 hazardous situations. The spatial extent of the traffic hazards was manually annotated using bounding boxes, separately for each video frame. The analysis of the driver's viewing behavior and the detection of fixations, smooth pursuits and saccades was performed online using the algorithms presented in the previous section. A traffic hazard was considered as perceived, if a fixation or a smooth pursuit cluster intersected with the bounding box around it at any frame. An example of an intersection between the driver's gaze and the bounding box is shown in Figure 6. The results are presented in Table 2.

In 169 of the 171 situations where the hazard was considered as perceived (i.e., an intersection between the fixation cluster and the bounding box occurred) the driver reacted by performing a braking or obstacle avoidance maneuver. In 7 of the 13 hazardous situations where the bounding box was not intersected by a fixation or smooth pursuit cluster no reaction to the situation happened (i.e., the target was missed). Note that in a real-world scenario, these situations would have led to accidents. In 6 situations the bounding box was not intersected but the driver reacted nevertheless. These hazards were perceived, even though the driver did not explicitly look at them. In 2 situations, where the bounding box was intersected by a fixation or smooth pursuit cluster, the driver did not react. Although the driver looked at the targets, they were not perceived. Again, in reality, such situations would result in accidents. In terms of predicting the recognition of traffic hazards by the driver (using only raw eye-tracking data), the algorithm showed an overall accuracy of 95,7%, a specificity of 96,5%, and a sensitivity of 77,8%. These results highlight the overwhelming reliability of the proposed method.

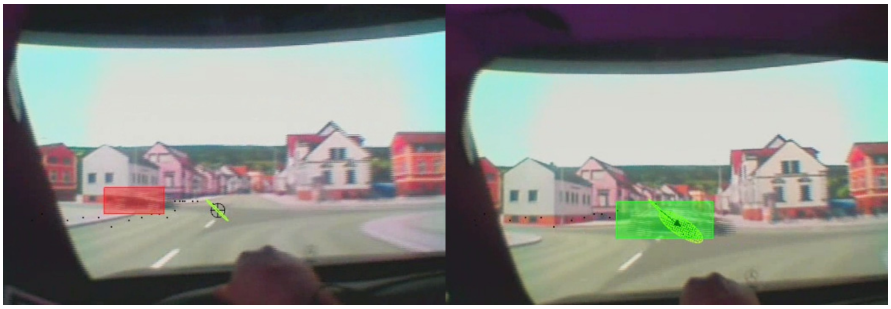


Fig. 6 A car appears on the left of the driving scene and is about to cut the driver's way. The red bounding box around the approaching hazard and the driver's fixation cluster are shown. Once the bounding box is intersected by the fixation cluster, the traffic hazard is marked as "perceived", highlighted by the green bounding box [TKK+13].

In order to look into the detailed per-class performance of our algorithm (i.e., the detection of fixations, saccades, and smooth pursuits), we randomly picked the eye tracking data of one of the subjects. For a six-minute-long driving sequence, two PhD students manually annotated the data points as fixations, saccades, or smooth pursuits. Note that this annotation task is very laborious, as the data has to be labeled frame-wise. Overall, there were 46 eye movement events that were labeled; 27 as fixations, 8 as smooth pursuits, and 11 as saccades. Table 2 shows the per-class true-positive and false-positive counts.

Table 2 True and false positive counts for the detection of fixations, smooth pursuits, and saccades

Eye movement type	Annotation	True Positives	False Positives
Fixation clusters	27	26	1
Smooth pursuit clusters	8	7	2
Saccades (single points)	11	11	0

Seven out of eight smooth pursuits and 26 out of 27 fixations were identified correctly. While our algorithm detected all saccades correctly, two fixations were falsely classified as smooth pursuits and one smooth pursuit as fixation. Although preliminary in nature, these results are very promising and we plan to further evaluate the algorithm on larger labeled datasets.

5 Conclusion

We presented an online adaptive, classification algorithm for detecting fixations, saccades, and smooth pursuits in driving scenarios. This algorithm was primarily evaluated with respect to its ability to detect hazardous traffic situations that might have been overlooked by the driver. In a user study with a state-of-the-art driving simulator, the method showed an impressive detection accuracy, which we think can be mainly explained by the method's ability to adjust the underlying model to the driver's visual behavior. An evaluation on the per-class detection of fixations, saccades, and smooth pursuits hints at the method's ability to recognize and distinguish between different types of eye movements. Apart from experiments on larger, labeled datasets, we also plan to investigate the integration of physiological models, which take heart rate and skin conductance into account to predict the driver's stress levels [KKR+14]. Such models could supplement models that are based on gaze recordings to predict traffic hazard perception and the driver's ability to react.

Acknowledgements. The authors would like to thank Daimler AG and the driving simulator experts for the technical support during the whole experimental study.

References

[BBM+09] Berg, D.J., Boehnke, S.E., Marino, R.A., Munoz, D.P., Itti, L.: Free viewing of dynamic stimuli by humans and monkeys. Journal of Vision 9(5), 1–15 (2009)

[Bis06] Bishop, C.M.: Machine Learning and Pattern Recognition. Springer-Verlag, New York, Inc., Secaucus (2006)

[Bli09] Blignaut, P.: Fixation identification: The optimum threshold for a dispersion algorithm. Attention, Perception, & Psychophysics 71(4), 881–895 (2009)

[Bus35] Buswell, G.T.: How people look at pictures. University of Chicago Press, Chicago (1935)

[BWLH12] Berger, C., Winkels, M., Lischke, A., Höppner, J.: GazeAlyze: A MATLAB toolbox for the analysis of eye movement data. Behavior Research Methods 44(2), 404–419 (2012)

[CNTN08] Camilli, M., Nacchia, R., Terenzi, M., Di Nocera, F.: Astef: A simple tool for examining fixations. Behavior Research Methods 40, 373–382 (2008)

[Cor12] Cornsweet, T.: Visual perception. Academic Press (2012)

[CU98] Chapman, P.R., Underwood, G.: Visual search of driving situations: Danger and experience. Perception London 27, 951–964 (1998)

[CUR02] Chapman, P., Underwood, G., Roberts, K.: Visual search patterns in trained and untrained novice drivers. Transportation Research Part F: Traffic Psychology and Behaviour 5(2), 157–167 (2002)

[Duc07] Duchowski, A.: Eye tracking methodology: Theory and practice. Springer, London (2007)

[Eye] Eyetellect. GazeTracker, `http://www.eyetellect.com/gazetracker/`

[Fer00] Ferrera, V.P.: Task-dependent modulation of the sensorimotor transformation for smooth pursuit eye movements. Journal of Neurophysiology 84(6), 2725–2738 (2000)

[FJ73] Forney Jr., G.D.: The viterbi algorithm. Proceedings of the IEEE 61(3), 268–278 (1973)

[FZ09] Fletcher, L., Zelinsky, A.: Driver inattention detection based on eye gaze-road event correlation. The International Journal of Robotics Research 28(6), 774–801 (2009)

[Git02] Gitelman, D.R.: ILAB: A program for postexperimental eye movement analysis. Behavioral Research Methods, Instruments and Computers 34(4), 605–612 (2002)

[HB05] Hayhoe, M., Ballard, D.: Eye movements in natural behavior. Trends in Cognitive Science 9(4), 188–194 (2005)

[HBCM07] Henderson, J.M., Brockmole, J.R., Castelhano, M.S., Mack, M.: Visual saliency does not account for eye movements during visual search in real-world scenes. In: Eye movements: A Window on Mind and Brain, pp. 537–562 (2007)

[HDBK$^+$13] Hamel, J., De Beukelear, S., Kraft, A., Ohl, S., Audebert, H.J., Brandt, S.A.: Age-related changes in visual exploratory behavior in a natural scene setting. Frontiers in Psychology 4(339) (2013)

[HMM$^+$08] Horswill, M.S., Marrington, S.A., McCullough, C.M., Wood, J., Pachana, N.A., McWilliam, J., Raikos, M.K.: The hazard perception ability of older drivers. The Journals of Gerontology Series B: Psychological Sciences and Social Sciences 63(4), P212–P218 (2008)

[HNA$^+$11] Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., Van de Weijer, J.: Eye tracking: A comprehensive guide to methods and measures. Oxford University Press (2011)

[HSCG01] Ho, G., Scialfa, C.T., Caird, J.K., Graw, T.: Visual search for traffic signs: The effects of clutter, luminance, and aging. Human Factors: The Journal of the Human Factors and Ergonomics Society 43(2), 194–207 (2001)

[IK00] Itti, L., Koch, C.: A saliency-based search mechanism for overt and covert shifts of visual attention. Vision Research 40(10-12), 1489–1506 (2000)

[IKN98] Itti, L., Koch, C., Niebur, E.: A model of saliency-based visual attention for rapid scene analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(11), 1254–1259 (1998)

[Itt05] Itti, L.: Quantifying the contribution of low-level saliency to human eye movements in dynamic scenes. Visual Cognition 12(6), 1093–1123 (2005)

[Jol86] Jolliffe, I.T.: Principal Component Analysis. Springer, New York (1986)

[Kas13] Kasneci, E.: Towards the Automated Recognition of Assistance Need for Drivers with Impaired Visual Field. PhD thesis, University of Tübingen, Wilhelmstr. 32, 72074 Tübingen (2013)

[KCC12] Konstantopoulos, P., Chapman, P., Crundall, D.: Exploring the ability to identify visual search differences when observing drivers' eye movements. Transportation Research Part F: Traffic Psychology and Behaviour 15(3), 378–386 (2012)

[KJKG10] Komogortsev, O.V., Jayarathna, S., Koh, D.H., Gowda, S.M.: Qualitative and quantitative scoring and evaluation of the eye movement classification algorithms. In: Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications, ETRA 2010, pp. 65–68. ACM, New York (2010)

[KK13] Komogortsev, O.V., Karpov, A.: Automated classification and scoring of smooth pursuit eye movements in the presence of fixations and saccades. Behavior Research Methods 45, 203–215 (2013)

[KKKR14] Kasneci, E., Kasneci, G., Kübler, T.C., Rosenstiel, W.: The applicability of probabilistic methods to the online recognition of fixations and saccades in dynamic scenes. In: Proceedings of the Symposium on Eye Tracking Research and Applications, ETRA 2014, pp. 323–326. ACM, New York (2014)

[KKR$^+$14] Kübler, T.C., Kasneci, E., Rosenstiel, W., Schiefer, U., Nagel, K., Papageorgiou, E.: Stress-indicators and exploratory gaze for the analysis of hazard perception in patients with visual field loss. Transportation Research Part F: Traffic Psychology and Behaviour 24, 231–243 (2014)

[KSA$^+$14] Kasneci, E., Sippel, K., Aehling, K., Heister, M., Rosenstiel, W., Schiefer, U., Papageorgiou, E.: Driving with Binocular Visual Field Loss? A Study on a Supervised On-Road Parcours with Simultaneous Eye and Head Tracking. PLoS ONE 9(2), e87470 (2014)

[Lan09] Land, M.F.: Vision, eye movements, and natural behavior. Visual Neuroscience 26(1), 51–62 (2009)

[LT09] Land, M.F., Tatler, B.W.: Looking and acting: vision and eye movements in natural behaviour. Oxford University Press (2009)

[LZ06] Leigh, R.J., Zee, D.S.: The neurology of eye movements. Oxford University Press (2006)

[Mar10] Markoff, J.: Google cars drive themselves, in traffic. The New York Times 10, A1 (2010)

[McC81] McConkie, G.W.: Evaluating and reporting data quality in eye movement research. Behavior Research Methods & Instrumentation 13(2), 97–106 (1981)

[MS99] Maltz, M., Shinar, D.: Eye movements of younger and older drivers. Human Factors: The Journal of the Human Factors and Ergonomics Society 41(1), 15–25 (1999)

[MS04] Maltz, M., Shinar, D.: Imperfect in-vehicle collision avoidance warning systems can aid drivers. Human Factors: The Journal of the Human Factors and Ergonomics Society 46(2), 357–366 (2004)

[MSP08] Munn, S.M., Stefano, L., Pelz, J.B.: Fixation-identification in dynamic scenes: comparing an automated algorithm to manual coding. In: Proceedings of the 5th Symposium on Applied Perception in Graphics and Visualization, APGV 2008, pp. 33–42. ACM, New York (2008)

[MWGK12] Minka, T., Winn, J.M., Guiver, J.P., Knowles, D.A.: Infer.NET 2.5. Microsoft Research Cambridge (2012), `http://research.microsoft.com/infernet`

[Nag78] Nagayama, Y.: Role of visual perception in driving. IATSS Research 2, 64–73 (1978)

[NH10] Nuthmann, A., Henderson, J.M.: Object-based attentional selection in scene viewing. Journal of Vision 10(8), 20 (2010)

[NS71] Noton, D., Stark, L.W.: Eye movements and visual perception. Scientific American 224(6), 34–43 (1971)

[PHD$^+$05] Pradhan, A.K., Hammel, K.R., DeRamus, R., Pollatsek, A., Noyce, D.A., Fisher, D.L.: Using Eye Movements To Evaluate Effects of Driver Age on Risk Perception in a Driving Simulator. Human Factors: The Journal of the Human Factors and Ergonomics Society 47(4), 840–852 (2005)

[Pom89] Pomerleau, D.A.: ALVINN: An autonomous land vehicle in a neural network. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems 1, pp. 305–313. Morgan Kaufmann, San Francisco (1989)

[PS00] Privitera, C.M., Stark, L.W.: Algorithms for defining visual regions-of-interest: Comparison with eye fixations. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(9), 970–982 (2000)

[PS05] Privitera, C.M., Stark, L.W.: Scanpath theory, attention, and image processing algorithms for predicting human eye fixations. In: Itti, L., Rees, G., Tsotsos, J. (eds.) Neurobiology of Attention, pp. 269–299 (2005)

[SBG11] Schütz, A.C., Braun, D.I., Gegenfurtner, K.R.: Eye movements and perception: a selective review. Journal of Vision 11(9), 1–30 (2011)

[SD04] Santella, A., De Carlo, D.: Robust clustering of eye movement recordings for quantification of visual interest. In: Proceedings of the 2004 Symposium on Eye Tracking Research & Applications, ETRA 2004, pp. 27–34. ACM, New York (2004)

[See] Seeing Machines Inc. faceLab 5, http://www.seeingmachines.com/product/facelab/

[Sen] SensoMotoric Instruments GmbH. SMI BeGaze Eye Tracking Analysis Software, http://www.smivision.com/en/gaze-and-eye-tracking-systems/products/begaze-analysis-software.html

[SG00] Salvucci, D., Goldberg, J.: Identifying fixations and saccades in eye-tracking protocols. In: Proceedings of the 2000 Symposium on Eye Tracking Tesearch & Applications, ETRA 2000, pp. 71–78. ACM, New York (2000)

[SMDRV91] Sauter, D., Martin, B.J., Di Renzo, N., Vomscheid, C.: Analysis of eye tracking movements using innovations generated by a Kalman filter. Medical and biological Engineering and Computing 29(1), 63–69 (1991)

[SNP96] Summala, H., Nieminen, T., Punto, M.: Maintaining lane position with peripheral vision during in-vehicle tasks. Human Factors: The Journal of the Human Factors and Ergonomics Society 38(3), 442–451 (1996)

[SR] SR Research Ltd. EyeLink 1000 and EyeLink II, http://www.sr-research.com/index.html.

[SSC08] Shic, F., Scassellati, B., Chawarska, K.: The incomplete fixation measure. In: Proceedings of the 2008 Symposium on Eye Tracking Research & Applications, ETRA 2008, pp. 111–114. ACM, New York (2008)

[TGB03] Turano, K.A., Geruschat, D.R., Baker, F.H.: Oculomotor strategies for the direction of gaze tested with a real-world activity. Vision Research 43, 333–346 (2003)

[THH$^+$13] Tafaj, E., Hempel, S., Heister, M., Aehling, K., Schaeffel, F., Dietzsch, J., Rosenstiel, W., Schiefer, U.: A New Method for Assessing the Exploratory Field of View (EFOV). In: Stacey, D., SoléCasals, J., Fred, A.L.N., Gamboa, H. (eds.) HEALTHINF 2013, pp. 5–11. SciTePress (2013)

[THLB11] Tatler, B.W., Hayhoe, M.M., Land, M.F., Ballard, D.H.: Eye guidance in natural vision: reinterpreting salience. Journal of Vision 11(5), 5 (2011)

[TKK$^+$13] Tafaj, E., Kübler, T.C., Kasneci, G., Rosenstiel, W., Bogdan, M.: Online classification of eye tracking data for automated analysis of traffic hazard perception. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 442–450. Springer, Heidelberg (2013)

[TKP$^+$11] Tafaj, E., Kübler, T., Peter, J., Schiefer, U., Bogdan, M., Rosenstiel, W.: Vishnoo - an open-source software for vision research. In: Proceedings of the 24th IEEE International Symposium on Computer-Based Medical Systems, CBMS 2011, pp. 1–6. IEEE (2011)

[TKRB12] Tafaj, E., Kasneci, G., Rosenstiel, W., Bogdan, M.: Bayesian online clustering of eye movement data. In: Proceedings of the Symposium on Eye Tracking Research & Applications, ETRA 2012, pp. 285–288. ACM, New York (2012)

[Tob] Tobii Technology AB. Eye Tracking for Research and Analysis, `http://www.tobii.com/en/eye-tracking-research/global/`.

[UAB$^+$08] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., et al.: Autonomous driving in urban environments: Boss and the urban challenge. Journal of Field Robotics 25(8), 425–466 (2008)

[UCBC02] Underwood, G., Chapman, P., Bowden, K., Crundall, D.: Visual search while driving: skill and awareness during inspection of the scene. Transportation Research Part F: Traffic Psychology and Behaviour 5(2), 87–97 (2002)

[ULIS07] Urruty, T., Lew, S., Ihadaddene, N., Simovici, D.A.: Detecting eye fixations by projection clustering. In: ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), vol. 3(4), pp. 1–20 (2007)

[UPW$^+$05] Underwood, G., Phelps, N., Wright, C., Van Loon, E., Galpin, A.: Eye fixation scanpaths of younger and older drivers in a hazard perception task. Ophthalmic and Physiological Optics 25(4), 346–356 (2005)

[VBG12] Vidal, M., Bulling, A., Gellersen, H.: Detection of smooth pursuits using eye movement shape features. In: Proceedings of the Symposium on Eye Tracking Research and Applications, ETRA 2012, pp. 177–180. ACM, New York (2012)

[VRK$^+$02] Velichkovsky, B.M., Rothert, A., Kopf, M., Dornhöfer, S.M., Joos, M.: Towards an express-diagnostics for level of processing and hazard perception. Transportation Research Part F: Traffic Psychology and Behaviour 5(2), 145–156 (2002)

[Wid84] Widdel, H.: Operational problems in analysing eye movements. In: Gale, A.G., Johnson, F. (eds.) Theoretical and Applied Aspects of Eye Movement Research Selected/Edited Proceedings of The Second European Conference on Eye Movements. Advances in Psychology, vol. 22, pp. 21–29. North-Holland (1984)

[Woo02] Wooding, D.S.: Fixation maps: quantifying eye-movement traces. In: Proceedings of the Eye Tracking Research and Applications, pp. 31–36 (2002)

[Yar67] Yarbus, A.L.: Eye movements and vision. Plenum Press, New York (1967)

[Zee10] Zeeb, E.: Daimler's New Full-Scale, High-dynamic Driving Simulator–A Technical Overview. In: Proceedings of the Driving Simulator Conference Europe, pp. 157–165. Institut national de recherche sur les transports et leur sécurité (2010)

Input Transformation and Output Combination for Improved Handwritten Digit Recognition

Juan M. Alonso-Weber, M. Paz Sesmero, German Gutierrez,
Agapito Ledezma, and Araceli Sanchis

Abstract. Recent Neural Network Architectures with a deep and complex structure and that rely on ensemble averaging have led to an improvement in isolated handwritten digit recognition. Here a specific set of input pattern transformations is presented that achieves good results with modestly sized Neural Networks. Using some heuristics for the construction of an ensemble allows reaching low error rates.

Keywords: Artificial Neural Networks, Back Propagation, Ensembles, MNIST, Handwritten Digit Recognition.

1 Introduction

Handwritten character recognition has been spurred in recent years due to the use of new Neural Network Models such as Convolutional Neural Networks and Deep Neural Networks. Both are rather complex and deep structures, which allow reaching a highly respectable performance measured on the popular MNIST Dataset [1][2]: 0.4% [3] and 0.35% [4]. Combining these architectures with committees or integrating them with ensemble-like structures allows to further improve down to 0.27% [5] or even 0.23% [6]. Using committees with a traditional MLP displays an error rate of 0.39% [7]. Other interesting works which are based on different approaches reach an error rate of 0.40% [8] and 0.32% [9] respectively.

Juan M. Alonso-Weber · M. Paz Sesmero · German Gutierrez ·
Agapito Ledezma · Araceli Sanchis
Computer Science and Engineering Department, Universidad Carlos III de Madrid
Avenida de la Universidad 30 Leganés 28911, Madrid, Spain
e-mail: `jmaw@ia.uc3m.es`,
`{msesmero,ggutierr,ledezma,masm}@inf.uc3m.es`

The present work, derived from [10][11], shows an alternative based on a relatively modest sized Multilayer Perceptron (MLP) trained with the standard Back Propagation algorithm. In order to avoid local minima and stalling during the learning phase, several processing measures are added. The effective potential of the training set can be increased applying on the original MNIST digits some pattern distortion in combination with other transformations such as displacements and rotations [3]. Here an alternative deformation process is applied. An additional improvement is achieved using an input size reduction.

Noise injection has been extensively related with other techniques such as weight decay and regularization, and is known to provide a better generalization under certain circumstances [12][13][14]. A specific variant of input noise addition that uses annealing proves to be a robust tool for reaching low error rates in the MNIST domain.

The unstable nature of the MLPs promotes the use of an ensemble averaging procedure as a simple and effective method to achieve an improvement in the classification results [15]. For a better performance, the networks should be at most precise and diverse [16][17], i.e. it is desirable that the MLP's have a low error rate and that their errors have a low correlation. Inducing diversity through some random process, for example, training networks with different weight initialization is a correct procedure [15] but has a limited effectiveness [18]. A higher diversity can be accomplished through a guided design of the ensemble [19][20] or inducing additional differentiation in the training process [18], for example modifying the input space. Here both approximations are tried out: a) with a ranked selection methodology, and b) with a set of displacement schemas.

2 Data Processing

The MNIST Database contains 70000 digitized handwritten numerals distributed in ten different classes. The whole dataset is divided into 60000 images for training purposes, and the remaining 10000 are reserved for the test set. The graylevel values of each pixel are coded in this work in the [0, 1] interval, using a 0 value for white pixels and 1 for black ones.

An important point for managing a high performance in the learning process is the construction of a useful training set. The 60000 different patterns contained in the MNIST database can be seen as a rather generous set, but evidence shows that the learning curve of the usual methods based on Back Propagation converges in a few hundred cycles towards a local minimum, failing on one hundred or more of the test set samples [10]. Changing the usual learning parameters provides little improvement. A different strategy relies on increasing the training set cardinality and variability, which has a more profound impact on the learning capability of the MLP. Usual actions comprise geometric transformations such as displacements, rotation, scaling and other distortions. In this work an alternative deformation is combined with displacements, rotations and an additive input noise schedule that yields rather good results.

Another problem with which the Back Propagation algorithm tackles is the relative high input dimensionality for the original 28x28 sized digits. Using downsized images helps to reduce the error rate in a small amount, with the additional benefit of a lower computational cost. Therefore, a second version of both the training and test sets are generated where each pattern is downsized through interpolation to 20x20 pixels.

Displacement

Each digit is displaced combining two independent zero, one or two pixel shifts in the horizontal and vertical axis. The size of each shift is selected randomly. Experimental evidence indicates that longer and diagonal displacements induce a lower improvement in the final performance, whereas shorter shifts show to be more useful. Finding an optimal probability distribution of the different displacements is a cumbersome task. An interesting possibility is to train different MLPs using different displacement schemas. This might induce some differentiation between the MLPs, improving the accuracy of the ensemble. This is shown further on in the Experimental Results Section.

For most of the training cases a standard displacement schema is used. Table 1 shows the probability distribution for the displacement combinations. 0v, 1v and 2v stand for 0, 1 and 2 pixel displacements in the vertical axis, and 0h, 1h, 2h for the horizontal axis. Some of the combinations represent a similar, symmetric displacement (e.g. 0v+1h and 1v+0h).

Table 1 Probability distribution of the displacements for the standard schema

	Standard Displacement Schema					
Displacements	0v+0h	0v+1h 1v+0h	0v+2h 2v+0h	1v+1h	1v+2h 2v+1h	2v+2h
Probability	0.12	0.30	0.15	0.20	0.18	0.05

Deformation and Rotation

The most important transformation relies on the so called deformation, which involves pulling or pushing each of the four image corner pixels in a random amount along the vertical and horizontal axis. The rest of the pixels are proportionally displaced simulating an elastic behaviour. This leads to a combination of partial stretching and/or compression of the image. Fig. 1 illustrates this process. For the full sized images the displacement interval of the corner pixels is [-5, +5] (distance measured as pixels). For the 20x20 sized images, the best results are achieved with displacements in the order of [-4, +4] pixels. In parallel with the deformation, a rotation is applied around the image center selecting a random angle between -0.15 and +0.15 radians. For technical reasons, the deformation and the rotation need to be computed in an inverse way.

Noise Addition

The last process applied to each pattern before the training phase is the noise addition. A particular variant is used based on the annealing concept: starting with high noise rates that are lowered at a small amount after each learning cycle, and ending with a zero noise rate.

Including the descending input noise schedule improves the MLP precision, on behalf of a longer learning process. A noiseless training requires about 500 cycles, whereas adding the input noise extends the learning up to 1000 cycles. In this circumstance, convergence is tempered down towards the end of the noise schedule. Usually few improvements are achieved during the 100-200 last cycles, and virtually none after the noise scheme is extinguished. The noise parameters and their relation are $R{=}N_0/T_{max}$, where T_{max} stands for the number of training cycles, R is the noise reduction value and N_0 is the initial noise value. Using an initial noise value of N_0=1.0 adds to each pixel a uniform random value from the interval [-1, +1]. For a 1000 cycle training, the value for R should be 0.001.

The noise addition and the geometrical transformations are applied to each pattern for each Back Propagation training cycle. Hence, the MLP sees each pattern just once.

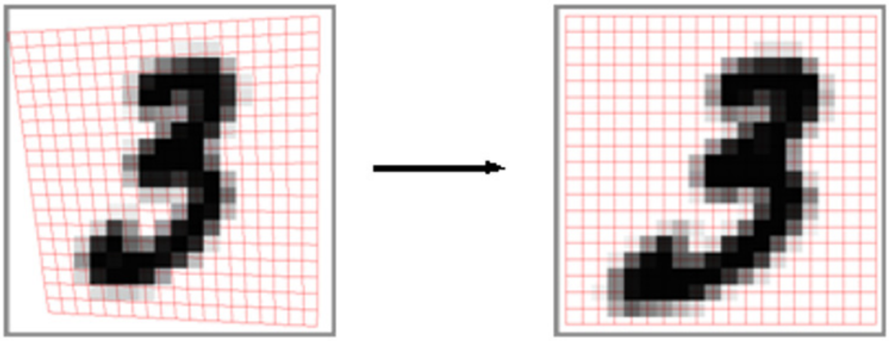


Fig. 1 The deformation is shown as an inverse mapping, at the left the original digit, at the right the deformed one

Output Averaging

Due to the inherent randomness of the training with the Back Propagation algorithm and the input transformations, the results of the Neural Networks will have a high variability. The construction of an ensemble with a set of trained Neural Networks can take advantage of this variability improving the accuracy in relation to their individual performance. The ensemble is constructed with the averaging of the outputs of the networks. A higher accuracy will be induced applying two heuristics: a) with a ranked selection methodology, and b) with a set of displacement schemas. Further details are shown in the Experimental Results Section.

3 Experimental Setup

All the experimentation is built up around training a collection of Multilayer Perceptrons, which are afterwards used to apply the proposed ensemble averaging.

The MLPs have a fixed size by default: $784 \times 300 \times 200 \times 10$, for the full sized image database. Each output unit activates only for a specific class following the one-out-of-n schema. The only variation in size is for the downsized 20×20 images where 400 input units are required. The training process is performed using online Back Propagation, where all patterns are presented to the input in a random order. All the patterns are processed in each cycle applying the above mentioned geometrical transformations (deformation, rotation and displacement) combined with the noise addition. The weight initialization is restricted to the [-0.3, +0.3] interval, and the Learning Rate is set to 0.03. The activation function is the usual logistic sigmoid.

A unique subset of the training patterns (10000 out of 60000) is randomly removed for validation purposes. This validation set can be used in several ways during the neural network training facing the posterior ensemble averaging: at first, for determining the stopping point of the learning process, and secondly as a criterion for establishing a ranking inside a set of trained neural networks.

As already stated, including the descending input noise schedule improves the final MLP precision, at the cost of a longer lasting learning process. As a rule, the annealing scheme lasts $T_{max}=1000$ cycles, and 100 noiseless cycles are added at the final stage in order to ensure a stable convergence. The initial noise value is $N_0=1.0$, and the descending noise rate $R=0.001$.

The training process of the MLPs was performed on Intel Core i7 and equivalent Xeon processors. Each processor allows to train up to 8 MLPs in parallel without a noticeable loss of performance. Given the size of the training set and the needed cycles, the whole learning process lasts about 20 hours for the 20x20 images, and 24 hours for the full sized images.

4 Experimental Results

This section presents the experiments performed at first with the full sized images, and then with the downsized images that achieve a slightly better performance.

Experiment 1, Ranked Selection with 28x28 Sized Images.

At first, a set of 90 MLPs are trained with 50000 images from the MNIST database, leaving 10000 randomly chosen digits for validation. Applying ensemble averaging on the whole MLP set gives an error rate of 0.39%.

Following the idea behind the statement that "many could be better than all" [19][20], a methodology for selecting the MLPs for the averaging procedure is proposed: validation errors are used in order to establish a ranking for the trained MLPs. The 20 best ranked MLPs are distributed into four subsets named *p*, *q*, *r* and *s*, where *p* contains the five neural networks that perform best on the validation set, and *s* the worst. Several ensembles are then built starting with the best subset (*p*), and progressively adding the subsets *q*, *r* and *s*. Table 2 shows the Test Errors committed for these ensembles. Also shown are the evolution of the worst, the best and the mean Test Errors for the selected MLPs. The mean Test Error is lower for the better ranked ones. Experiments performed on various MLP sets with different cardinalities suggest that using different seeds for weight initialization derives in a limited differentiation, i.e. the individual MLPs have highly correlated errors. The ensemble averaging tends to acquire the best performance with nine to fifteen members.

Table 2 Test Errors (in %) for the ensembles built with the progressive addition of the MLPs with the best validation values

		Ranked MLPs				All
	(sub) set	p	p, q	p, q, r	p, q, r, s	90
MLPs	Mean	0.488	0.485	0.495	0.492	0.51
	Min	0.46	0.45	0.45	0.45	0.43
	Max	0.52	0.53	0.61	0.61	0.61
Ensemble		0.38	**0.36**	**0.36**	0.40	0.39

In order to establish a reference for evaluating the heuristic, another procedure for building ensembles is included: four *K*-sized ensembles (*K*=5, 10, 15, 20) whose members are randomly selected on 1000 trials from the whole MLP set (i.e. 90 members). This allows establishing a mean value for reference. The results in Table 3 show that this value converges steadily to 0.39%. The ranked selection methodology gives a slightly better performance at 0.36%.

Table 3 Test Errors (in %) for 1000 ensembles built with *K* randomly selected MLPs

		Randomly selected MLPs				All
	K=	5	10	15	20	90
Ensembles	Mean	0.406	0.397	0.394	0.393	0.39
	Min	0.33	0.33	0.33	0.33	
	Max	0.49	0.47	0.45	0.45	

Experiment 2, Differentiation with Displacement Schemas, 20x20 Sized Images.

Whereas the MLPs trained on the original MNIST database achieve a mean error rate of 0.51% (see Table 2), using the 20x20 downsized images allows for a slightly lower 0.46% using the standard displacement schema D6 (see Table 5). For these MLPs, the averaging procedure provides an error rate of 0.365%. The following experiment shows that building these MLPs with a specific differentiation indeed improves the results.

Although the use of the continuous random deformations generates a training set with a virtually unlimited number of patterns, in practice, leaving out a fraction of the original pattern set for validation purposes leads to a descent in performance, especially with the 20x20 sized image set. Therefore, in the following experiments the whole original training set without any geometrical transformations is used for validation purposes. The drawback is that this particular validation set does not allow using the ranked selection method on the trained MLPs: the number of validation errors seems to have low relation with the behaviour of the MLPs on the test set, which leads to no benefit in the averaging procedure.

In order to increase the diversity between the trained MLPs, instead of using a unique displacement method, different displacement schemas are established. Each schema relies on a different probability distribution of the possible displacements (shown in Table 4), and is combined with the usual geometric transformation and noise addition.

For this experiment 10 Neural Networks for each displacement schema were trained (on 20x20 images). The Mean Test Errors for the six groups varies between 0.465% (D6) and 0.496% (D4), as shown in Table 5. Averaging each MLP group provides a decrease in the error rates that varies between 0.36% and 0.41%. The full ensemble contains 60 members and displays an error rate of 0.34%.

Table 4 Probability distribution of the displacements inside each schema. D6 is the standard displacement schema used for training MLPs in the previous experiment.

	Displacement Schema					
Displacements	D1	D2	D3	D4	D5	D6
0v+0h	0.43	0.27	0.22	0.20	0.15	0.12
0v+1h,1v+0v	0.57	0.49	0.45	0.32	0.41	0.30
0v+2h,2v+0v	-	0.16	0.22	0.32	0.05	0.15
1v+1h	-	0.08	0.11	0.16	0.31	0.20
1v+2h,2v+1v	-	-	-	-	0.08	0.18
2v+2h	-	-	-	-	-	0.05

Table 5 Ensembles built with six MLP sets trained each with a different displacement schema

	Displacement Schema					
	D1	D2	D3	D4	D5	D6
MLP	0.487%	0.468%	0.469%	0.496%	0.485%	0.465%
Ensembles	0.39%	0.36%	0.36%	0.36%	0.41%	0.36%
Full Ensemble	**0.34%**					

Averaging all the MLPs trained with the six different displacement schemas shows a lower error rate than those committed by the best ensemble based on an individual displacement schema.

5 Conclusions

This work shows that improving the recognition rate of Handwritten Digits with modestly sized Neural Networks trained with the standard Back Propagation algorithm can be accomplished applying a set of specific input transformations. These transformations increase the cardinality of the training set, which helps the learning process to avoid stalling and local minima. Combining the outputs of a set of trained Neural Networks takes advantage of the variable nature of these, allowing to improve the accuracy of the ensemble in relation to the individual members. Using some specific heuristics such as a ranked selection or different displacement schemas can push the accuracy one step further.

Acknowledgments. This work has been supported by the Spanish Government under projects TRA2010-20225-C03-01, TRA2011-29454-C03-02 and TRA2011-29454-C03-03.

References

[1] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE 86(11), 2278–2324 (1998)

[2] LeCun, Y., Cortes, C.: THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/

[3] Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the 2003 Seventh International Conference on Document Analysis and Recognition, vol. 1, pp. 958–963 (2003)

[4] Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, Big, Simple Neural Nets for Handwritten. Neural Comput. 22(12), 3207–3220 (2010)

[5] Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Convolutional Neural Network Committees for Handwritten Character Classification. In: 2011 International Conference on Document Analysis and Recognition, vol. 10, pp. 1135–1139 (2011)

[6] Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column Deep Neural Networks for Image Classification. In: IEEE Conf. on Computer Vision and Pattern Recognition CVPR 2012, pp. 3642–3649 (2012)

[7] Meier, U., Ciresan, D.C., Gambardella, L.M., Schmidhuber, J.: Better Digit Recognition with a Committee of Simple Neural Nets. In: 2011 Int. Conf. Doc. Anal. Recognit., vol. 1, pp. 1250–1254 (September 2011)

[8] Ranzato, A.M., Poultney, C., Chopra, S., LeCun, Y.: Efficient Learning of Sparse Representations with an Energy-Based Model. In: Advances in Neural Information Processing Systems 19, NIPS 2006 (2006)

[9] Cruz, R., Cavalcanti, G., Ren, T.: Handwritten digit recognition using multiple feature extraction techniques and classifier ensemble. In: Int. Conf. Syst. Signals Image Process, pp. 215–218 (2010)
[10] Sesmero, M.P., Alonso-Weber, J.M., Gutiérrez, G., Ledezma, A., Sanchis, A.: A new artificial neural network ensemble based on feature selection and class recoding. Neural Comput. Appl. 21(4), 771–783 (2012)
[11] Alonso-Weber, J.M., Sanchis, A.: A Skeletonizing Reconfigurable Self-Organizing Model: Validation Through Text Recognition. Neural Process. Lett. 34(1), 39–58 (2011)
[12] Matsuoka, K.: Noise Injection into Inputs in Back-Propagation Learning. IEEE Trans. Syst. MAN, Cybern. 22(3), 436–440 (1992)
[13] An, G.: The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. Neural Comput. 8, 643–674 (1996)
[14] Bishop, C.M., Avenue, J.J.T.: Training with Noise is Equivalent to Tikhonov Regularization. Neural Comput. 7(1), 108–116 (1995)
[15] Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. J. Artif. Intell. Res. 11(1), 169–198 (1999)
[16] Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000)
[17] Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms, vol. 47(4). Wiley-Interscience (2005)
[18] Brown, G., Wyatt, J., Harris, R., Yao, X.: Diversity creation methods: A survey and categorisation. Inf. Fusion 6(1), 5–20 (2005)
[19] Perrone, M.P., Cooper, L.N.: When networks disagree: Ensemble methods for hybrid neural networks. In: Mammone, R.J. (ed.) Neural Networks for Speech and Image Processing, ch. 10. Chapman-Hall (1993)
[20] Sharkey, A.J.C., Sharkey, N.E., Gerecke, U., Chandroth, G.O.: The 'Test and Select' Approach to Ensemble Combination. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 30–44. Springer, Heidelberg (2000)

Feature Selection for Interval Forecasting of Electricity Demand Time Series Data

Mashud Rana, Irena Koprinska, and Abbas Khosravi

Abstract. We consider feature selection for interval forecasting of time series data. In particular, we study feature selection for LUBEX, a neural network based approach for computing prediction intervals and its application for predicting future electricity demands from a time series of previous demands. We conduct an evaluation using half-hourly electricity demand data for Australia and the United Kingdom. Our results show that the mutual information and correlation-based feature selection methods are able to select a small set of lag variables that when used with LUBEX construct valid prediction intervals in most cases (coverage probability of 97.44% and 96.68% for the Australian data, and 88.32% and 91.89% for the British data, for confidence level of 90%). However, the widely used partial autocorrelation feature selection method failed to do this (coverage probability of 69.69% for the Australian data and 47.07% for the British data).

Keywords: Electricity demand forecasting, prediction intervals, uncertainty quantification, neural networks, feature selection, mutual information, correlation.

1 Introduction

We consider that the task of forecasting the future electricity demand (load) from a time series of previous electricity demands. In particular, we study a time series of electricity demands measured every half an hour and our goal is to make predictions for the next value. This task is classified as very short-term electricity demand forecasting and is important for the operation of electricity markets and

Mashud Rana · Irena Koprinska
School of Information Technologies, University of Sydney, Sydney, Australia
e-mail: {mashud.rana,irena.koprinska}@sydney.edu.au

Abbas Khosravi
Centre of Intelligent Systems Research, Deakin University, Geelong, Australia
e-mail: abbas.khosravi@deakin.edu.au

the smart grid. We focus on *interval forecasting* as opposed to *point forecasting*. In interval forecasting, at time t the task is to predict an interval of electricity demand values for time $t+h$ with a certain probability. In point forecasting, the task is to predict a single demand value for time $t+h$.

More formally, our task can be defined as follows: given a time series of n previous half-hourly electricity demands $X_1, X_2, \ldots, X_n$, the goal is to forecast a Prediction Interval (PI) for the next value of the series X_{n+1}. A PI consists of a lower bound L and an upper bound U, between which the future value is expected to lie with a minimum pre-specified probability μ. Thus, a PI for X_{n+1} is a valid PI if the probability of X_{n+1} to be between the PI's lower and upper bound is equal or greater than the pre-specified confidence level, i.e. the following condition is satisfied: $P(\, L(X_{n+1}) \leq X_{n+1} \leq U(X_{n+1})\,) \geq \mu$.

Interval forecasts give more information about the variability of the target variable and the associated uncertainty. They are more suitable than point forecasts for risk management, especially in applications requiring balancing of demand and supply, e.g. electricity and financial markets, stock manufacturing and inventory management [1].

Most of the existing approaches for very short-term electricity demand forecasting are concerned with point forecasting. These approaches fall into two main categories: using statistical methods such as exponential smoothing, autoregressive moving average and regression-based models [2-7], and using Neural Networks (NNs) [5, 7-12]. Although interval forecasting is very useful for electricity markets, it still hasn't received enough attention. Recently, a method for predicting PIs using NNs, called LUBE, was proposed in [13]. To predict PIs for new data, it uses the point forecasts for the training data and a novel cost function that is minimized during the NN training. LUBE was compared with other NN-based methods for PI construction such as the delta [14], Baysian [15] and bootstrap [16] methods, and was shown to generate valid PIs, typically outperforming the other methods. In our previous work [17] we proposed an extension of LUBE, called LUBEX, which utilizes an ensemble of NNs instead of a single NN to reduce the sensitivity of LUBE's performance to the NN architecture, random weight initialization and random weight perturbation during training.

In this paper we further extend [13, 17] by studying feature selection for interval forecasting. The few existing methods for interval forecasting haven't investigated the effect of feature selection on the quality of PIs. In this paper we investigate the performance of three feature selection methods – Partial Autocorrelation (PA), Mutual Information (MI) and Correlation-based Feature Selection (CFS). PA is widely used in time series analysis; MI and CFS are less popular for time series analysis but are state-of-the-art machine learning methods successfully used in many applications. We utilized these feature selection methods in conjunction with the interval forecasting approach LUBEX, and evaluate their performance for electricity demand forecasting. This paper is an extended version of [18] – it provides a more comprehensive evaluation by using electricity demand data for two countries: Australia and the United Kingdom.

The rest of the part of this paper is organized as follows. The next section discusses the data and data characteristics. Section 3 presents the measures used to evaluate the quality of the constructed PIs. Section 4 briefly reviews the LUBEX method. Section 5 summarizes the feature selection methods and describes how they were applied to our task. Section 6 presents the results and discusses them, and finally Section 7 concludes the paper.

2 Data and Data Characteristics

2.1 Data

We use half-hourly electricity demand data for two countries: Australia and the United Kingdom, for one year: 2010. The Australian data is for the state of New South Wales and is publicly available from the website of the Australian Energy Market Operator [19]. The British data is also publicly available from the website of the National Grid [20].

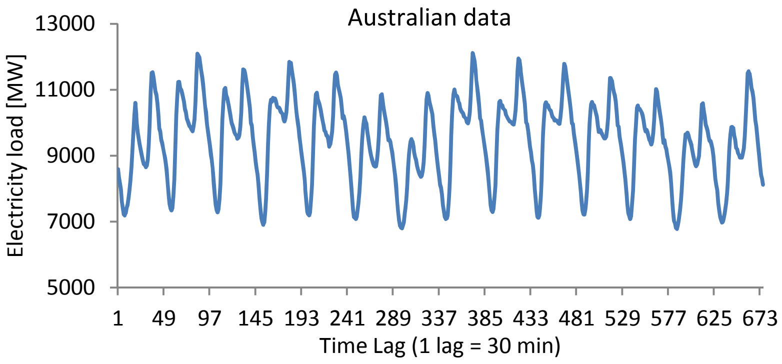
The data is recorded for each month of the year. We study each month separately in 12 case studies, one for each month. The total number of samples in each dataset is 8,760. The number of samples for each case study is between 1,344 (for February) and 1,488 (for the 31 days months).

The data for each case study is divided into three non-overlapped subsets - training set (D_{train}), validation set (D_{valid}) and testing set (D_{test}). The training set contains 50% of the data and is used for feature selection and building of the prediction models. The validation set contains 30% of the data and is used for selecting the best ensemble of NNs for the LUBEX method. The testing set contains the remaining 20% of the data and is used for performance evaluation.

2.2 Data Characteristics

The electricity demand time series data is complex and non-linear, and includes both cyclic and random components. There are three main cycles - daily, weekly and annual. The random components are caused by fluctuations in the electricity usage of individual users, large industrial loads with irregular hours of operation, special events, holidays, extreme weather and sudden weather changes.

Fig. 1 plots the electricity demand data for the Australian and British data for the same fortnight period - from 2nd Aug (Monday) to 15th Aug (Sunday), 2010. The two graphs are similar and we can clearly see the daily and weekly cycles of the electricity demand data. The daily cycle is evident from the similarity of the electricity demand profiles of the individual days, e.g. the demand profile for Wednesday is similar to the profile for Thursday in the same week. The daily demand profile is consistent with the human activity – the electricity demand is higher during the day, with a peak in the morning and evening, and lower during the night. The weekly cycle is evident from the similarity of demand profiles of the two weeks – e.g. the electricity demand on the two Wednesdays is very similar, as is the demand for the other days of the week. In addition, and as expected, the electricity demand during the business days (Monday to Friday) is higher than during the weekend (Saturday and Sunday). This difference is greater for the British data than for the Australian data.



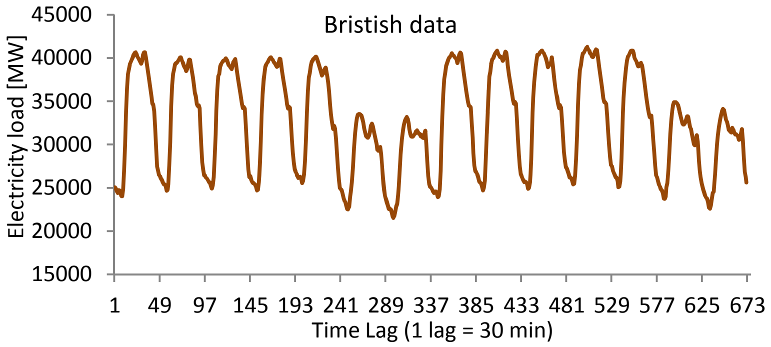


Fig. 1 Half-hourly electricity demand for two consecutive weeks: 2 August (Monday) – 15 August (Sunday), 2010

Although the two graphs in Fig. 1 are similar due to the similar cyclic nature of the electricity demand for the two countries, the range of values is different– the British electricity demand is much higher than the Australian. For example, for the 2 weeks period in Fig. 1, the Australian demand varies between 6,779 and 12,119 MW and the British demand varies between 21,519 and 41,307 MW.

2.3 *Irregular Days and Weather Variables*

Irregular days are days with extreme weather or sudden weather change, public and school holidays and other special events. Since the electricity demand on such days is different than the demand on normal days, these days are usually identified and treated separately as outliers. When building a prediction model, the demand for these days is typically replaced by the mean demand of the previous day or weeks [3]. When making online predictions, the regular prediction model is replaced by a pre-computed forecast for special days. In this study, we didn't detect outlier days. We treated all days equally and didn't apply any smoothing which might have resulted in lower accuracy. We plan to investigate irregular days in future work.

Another factor that influences the forecasting accuracy is the weather, e.g. air temperature and humidity, wind direction and speed. However, for small forecasting horizons such as half an hour ahead prediction, it is considered that the weather changes are already reflected in the electricity demand series. Taylor et al. [3, 21] found that the use of weather variables was beneficial only for forecasting horizons greater than several hours. Hence, in this study we only use previous electricity demand data and don't use weather data.

3 Measures for Evaluating the Quality of PIs

A PI has two important properties: coverage probability and width. A good quality PI will have high coverage probability (higher than the predefined confidence level μ) and a small width. The coverage probability has been widely used to assess the quality of PIs, while the interval width has only recently been considered. In this paper we use measures that consider both properties of PIs; in particular, we use the following three measures introduced in [13]: Prediction Interval Coverage Probability (PICP), Prediction Interval Normalized Average Width (PINAW), and their combination, Coverage Width Based Criterion (CWC).

PICP. Given a data set of N examples, PICP is the probability that the target value X_i of the i-th example will fall between the upper bound U_i and lower bound L_i of the prediction interval PI_i, averaged over all examples i. It is calculated empirically by counting the target values that fall within the bounds:

$$PICP = \frac{1}{N}\sum_{i=1}^{N} c_i \,.\, 100\%, \text{where}$$

$$c_i = \begin{cases} 1, & if\ X_i \in [L_i, U_i] \\ 0, & otherwise \end{cases}$$

PIs that do not satisfy the condition $PICP \geq \mu$, where μ is a predefined confidence level are not reliable. In this paper we use μ =90%.

PINAW. PINAW measures the average width of the PIs for all examples in the data set, normalized by the range of the actual target R:

$$NMPIW = \frac{1}{N\,R}\sum_{i=1}^{N}(U_i - L_i)$$

CWC. CWC combines PICP and PINAW using the parameters η and γ, which determine the weighting of the two components:

$$CWC = PINAW\left(1 + \gamma e^{-\eta(PICP-\mu)}\right), \text{ where}$$

$$\gamma = \begin{cases} 0, & if\ PICP \geq \mu \\ 1, & otherwise \end{cases}$$

CWC has two main principles:

1) If the coverage probability is above the confidence threshold, CWC should depend only on the PI's width. This is achieved by setting γ to 0; CWC becomes equal to the width PINAW and has a low value;

2) If the coverage probability is below the confidence threshold, i.e. the PIs are not valid, CWC should have a high value, regardless of the width. This is achieved by using a high value for η in the exponential term and by setting γ to 1 to consider this term. Due to the high value of the exponential term, the influence of PINAW is lost and CWC becomes high.

Thus, CWC balances the PI's usefulness (narrow width) and correctness (acceptable coverage probability). An analysis of CWC is presented in [13].

4 The LUBEX Method

LUBEX [17] is a recently proposed method for computing PIs using an ensemble of NNs. It is an extension of the LUBE method [13]. A single LUBE NN is a multilayer perceptron with p input neurons, corresponding to the input variables of each example, two output neurons corresponding to the lower and upper PI bounds for this example and one or more hidden layers. A LUBE NN is trained to minimize CWC using the simulated annealing algorithm which combines hill-climbing and random walk. Note that the backpropagation algorithm cannot be applied as CWC is not differentiable. During training, the two targets U_{i+1} and L_{i+1} of PI_{i+1} are both set to the target point forecast X_{t+1}. The trained NN is then used to predict the PIs for the testing data.

A single LUBE NN is sensitive to the network architecture, random initialization of weights and random perturbation of weights during training. To reduce this sensitivity, LUBEX uses an ensemble method that combines LUBE NNs. It considers NNs with one hidden layer and constructs n NN architectures $A_1,..,A_n$ with 1 to n hidden neurons, respectively (we used n=30 in our experiments). For each NN architecture A_i, it builds an ensemble E_i of m NNs (m=100 in our experiments). The ensemble members of E_i have the same architecture A_i but are initialized to different random weights. Each of them is trained on the training set.

The prediction process for a new example is illustrated in Fig. 2. To predict the PI for the new example *i*, E_i combines the predictions of its members by taking the median of their lower and upper bounds: $PI_i = [median(L_{i1},..L_{im}), median(U_{i1},...,U_{im})]$.

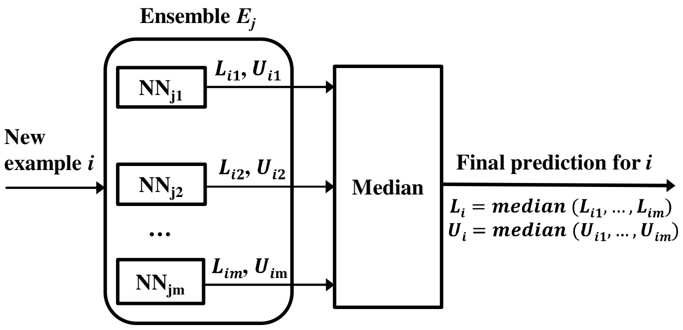


Fig. 2 Predicting a new example by ensemble E_j

The *n* ensembles are evaluated on the validation set, the best one is selected (the one with smallest CWC) and then used to predict the test data. Rana et al. [17] found that the use of ensemble in LUBEX is very important. They compared the performance of LUBEX using an ensemble of NNs and using a single NN, and found that LUBEX with a single NN was not able to generate PIs with satisfactory coverage probability in many cases and that its performance varied considerably for multiple runs of the algorithm.

5 Feature Selection for Constructing PIs

5.1 CFS

CFS [22] is a state-of-the-art filtering algorithm for feature subset selection. Given a set of candidate features, it uses a search algorithm to find the best possible feature subset *S*, the one that maximizes the following heuristic measure:

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k+k(k-1)\overline{r_{ff}}}},$$

where *k* is the number of features, features $\overline{r_{cf}}$ is the average feature to class variable correlation and $\overline{r_{ff}}$ is the average feature to feature correlation.

This heuristic favors feature subsets where the features are good individual predictors of the class variable ($\overline{r_{cf}}$ is high) but are not correlated with each other ($\overline{r_{ff}}$ is low). It is suitable for our task as the candidate features (previous lag variables) are highly correlated with each other.

To conduct feature selection using CFS, we first form a set of candidate features that includes all lag variables from a 1-week sliding window from the training data. We found that one week is a sufficient length as it captures both the daily and weekly patterns of the electricity demand data. As we are using half-hourly data, the candidate set consists of 336 features. To select the best feature subset for these 336 features we applied a best-first search algorithm; applying an exhaustive search algorithm is not possible due to the big number of possible subsets. The feature selection is done separately for each case study (month) and dataset. Table 1 lists the features selected by CFS for all case studies; the total number of selected features is shown in brackets.

Table 1 Selected features (lag variables) using CFS

Case study	Australian data	British data
1 (Jan)	1, 46, 120, 336 (4)	1, 48, 65, 120, 216, 335 (6)
2 (Feb)	1-3, 48, 120, 217, 288, 333-335 (10)	1-2, 47, 192, 216, 335-336 (7)
3 (Mar)	1, 48, 115, 336 (4)	1-2, 48, 167, 335 (5)
4 (Apr)	1-2, 47, 121, 287, 336 (6)	1-2, 144, 287, 334 (5)
5 (May)	1, 48, 223, 335-336 (5)	1, 168, 288, 336 (4)
6 (Jun)	1-2, 47, 158, 225, 335 (6)	1, 48, 168, 335 (4)
7 (Jul)	1, 48, 225, 255, 288, 335 (6)	1, 48, 120, 144, 288, 335 (6)
8 (Aug)	1, 24, 48, 288, 335 (5)	1, 48, 335-336 (4)
9 (Sep)	1, 48, 208, 220, 288, 335 (6)	1, 48, 288, 335-336 (5)
10 (Oct)	1-2, 119, 144, 210, 286, 335 (7)	1, 48, 218, 335 (4)
11 (Nov)	1-3, 20, 95, 192, 216, 334-336 (10)	1, 48, 168, 288, 335 (5)
12 (Dec)	1-2, 48, 120, 216, 335 (6)	1, 167, 288, 335-336 (5)

The CFS variable selection is consistent with the daily and weekly cycles of the electricity demand data. For all case studies CFS selects variables corresponding to the electricity demand from the previous few lags (e.g. 1-3), the previous day around the prediction time (e.g. 46-48) and the previous week around the prediction time (e.g. 335-336).

5.2 MI

MI measures the dependence between two variables. If the two variables are independent, MI is zero; if they are dependent, MI has a positive value corresponding to the strength of the dependency. Unlike CFS, MI can identify both linear and non-linear dependencies. Therefore, it is an appropriate feature selector for electricity demand forecasting as it can capture both linear and non-linear relationships between the lag variables and the target variable.

Computing MI for continuous variables is more difficult than for nominal variables as it requires assumption about the data distribution. In this paper we apply a method for MI estimation based on *k*-nearest neighbor distances [23]. This method has a minimal bias and was shown to be efficient and reliable.

To conduct feature selection using MI, we again form a set of candidate features that includes all lag variables from a 1-week sliding window from the training data. Then we compute the MI score between each candidate feature and the target variable and rank the candidate features in decreasing order based on their MI score. Fig. 3 shows the normalized MI score for all 336 features from the 1-week window in ranked order, for all case studies.

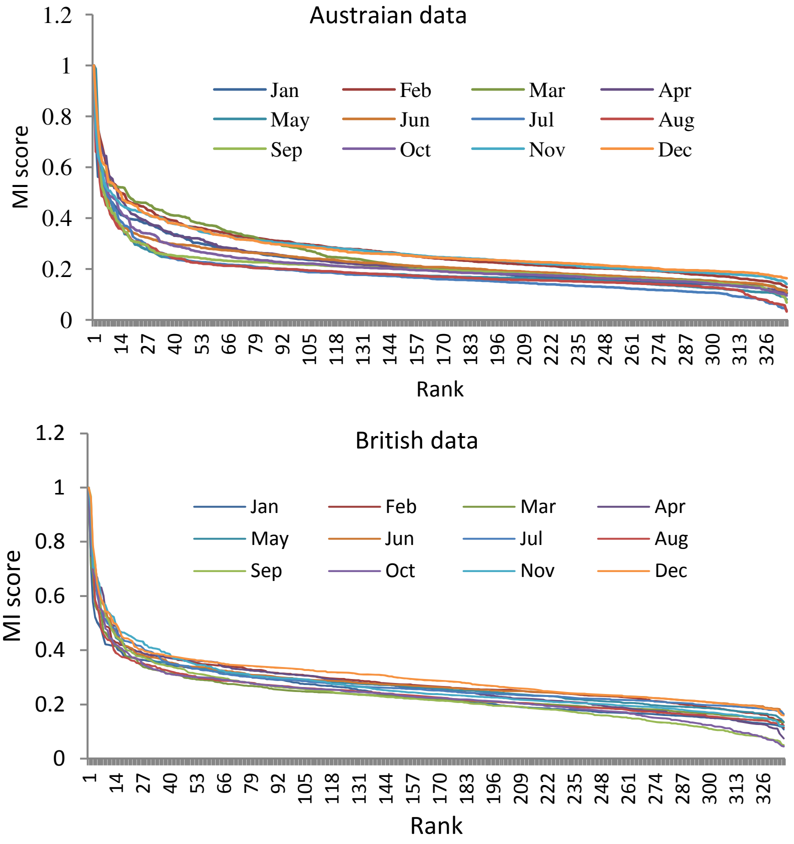


Fig. 3 MI score and ranking of features

We can make the following observations: 1) The graphs for all case studies for both datasets are very similar; 2) The MI score decreases abruptly at the beginning till about ranked feature 10 and then continues to decrease gradually before almost flattening. Based on these results, we select the top 10 highly ranked features for each case study and disregard the remaining features. This means that only 3% of the candidate features are selected, which is a substantial feature reduction.

The selected features are listed in Table 2. Similarly to CFS, MI also selects variables that reflect the daily and weekly cycles of the electricity demand data – lag variables from the same day, the previous day and the previous week, just before or after the prediction time.

Table 2 Selected lag variables using MI

Case study	Australian data	British data
1 (Jan)	1-4, 47-50, 335-336 (10)	1-5, 47-48, 288, 335-336 (10)
2 (Feb)	1-3, 47-49, 289, 334-336 (10)	1-4, 47-49, 334-336 (10)
3 (Mar)	1-4, 48, 144, 287-288, 335-336 (10)	1-4, 47-49, 288, 335-336 (10)
4 (Apr)	1-3, 48-49, 287-289, 335-336 (10)	1-2, 48, 286-289, 334-336 (10)
5 (May)	1-3, 48-49, 287-288, 334-336 (10)	1-3, 48, 286-289, 335-336 (10)
6 (Jun)	1-4, 47-49, 288, 335-33 (10)	1-3, 47-49, 288, 334-336 (10)
7 (Jul)	1-3, 48-49, 144, 288, 334-336 (10)	1-3, 48-49, 144, 288, 334-336 (10)
8 (Aug)	1-3, 47-49, 287-288, 335-336 (10)	1-3, 48-49, 288, 333-336 (10)
9 (Sep)	1-3, 47-49, 288, 334-336 (10)	1-3, 47-49, 288, 334-336 (10)
10 (Oct)	1-3, 48, 286-288, 334-336 (10)	1-3, 47-50, 334-336 (10)
11 (Nov)	1-5, 48, 49,288, 335-336 (10)	1-3,48-49, 287-289, 335-336 (10)
12 (Dec)	1-4, 48, 287-288, 334-336 (10)	1-3, 48, 287-289, 334-336 (10)

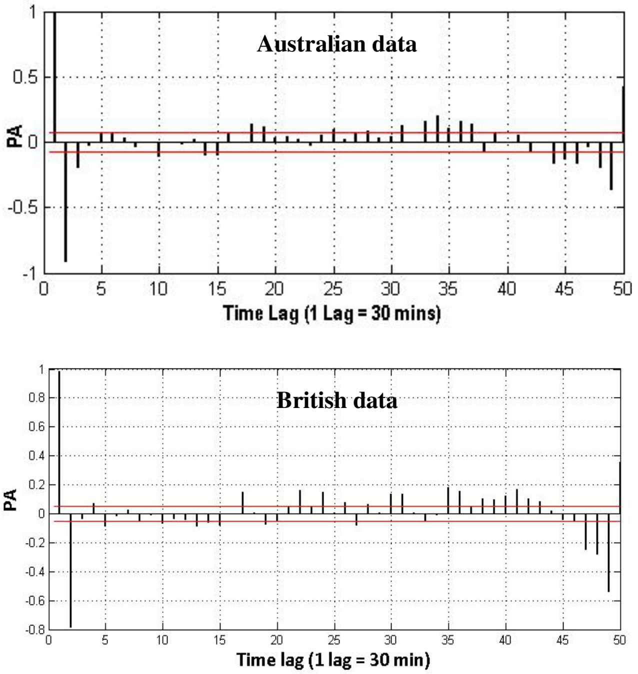


Fig. 4 PA function for case study 1 for the Australian and British data

5.3 PA

PA is a widely used feature selection method for time series data. The PA value between two observations X_t and X_{t-h} in a time series is the linear correlation between them, conditional on $X_{t-h+1}, \ldots, X_{t-1}$, the set of observations between them. PA only measures linear dependencies; a value close to 1 or -1 shows a strong positive or negative dependency, while a value close to 0 shows no dependency.

To form a feature set, we compute the PA till lag 336 (i.e. a 1-week sliding window) using the training data, for each case study separately. As an example, Fig. 4 shows the PA function for the first 50 lags of case study 1 for the Australian and British data. We then select the lags with PA higher than the confidence threshold (shown with the two parallel horizontal lines in Fig. 4). The selected features are not shown due to space limitation. For the Australian data, their number varied between 39 for case study 3 and 68 for case study 6. For the British data, their number was between 33 for case study 3 and 56 for case study 12. Compared to CFS and MI, PA selects a higher number of features.

6 Results and Discussion

To evaluate the quality of the constructed PIs we compute the three performance measures: PICP, PINAW and CWC. The results are presented in Tables 3 and 4 and discussed in the next three subsections. Each value in the tables is the average

Table 3 PIs constructed by LUBEX with CFS, MI and PA for the Australian data

Case	CFS			MI			PA		
Study	PICP	PINAW	CWC	PICP	PINAW	CWC	PICP	PINAW	CWC
1 (Jan)	98.35	19.27	19.27	99.91	12.55	12.55	71.60	7.31	$1.3x10^{16}$
2 (Feb)	100.0	19.64	19.64	100.0	17.04	17.04	68.61	9.03	$6.4x10^{14}$
3 (Mar)	98.17	18.46	18.46	99.91	12.40	12.40	73.16	6.07	$1.4x10^{12}$
4 (Apr)	91.81	13.76	13.76	91.40	10.33	10.33	70.23	6.54	$2.1x10^{16}$
5 (May)	95.22	13.19	13.19	97.22	9.69	9.69	55.74	5.08	$2.8x10^{14}$
6 (Jun)	92.49	11.13	11.13	91.95	8.84	8.84	57.39	4.55	$1.5x10^{16}$
7 (Jul)	98.18	16.85	16.85	99.04	10.83	10.83	63.03	4.76	$1.9x10^{14}$
8 (Aug)	98.27	20.55	20.55	99.65	14.06	14.06	84.42	6.82	$6.4x10^{15}$
9 (Sep)	96.38	18.64	18.64	97.83	12.13	12.13	80.36	7.61	$8.1x10^{15}$
10 (Oct)	99.22	15.82	15.82	99.30	13.25	13.25	79.57	6.51	$5.9x10^{17}$
11 (Nov)	99.73	11.65	11.65	99.82	10.23	10.23	78.29	4.96	$2.6x10^{11}$
12 (Dec)	92.29	12.71	12.71	93.30	11.01	11.01	53.85	4.70	$8.3x10^{17}$
mean	96.68	15.97	15.97	97.44	11.86	11.86	69.69	6.16	$1.2x10^{17}$
st.dev.	2.76	3.10	3.10	3.29	2.24	2.24	9.45	1.29	$2.6x10^{17}$

Table 4 PIs constructed by LUBEX with CFS, MI and PA for the British data

Case	CFS			MI			PA		
Study	PICP	PINAW	CWC	PICP	PINAW	CWC	PICP	PINA W	CWC
1 (Jan)	96.71	22.81	22.81	98.26	16.80	16.80	48.02	7.12	$1.6x10^{11}$
2 (Feb)	86.93	13.21	182.44	96.83	10.63	10.63	3.66	5.88	$3.5x10^{19}$
3 (Mar)	95.58	15.72	15.72	93.83	11.69	11.69	59.74	6.73	$6.8x10^{8}$
4 (Apr)	97.65	16.35	16.35	98.19	12.88	12.88	54.93	7.29	$3.2x10^{11}$
5 (May)	90.52	20.23	20.23	96.09	10.58	10.58	62.94	6.31	$1.0x10^{8}$
6 (Jun)	99.10	14.07	14.07	99.10	8.65	8.65	81.35	4.33	$3.4x10^{3}$
7 (Jul)	93.51	16.88	16.88	99.57	11.75	11.75	69.09	4.76	$5.0x10^{5}$
8 (Aug)	72.17	11.00	$1.3x10^{5}$	83.13	8.86	$5.2x10^{2}$	33.83	3.52	$8.2x10^{12}$
9 (Sep)	88.42	10.95	43.95	73.57	8.08	$1.5x10^{5}$	25.97	5.40	$4.5x10^{14}$
10 (Oct)	91.86	16.82	21.87	98.00	9.58	9.58	67.19	5.21	$2.4x10^{6}$
11 (Nov)	81.90	14.68	$2.4x10^{3}$	92.40	9.28	9.28	31.22	3.69	$4.4x10^{13}$
12 (Dec)	65.48	19.25	$1.3x10^{8}$	73.74	14.72	$1.6x10^{5}$	26.93	6.42	$7.4x10^{15}$
mean	88.32	16.00	$1.1x10^{7}$	91.89	11.13	$2.6x10^{4}$	47.07	5.55	$2.9x10^{18}$
st.dev.	9.98	3.42	$3.6x10^{7}$	9.20	2.51	$5.9x10^{4}$	21.80	1.23	$9.7x10^{18}$

value of five runs of LUBEX, i.e. we created and evaluated an NN ensemble five times for each case study to further reduce the variability of the NN performance due to the random initialization of weights and random perturbation of weights during training, that are part of the simulated annealing algorithm. We discuss the variability of the PICP results over the five runs in the last subsection. All reported results in this section are results on the testing data.

6.1 Coverage Probability

We first examine the coverage probability PICP. Fig. 5 shows the PICP results from Tables 3 and 4 for both datasets and all case studies.

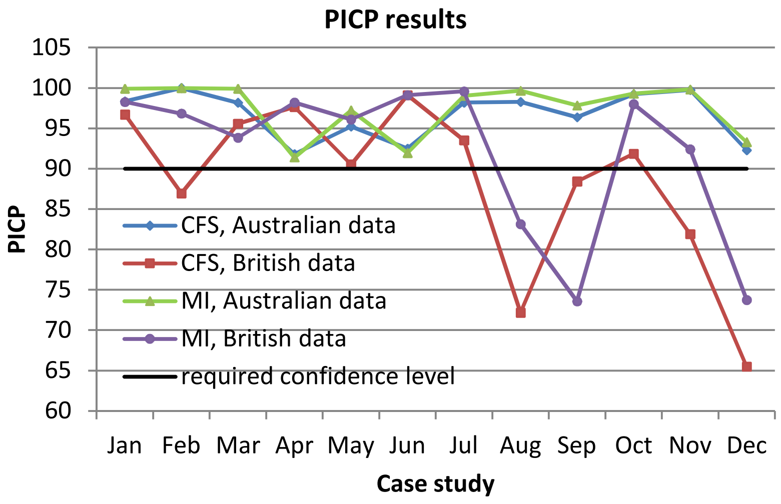


Fig. 5 Comparison of coverage probability (PICP) using CFS and MI. (PICP results using PA are not shown as they are disproportionally lower).

For the Australian data (see Table 3 and Fig. 5), we can see that PICP for CFS and MI is higher than the prescribed confidence level (μ=90%) for all case studies, i.e. the constructed PIs are valid. The average PICP over all case studies and the standard deviation are: 96.68±2.76% for CFS and 97.44±3.29% for MI; hence, the constructed PIs considerably outperformed μ. For both CFS and MI, nine out of twelve cases achieve PICP greater than 95% and the remaining three case studies (4, 6 and 12) have PICP between 91.40% and 93.30%. Overall, the PICP is higher when using MI than CFS. In contrast to CFS and MI, the PICP for PA is lower than the prescribed confidence level for all case studies. The average PICP for PA is 69.69±9.45%, which is considerably lower than 90%. In summary, for the Australian data, LUBEX with CFS and MI as feature selection methods was able to construct PIs with high coverage probability while LUBEX with PA failed to do this.

The results for the British data, as shown in Table 4 and Fig. 5, are less accurate. PICP for CFS satisfies the prescribed 90% confidence level for 7 out of 12 cases and PICP for MI satisfies this level for 9 out of the 12 cases. The average PICP over all case studies and the standard deviation are: PICP=88.32±9.98% for CFS and PICP=91.89±9.20% for MI; i.e. slightly below or above the 90% level, with relatively high standard deviation. For both CFS and MI, the results for August, September and December are not satisfactory. These months include several holidays - summer, school and Christmas, and might also include sudden weather changes. It is possible that the electricity demand was more irregular and difficult to predict due to these special days; we plan to investigate this in future work. Similarly to the Australian data, PA is the worst performing feature selector. It is underperforming for all cases, achieving average PICP of 47.07±21.80%, which is substantially lower than 90%. This indicates that the constructed PIs are unreliable and invalid. In summary, for the British data, LUBEX with CFS and MI as feature selectors was able to construct PIs with the required coverage probability in about 70% of the cases, while LUBEX with PA was unsuccessful in all cases.

6.2 Interval Width

We now examine the interval width PINAW. Fig. 6 presents the PINAW results from Tables 3 and 4 for visual comparison of the three feature selectors.

For the Australian data, we can see than LUBEX generates narrow PIs with all feature selection methods. The average widths are: 15.97±3.10 for CFS, 11.86±2.24 for MI and 6.16±1.29 for PA. Although the PI width for PA is the narrowest, the coverage probability of these intervals is unacceptable. We can also directly compare the performance of CFS and MI for case studies 2, 10 and 11as the corresponding coverage probabilities are very similar. We can see that the interval widths for MI are smaller than those for CFS. This indicates lower uncertainty for LUBEX with MI than LUBEX with MI for these case studies.

We can draw similar conclusions for the British data - LUBEX generates narrow PIs with all feature selection methods. The average widths are: 16.00±3.42 for CFS, 11.13±2.51 for MI and 5.55±1.23 for PA. Again, the PI width for PA is the narrowest but the coverage probability of these intervals is below the 90% confidence level, therefore these intervals are invalid, regardless of their width. We can compare the performance of CFS and MI for case study 6; the coverage probability is the same (99.10%, the highest for both feature selectors) and MI generates a narrower interval than CFS. Thus, for this case study, LUBEX with MI has lower uncertainly than LUBEX with CFS.

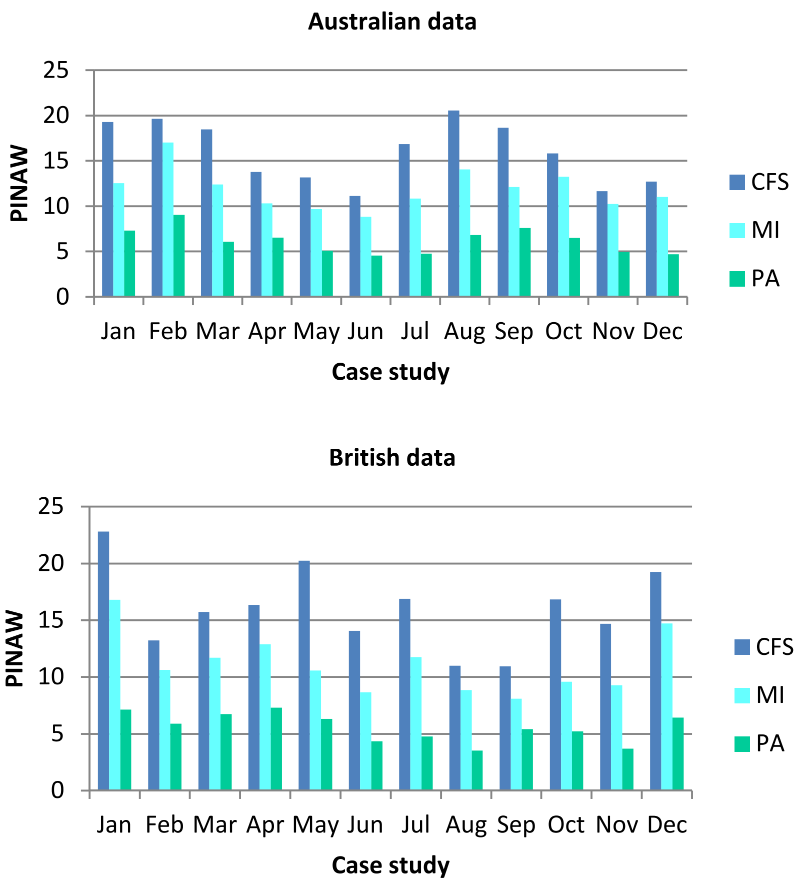


Fig. 6 Comparison of interval widths (PINAW)

6.3 Coverage Width Criterion

We finally examine the CWC values. In all cases where the coverage probability is higher than the prescribed confidence value, CWC=PINAW. This is true for CFS and MI for all 12 months for the Australian data and for the majority of months for the British data. In contrast, the CWC values for PA are very high for both datasets. This is also as expected and follows from the definition of CWC – when the PIs fail to satisfy the required minimum coverage probability, they are invalid regardless of their width; this results in high CWC values as CWC includes a heavy penalty for invalid PIs.

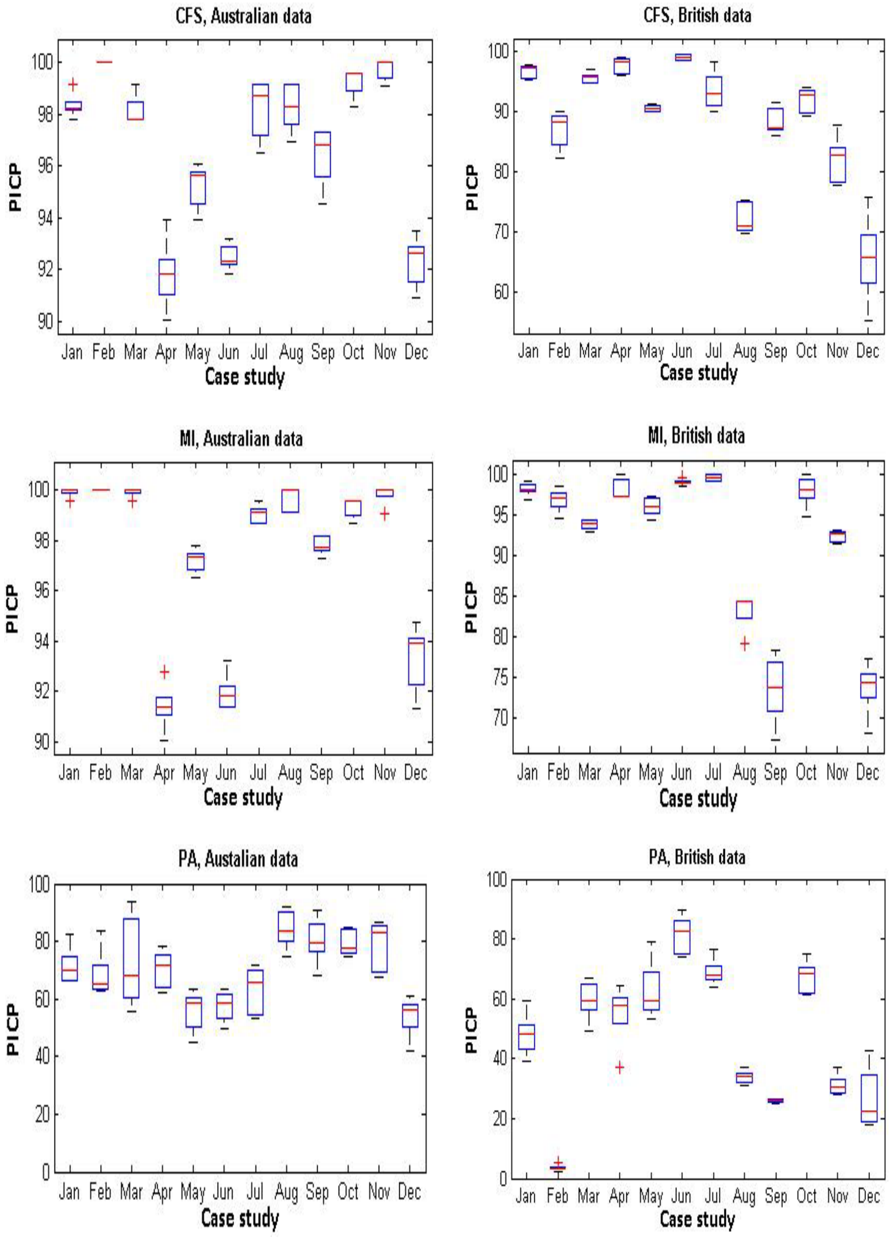


Fig. 7 Variability of PICP over the five runs for LUBEX with CFS, MI and PA

6.4 Variability of PICP

We also investigate the variability of the coverage probability of the generated PIs over the five runs of LUBEX for each case study. Fig. 7 shows box plots of PICP for LUBEX with CFS, MI and PA, respectively, for the five runs of each case study, for the two datasets. The mark in the middle of a box is the median value, the edges of the box are the 25th and 75th percentiles and the whiskers are the minimum and maximum values that are not considered outliers. Values that are considered outliers are indicated with a cross. When comparing the results in Fig. 7, it is important to note the different scales on the *y* axes.

We can see that the PICP variation over the five runs is high for LUBEX with PA, for both datasets; the PICP standard deviations for the 12 months is in the range of 4.76-16.07 for the Australian data and 0.69-10.33 for the British data. On the other hand, the PICP variation is very small for LUBEX with CFS and MI for the Australian data – the PICP standard deviation is in the range of 0-1.39 for CFS and 0-1.36 for MI, for the 12 case studies. For the British data, this variation is moderate: 0.45-7.36 for CFS and 0.32-4.22 for MI, and is lower for the months satisfying the prescribed confidence level and higher for the underperforming months. By comparing CFS and MI, we can also see that overall MI is less variable than CFS.

We conclude that LUBEX with CFS and MI generated PIs that were relatively stable over multiple runs while LUBEX with PA generated highly unstable PIs.

7 Conclusion

In this paper we considered the task of constructing NN-based PIs for electricity demand forecasting. We extended the interval forecasting method LUBEX by studying the effect of three feature selection methods on the quality of the generated PIs: the traditional PA and the less popular in time series analysis CFS and MI methods. We conducted an evaluation using half-hourly electricity demand data for one year, for two countries: Australia and the United Kingdom.

Our results showed that CFS and MI were able to identify a small set of informative lag variables, that when used with LUBEX resulted in good quality PIs for forecasting of new data. For the Australian data, all PIs were valid as they satisfied the minimum coverage probability of 90% (MI: PICP=97.44±3.29%, CFS: PICP=96.68±2.76%) and they also showed little variation for multiple runs of LUBEX. For the British data, 70% of the constructed PIs were valid (MI: PICP=88.32±9.98%, CFS: PICP=91.89±9.20%) and these PIs were moderately stable for multiple runs of LUBEX. In contrast, LUBEX with PA produced PIs that were invalid and highly variable (PICP=69.69±9.45% for the Australian data and PICP=47.07±21.805% for the British data). In addition, CFS and MI selected a considerably smaller set of features in comparison to PA, 4-10 versus 39-68, which means faster training of the NN component and faster prediction for new instances. Overall, the best performance was achieved by LUBEX with MI for both datasets.

In future work, we plan to investigate the use of weather variables in addition to previous demand data in order to improve the quality of the generated prediction intervals. We will also study the effect of outlier days, e.g. days with extreme weather, days with sudden weather changes and holidays, and develop a method to deal with them.

Acknowledgement. Mashud Rana is supported by an Endeavour Postgraduate Award funded by the Australian government.

References

1. Chatfield, C.: Time-Series Forecasting. Chapman &Hall/CRC (2000)
2. Taylor, J.W.: Short-term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing. Journal of Operational Research Society 54, 799–805 (2003)
3. Taylor, J.W.: An Evaluation of Methods for Very Short-Term Load Forecasting Using Minite-by-Minute British Data. International Journal of Forecasting 24, 645–658 (2008)
4. Taylor, J.W.: Triple Seasonal Methods for Short-term Electricity Semand Forecasting. European Journal of Operational Research 204, 139–152 (2010)
5. Liu, K., Subbarayan, S., Shoults, R.R., Manry, M.T., Kwan, C., Lewis, F.L., Naccarino, J.: Comparison of Very Short-term Load Forecasting Techniques. IEEE Transactions on Power Systems 11, 877–882 (1996)
6. Fan, S., Hyndman, R.J.: Short-term Load Forecasting Based on a Semi-parametric Additive Model. IEEE Transactions on Power Systems 27, 134–141 (2012)
7. Sood, R., Koprinska, I., Agelidis, V.G.: Electricity Load Forecasting Based on Autocorrelation Analysis. In: International Joint Conference on Neural Networks (IJCNN). IEEE Press, Barcelona (2010)
8. Koprinska, I., Rana, M., Agelidis, V.G.: Yearly and Seasonal Models for Electricity Load Forecasting. In: International Joint Conference on Neural Networks (IJCNN), pp. 1474–1481. IEEE Press, San Jose (2011)
9. Shamsollahi, P., Cheung, K.W., Chen, Q., Germain, E.H.: A Neural Network Based Very Short Term Load Forecaster for the Interim ISO New England Electricity Market System. In: 22nd IEEE PES International Conference on Power Industry Computer Applications (PICA), pp. 217–222 (2001)
10. Charytoniuk, W., Chen, M.-S.: Very Short-term Load Forecasting Using Artificial Neutal Networks. IEEE Transactions on Power Systems 15, 263–268 (2000)
11. Rana, M., Koprinska, I., Troncoso, A.: Forecasting Hourly Electricity Load Profile Using Neural Networks. In: International Joint Conference on Neural Networks (IJCNN). IEEE Press, Beijing (2014)
12. Chen, Y., Luh, P.B., Guan, C., Zhao, Y., Michel, L.D., Coolbeth, M.A.: Short-Term Load Forecasting: Similar Day-based Wavelet Neural Network. IEEE Transactions on Power Systems 25, 322–330 (2010)
13. Khosravi, A., Nahavandi, S., Creigton, D., Atiya, F.: Lower Upper Bound Estimation Method for Construction of Neural Network-Based Prediction Intervals. IEEE Transactions on Neural Networks 22(3), 337–346 (2011)
14. Hwang, J.T.G., Ding, A.A.: Prediction Intervals for Artificial Neural Networks. Journal of the American Statistical Association 92(438), 2377–2387 (1997)

15. MacKay, D.J.C.: The Evidence Framework Applied to Classification Networks. Neural Computation 4(5), 720–736 (1992)
16. Heskes, T.: Practical Confidence and Prediction Intervals. In: Mozer, T.P.M., Jordan, M. (eds.) Neural Information Processing Systems. MIT Press (1997)
17. Rana, M., Koprinska, I., Khosravi, A., Agelidis, V.G.: Prediction Intervals for Electricity Load Forecasting Using Neural Networks. In: International Joint Conference on Neural Networks (IJCNN). IEEE Press, Dallas (2013)
18. Rana, M., Koprinska, I., Khosravi, A.: Feature Selection for Neural Network-Based Interval Forecasting of Electricity Demand Data. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 389–396. Springer, Heidelberg (2013)
19. AEMO (2013), http://www.aemo.com.au
20. National Grid UK (2013), http://www2.nationalgrid.com/uk/
21. Taylor, J.W.: Short-term load forecasting with exponentially weighted methods. IEEE Transactions on Power Systems 27, 458–464 (2012)
22. Hall, M.A.: Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In: Int. Conference on Machine Learning (ICML), pp. 359–366 (2000)
23. Kraskov, A., Stögbauer, H., Grassberger, P.: Estimating Mutual Information. Physical Review E 69 (2004)

Stacked Denoising Auto-Encoders for Short-Term Time Series Forecasting

Pablo Romeu, Francisco Zamora-Martínez, Paloma Botella-Rocamora, and Juan Pardo

Abstract. In this chapter, a study of deep learning of time-series forecasting techniques is presented. Using Stacked Denoising Auto-Encoders, it is possible to disentangle complex characteristics in time series data. The effects of complete and partial fine-tuning are shown. SDAE prove to be able to train deeper models, and consequently to learn more complex characteristics in the data. Hence, these models are able to generalize better. Pre-trained models show a better generalization when used without covariates. The learned weights show to be sparse, suggesting future exploration and research lines.

1 Introduction

Time series forecasting is the task of predicting some future values of a given sequence, using historical data from the same signal (univariate forecasting), or using historical data from several correlated signals (multivariate forecasting). In the literature, different models has been described, each one with its particular characteristics. Exponential smoothing [14], Autoregressive Integrated Moving Average (ARIMA) models [6], or Artificial Neural Networks (ANNs) [12] are examples of the most important models used for time series forecasting. Usually, statistical methods are used to estimate the weight parameters of these models. ANNs have been widely applied to this task [35, 2, 34], normally trained by one of the different variants of Gradient Descent (GD) algorithm.

Pablo Romeu
Embedded Systems and Artificial Intelligence Group,
Escuela Superior de Enseñanzas Técnicas, Universidad CEU Cardenal Herrera,
C/ San Bartolomé 46115 Alfara del Patriarca, Valencia, Spain
e-mail: `pablo.romeu@uch.ceu.es`

Francisco Zamora-Martínez · Paloma Botella-Rocamora · Juan Pardo
Universidad CEU Cardenal Herrera
e-mail: {`francisco.zamora,paloma.botella,juan.pardo`}`@uch.ceu.es`

© Springer International Publishing Switzerland 2015
P. Koprinkova-Hristova et al. (eds.), *Artificial Neural Networks*,
Springer Series in Bio-/Neuroinformatics 4, DOI: 10.1007/978-3-319-09903-3_23

Recently, deep learning has been defined as a new area in machine learning field, for the study of new learning techniques and paradigms which allow to train models with many number of layers, usually deep ANNs, but also deep Restricted Boltzmann Machines (RBMs) among others. Deep architectures, with many levels of non-linearity, theoretically can represent complex features of its inputs [31] with more flexibility than shallow architectures. However, training of deep models is difficult because of the highly non-convex surface of the training criterion used in GD algorithm [31, 15, 10]. Depending on the initialization of the model parameters, the algorithm converges to different local minima. In the case of deep models this problem has been shown as an even more important [11]. Different ideas have been developed to allow the training of deep models, being the unsupervised greedy layer-wise pre-training [18] one of the most successful in computer vision problems. Simpler non-linearities, as the rectified linear units [16] or maxout networks [17], are shown to be useful to reduce the impact of gradient vanishing in deep models. Stacking many convolutional layers, instead of a fully connected layers, as it is done in convolutional ANNs [22], is another example of successful ideas in deep learning. In fact, convolutional ANNs are being used traditionally with many layers showing good training convergence [22].

Deep learning is being depth studied in computer vision and natural language processing tasks [28, 36, 9], allowing to train complex classifiers which learn to extract interesting features from the input data [32]. In time series forecasting literature is difficult to find works following such ideas. As far as we know, only few works are available [7, 20], principally using RBMs as proposed by [18]. Stacked denoising auto-encoders (SDAEs) [32] have been found useful and simpler, because they are based on standard GD algorithms and almost any ANN toolkit can deal with them. This paper is an extension of a previous work [27] where minor differences has been observed comparing standard ANNs with deep ANNs pre-trained with SDAEs for time series forecasting. A new set of experiments, with different and larger data, besides an analysis and discussion of the learned models, comparing deep and shallow architectures, has been done for this extension.

Two tasks, indoor temperature forecasting, and electric power consumption forecasting, were selected for this study, because of its interest for future study of energy efficiency in domotic environments [13].

2 Time Series Forecasting

A time series is a collection of numerical data recorded over a period of time, usually occurring in uniform intervals. Time series can be formalized as a sequence of scalars from a variable x observed as output of a process of interest:

$$\bar{s}(x) = s_0(x), s_1(x), \ldots, s_{i-1}(x), s_i(x), s_{i+1}(x), \ldots \tag{1}$$

A fragment in the time series from the time point at position i until time point at position j is denoted by $s_i^j(x)$. Without loss of generality, $s_i(x_0)$ is denoted as s_i when no confusion is possible and only when it refers to the interesting variable x_0.

Time series forecasting methods predict the interest variable x_0 using past values of x_0 (*univariate forecasting*). Some of these methods allow to incorporate also additional information of other related covariates in the process (*multivariate forecasting*).

Univariate forecasting methods predict the interest variable x_0 using only past values of x_0. This kind of methods are focused in understanding the underlying structure composed, mainly, for trend and pattern repetition through time. This past behavior learned is extrapolated into the future.

Multivariate forecasting methods use to predict the interest variable x_0 not only past values of x_0, but also additional values of a number C of variables $(x_1,x_2,...,x_C)$ related with the predicted variable x_0. Multivariate approaches performs better than univariate when additional variables $(x_1,x_2,...,x_C)$ are related with the predicted variable x_0.

2.1 Forecast Models

A forecast model could be formalized as a function F which receive as inputs the interest variable (x_0) with its past values until current time t, and a number C of covariates $(x_1,x_2,\ldots,x_C)$ with also its past values until current time t, and produces a future window of size H for the given x_0 variable:

$$\langle \hat{s}_{t+1}(x_0), \hat{s}_{t+2}(x_0), \ldots, \hat{s}_{t+H}(x_0) \rangle = F(\Omega^t(x_0), \Omega^t(x_1), \ldots, \Omega^t(x_C)), \tag{2}$$

being $\Omega^t(x) = s^t_{t-I(x)+1}(x)$ the $I(x)$ past values of variable/covariate x. The sum of all the input sizes will be $m = \sum_x I(x)$.

Different parameters are important in the process of estimation of forecasting models: the number of past values $(I(x_0), I(x_1), ..., I(x_C))$, the size of the future window (H), and the position in the future of the prediction (future horizon).

There are several possibilities to classify forecasting methods based on the size of future window and how it is produced [2]. Depending on the size of future window, forecasting methods could be grouped as *single-step-ahead forecasting methods*, if the model forecasts only the next time step ($H = 1$), and *multi-step-ahead forecasting methods*, if the method forecasts a future window of size H, where $H > 1$.

At the same time multi-step-ahead forecasting methods can be grouped into *multi-step-ahead iterative forecasting* and *multi-step-ahead direct forecasting* [8]. On multi-step-ahead *iterative* forecasting methods the model forecasts only the next time step, and the future window is forecasted by an iterative process. At first iteration model uses past data $(\Omega^t(x_0), \Omega^t(x_1), \ldots, \Omega^t(x_C))$ to forecast x_0 at time $t+1$ $(\hat{s}_{t+1}(x_0))$. At second iteration, same model forecasts $\hat{s}_{t+2}(x_0)$ using past data for covariates $(\Omega^{t+1}(x_1), \ldots, \Omega^{t+1}(x_C))$ besides the $\hat{s}_{t+1}(x_0)$ predicted at previous step:

$$\Omega^{t+1}(x_0) = \langle s_{t-I(x_0)+2}(x_0), s_{t-I(x_0)+3}(x_0), ..., s_t(x_0), \hat{s}_{t+1}(x_0) \rangle \tag{3}$$

And thus for the rest of future time steps. On the other hand, on multi-step-ahead *direct* forecasting the model forecasts directly in one step, a large future window of size H. Following this forecasting approach, the most used type of modelling are *Pure direct strategy* and *Multiple Input Multiple Output* (MIMO). Pure direct strategy approach uses H different forecasting models with same inputs, one for each future time step, to forecasting the complete future window.

$$\begin{aligned}
\hat{s}_{t+1}(x_0) &= F_1(\Omega^t(x_0), \Omega^t(x_1), \dots, \Omega^t(x_C)) \\
\hat{s}_{t+2}(x_0) &= F_2(\Omega^t(x_0), \Omega^t(x_1), \dots, \Omega^t(x_C)) \\
&\dots \\
\hat{s}_{t+H}(x_0) &= F_H(\Omega^t(x_0), \Omega^t(x_1), \dots, \Omega^t(x_C))
\end{aligned}$$

However MIMO approach uses one unique model to compute the full H future window given its inputs, following Equation 2. There are several advantages in this approach due to the join learning of inputs and outputs because the model learns the stochastic dependency between predicted values. Discriminative models, as *Artificial Neural Networks* (ANNs), gain a big profit of this input/output mapping.

2.2 Evaluation Measures

Performance of forecasting methods over one time series could be assessed by several different evaluation functions that measure the empirical error of the model. In this work, for a deeper analysis of the results, three different error functions were used: Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE). In every time step t the error is computed comparing target values for the time series $s_{t+1}, s_{t+2}, \dots, s_{t+H}$ and its corresponding time series prediction $\hat{s}_{t+1}, \hat{s}_{t+2}, \dots, \hat{s}_{t+H}$ using the model θ:

$$\mathrm{MAE}(\theta, t) = \frac{1}{H} \sum_{h=1}^{H} |\hat{s}_{t+h} - s_{t+h}| \tag{4}$$

$$\mathrm{MSE}(\theta, t) = \frac{1}{2} \sum_{h=1}^{H} (\hat{s}_{t+h} - s_{t+h})^2 \tag{5}$$

$$\mathrm{RMSE}(\theta, t) = \sqrt{\frac{1}{H} \sum_{h=1}^{H} (\hat{s}_{t+h} - s_{t+h})^2} \tag{6}$$

The results could be measured over all time series (one of each time step t) in a given dataset $\mathscr{D}$ as:

$$L^{\star}(\theta, \mathscr{D}) = \frac{1}{|\mathscr{D}|} \sum_{t=1}^{|\mathscr{D}|} L(\theta, t), \tag{7}$$

being $|\mathscr{D}|$ the size of the data set, and $L = \{\mathrm{MAE}, \mathrm{MSE}, \mathrm{RMSE}\}$ the loss-function, defining $\mathrm{MAE}^{\star}$, $\mathrm{MSE}^{\star}$, $\mathrm{RMSE}^{\star}$.

3 Unsupervised Greedy Layer-Wise Pre-training

Deep learning research has shown the importance of weight initialization when training deep ANNs [21]. Greedy layer-wise pre-training [18] is the first approach which has been found useful to reduce initial weights sensitivity. The greedy algorithm follows an iterative procedure that trains one layer at a time. Thus, during the training of one layer, the gradient computation is focused only in one hidden layer, avoiding deep non-linear compositions. Every layer is trained in order to model the value representations (distribution) produced by the previous layer. With every stacked layer, a new distribution is obtained, which could be used as input for a new layer. In this way, it is easy to build a deep ANN, with many non-linearities.

3.1 Stacked Denoising Auto-Encoders

An Auto-Encoder (AE) is a kind of ANN which is trained to reproduce as output the data given in its input. The hidden layer of AEs is able to learn anonymous latent variables [3], which are a representation of the observed data x in a different coordinates space. Principal Component Analysis (PCA) or Independent Component Analysis (ICA) are also procedures that allow to extract latent variables, following a similar approach. An AE ($r(\cdot)$ function) computes a reconstructed version of its input x following:

$$r(x) = g(f(x)) \tag{8}$$

$$f(x) = \phi_f(b_f + \mathbf{W}x) \tag{9}$$

$$g(x) = \phi_g(b_g + \mathbf{W}^T f(x)) \tag{10}$$

being $\phi_f(\cdot)$ a non-linear activation function, $\phi_g(\cdot)$ a linear or non-linear activation function, x a column vector of size m, W a weights matrix with h rows and m columns, b_f is a bias column vector with h rows, and b_g a bias column vector with m rows. Each row of the matrix W is a different transformation of the input data (or neuron) and correspond to the hidden latent variables. Functions $f(\cdot)$ and $g(\cdot)$ are known as encoding and decoding functions. Using a number of latent variables $h < m$ (bottleneck AEs), the model is regularized in the same way as PCA does. However, when $h > m$ the model could learn the identity function, learning useless latent variables. The model is trained in order to reduce the reconstruction error, e.g., it minimizes $||r(x) - x||^2$.

In recent approaches to this issue, Stacked Denoising Auto-Encoders (SDAEs) [11, 32] have been proposed, where previous formalization is slightly modified, introducing small perturbations (noise) at the input of every layer during pre-training phase. Therefore, the DAE is trained to recover a clean (denoised) version of its noisy input. Stacking multiple layers of DAEs it is possible to obtain useful high-level features, and lower error reconstruction, therefore better generalization is expected. Denoising goal is another way of regularization, that allows to train models where $h > m$, preventing the model to learn identity function. The DAE takes a noisy version $N(x)$ of the original input x, and it is trained to reduce the reconstruction error, e.g., as before, it minimizes $||r(N(x)) - x||^2$. Another ways to AE regularization are presented in the literature [26, 23], showing successful results in computer vision problems.

Models with $h > m$ has an overcomplete representation of the data. That situation is useful when the representation is also *sparse* [32]. Interest in sparse representations comes from the apparently evidence of overcomplete but sparse neural activity in the brain [25]. Other motivations for sparse representations include its easier interpretation by the subsequent classifier.

Different noise strategies are possible. This paper follows the use of additive Gaussian noise and masking noise. In this way, the noise function is as follows:

$$N(x) = \left(x + \mathcal{N}(0, \sigma^2)\right) .* B(m, 1-p) \tag{11}$$

where the Gaussian noise is sampled from a normal distribution with mean $\mu = 0$ and σ^2 variance (independent and equal for every input component), and the masking noise is sampled from a Binomial distribution with probability $1 - p$ of being 1 and probability p of being 0 (masked). The operator $.*$ indicates a component-wise multiplication.

In the literature, diverse activation functions are being used. This paper uses the linear activation function in the visible layer of first AE, because the indoor temperature and the electric power consumptions are not bounded. In the following layers (hidden and visible), softsign [5] or logistic functions are used:

$$softsign(x) = \frac{x}{1 + |x|} \tag{12}$$

$$logistic(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

3.2 Greedy Layer-Wise Pre-training Algorithm and Greedy Layer-Wise Hyper-Parameter Optimization

The SDAE training consists in two phases. At the first phase (*pre-training phase*) several auto-encoders (AEs) are trained, forcing each AE to reconstruct the encoding computed at the hidden layer of previous AE, except the first AE which is trained to reconstruct the input features of the task. SDAE [11, 32] introduces noise to the input data of the pre-training task as shown in Figure 1. Therefore, the pre-trained layers extract robust features from data while trying to reconstruct its inputs. Once all layers have been pre-trained, there is a second supervised phase (*fine-tuning phase*) where the output layer is added and the whole ANN is trained to solve a concrete task.

The hyper-parameter optimization is also done in a greedy way, that is, the learning rate, momentum, weight decay, ... hyper-parameters are optimized for one layer at a time. Once a layer hyper-parameter optimization ends, the best values are fixed

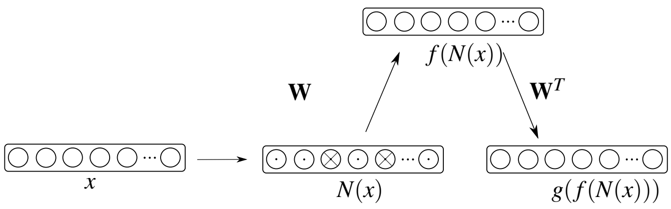


Fig. 1 Denoising Auto-Encoder with Gaussian (GN) and masking noisy (MN) inputs

for the training of the next layer, and so on. This optimization phase allows to reduce the computational effort, because ANNs with many hidden layers share the computation of layers until the last one. For instance, the ANN with one hidden layer uses the same first layer as the ANN with two hidden layers.

3.3 SDAEs for Time-Series: Expectations

The interest of deep ANNs for time-series comes from the widely use of shallow models (shallow ANNs and/or ARIMA models) in this field for complex and strongly non-linear series. This paper is an attempt to model this non-linearities with up to three hidden layers, expecting to find interesting dependencies between input/output data. In order to reduce the computational effort needed for the experimentation, input layer is constrained to the short-term past values of the observed value x. However, a deeper analysis could be interesting, in order to study if large input layers could lead to better learned features in the deep ANN. This extension of the input could be overcome by using convolutional ANNs in the first layers of the ANN.

From a machine learning perspective, it will be interesting to observe if deeper ANNs could learn temporal abstractions by using larger inputs. A trade-off between the input size and the addition of temporal and other covariates was expected to be found. The ability of a machine to learn this temporal dependencies without any other information but the observed time-series could be very interesting for the development of intelligent agents in partially observed environments.

4 Experimentation

Several experiments have been performed in order to study the effect of hyperparameters, pre-training technique and different fine-tuning schemes. Two different data sets of time series were employed to make the experiments. The first one is related to temperature forecasting and second one about electric power consumption. Next they are described with the corresponding experimental setup and results.

In the first task, the study is focused in different training schemes, in order to state which one is better for training deep ANNs. Regularization effect of SDAEs pre-training was also studied. In the second task, a deeper study of the differences between pre-trained and not pre-trained models was performed.

4.1 Temperature Forcasting

This first task studies the effect of pre-training with SDAEs in a smooth indoor temperature forecasting problem.

4.1.1 Data Description

This is the first data set that was employed for the experimentation. It was obtained from the Small Medium Large House (SMLhouse) research facility at the University CEU Cardenal Herrera. This data set is available at the UCI (Unviersity of California) Machine Learning Repository (`http://archive.ics.uci.edu/ml`) [1] under the name SML2010.

SMLhouse is a solar-powered house constructed to participate in the 2012 Solar Decathlon Europe competition [30]. The house has a monitoring system able to obtain a huge quantity of variables to study energy consumption, but for our study only some have been used in order to obtain the indoor temperature forecasting model.

The variables are [34]:

1. Indoor temperature in degrees Celsius. This is the interest forecasted variable.
2. Hour feature in Universal Time Coordinated (UTC), extracted from the timestamp of each pattern. The hour of the day is important to estimate the Sun position.
3. Sun irradiance in $W = m^2$. It is correlated with temperature because more irradiance will mean more heat.
4. Indoor relative humidity percentage. The humidity modifies the inertia of the temperature.
5. Indoor air quality in CO2 ppm (parts per million). The air quality is related to the number of persons in the house, and a higher number of persons mean an increase in temperature.
6. Raining Boolean status. The result of sub-sampling this variable is the proportion of minutes in sub-sampling period (15 minutes) where raining sensor was activated with True.

Input signal sequence is sampled at one minute period (indoor temperature time series task presented at [34]), and pre-processed by a low-pass filter to get the mean value of the current plus last 14 samples, introducing a delay of 7 minutes in the predicted values. Hence, $s'_1 s'_2 \dots s'_N$ are computed, where

$$s'_i = (s_i + s_{i-1} + s_{i-2} + s_{i-3} + \dots + s_{i-14})/15\,. \tag{14}$$

In a second step, differences are calculated between each two adjacent elements, to estimate the variation of temperature within 15 minutes. Then $s''_1, s''_2 \dots s''_{N-1}$ are calculated, being

$$s''_i = s'_i - s'_{i+1}\,. \tag{15}$$

Data is divided in three partitions: training (2016 training patterns, 21 days), validation (672 validation patterns, 7 days) used during training to avoid over-fitting, and the last one for testing (672 test patterns, 7 days). The validation partition is sequential with the training partition, but the test partition is one week ahead from them.

4.1.2 Experimentation Setup

Experiments using three type of training for each ANN have been performed. First mode, named Train Mode 0 and denoted (TM-0) consists in training the ANN model to forecast directly, without pre-training. Second mode, named Train Mode 1 (TM-1) pre-trains the ANN using SDAE and fine-tuning all the layers. Finally, third mode named Train Mode 2 (TM-2) pre-trains the ANN using SDAE and fine-tuning but not in all layers, only the last one (forecasting layer).

A combination of grid and random search for hyper-parameter optimization was used, instead of using a grid search simply. Some works [4] have demonstrated that random search is more efficient in finding good hyper-parameter configurations within less trials than grid search. Moreover, random search is also easier to parallelize. For all experiments, the Gaussian Noise variance is set to 0.01 and the length of future window forecasted is $H = 12$, whilst mini-batch size is set to 32 in all cases. To carry out the experiments, a grid with the following hyper-parameters was created:

- the train mode (TM-0, TM-1, TM-2)
- number of hidden layers (1, 2, 3)
- Mask noise percentage (0.02, 0.04, 0.10, 0.20)

For each grid node 100 random trials were performed sampling different values for the following hyper-parameters:

- *Input size* (I): uniform distribution (12, 24, 36, 48, 60, 72, 84, 96).
- *Learning rate*: two hyper-parameters, one for pre-train and other the fine-tuning, sampled uniformly and independently in the range $[10^{-3}, 10^{-2}]$.
- *Momentum*: one hyper-parameter for each phase, independently sampled $\sim \mathcal{N}(10^{-3}, 5\times10^{-3})$, avoiding negative values.
- *Weight decay*: in TM-0, weight decay was used at the ANN training. In TM-1 and TM-2, weight decay was used at the pre-training phase, but not at fine-tuning. Uniformly sampled in the range $[0, 10^{-5}]$.
- *Hidden layer sizes*: uniformly distributed at range $[4, 1024]$, but only multipliers of 4 were took (to avoid memory alignment issues).

Let $\bar{y} = \langle y_1, y_2, \dots, y_H\rangle = \langle \hat{s}_{j+1}, \hat{s}_{j+2}, \dots, \hat{s}_{j+H}\rangle$ be the forecasted values at time j, $\bar{t} = \langle t_1, t_2, \dots, t_H\rangle = \langle s_{j+1}, s_{j+2}, \dots, s_{j+H}\rangle$ the ground truth values at time j and $\bar{w} = \langle w_1, w_2, \dots, w_n\rangle$ the ANN weights excluding biases. For this train experiments, a L2 regularization term ε has been used, and hence, the loss function is computed as:

$$L = \sum_i (y_i - t_i)^2 + \frac{\varepsilon}{2} \sum_i (w_i^2) \tag{16}$$

The number of iterations used over the training have been a minimum of 50 and maximum of 4 000 iterations. Training stops if ever, at iteration k, the best validation performance was observed before iteration $k/2$. In total 3 600 experiments were performed, all of them using the `April-ANN`[1] toolkit [33], that implements SDAE and efficient ANN training algorithms.

4.1.3 Results

First, as expected, it should be noted that in deeper networks the training mode TM-0 were difficult to train. For three layered ANNs about 58% of experiments achieve MAE* greater than 0.5, and the 33% for two layered ANNs. This behavior has not been found for TM-1 and TM-2 experiments.

Results of validation MAE* have been analized and are shown in the Figure 2 for different hyper-parameters: input size, encoding layer size, mask noise percentage and learning rate at fine-tuning phase. Regarding the input size behavior for each training mode, as can be observed in Figure 2-a, shows worst results for TM-2 and better (and similar) for TM-0 and TM-1 training modes. For TM-0 and TM-1 modes, best results are between 48 and 60 input sizes (12–15 hours). This result is coherent with the input signal observed frequency, where 12 hours past info seems to be enough to forecast the slope of the function, and therefore, to restrict next forecasted values to a short range.

The lack of full fine-tuning harms the performance of TM-2 as hidden layers are added. Encoding layer (the number of neurons at the last hidden layer) results at Figure 2-b show consistently that rising the number of hidden layers introduces instability to the prediction results on TM-2. TM-0 and TM-1 remain more stable while increasing the number of hidden layers. Figure 2-c shows that masking noise harms the performance of TM-0. Comparing learning rates at fine-tuning phase, shown at Figure 2-d, it is observed that TM-0 needed a higher learning rate to achieve good results, while for pre-trained TM-1 it was not an important parameter. All four figures show that as more hidden layers were added, the more unstable TM-2 was.

Test results in MAE* and RMSE* for the systems which perform better in validation are shown at Figure 3. The systems were optimized separately following a random search algorithm of [4]. The best system topologies, regarding to validation set performance, were: for TM-0 60 inputs and two hidden layers of 756 and 60; for TM-1 48 inputs and three hidden layers of 648, 920 and 16; and for TM-2 96 inputs and one hidden layer of 712. Non pre-trained ANNs TM-0 achieved similar errors as pre-trained ANNs TM-1.

[1] Developed by members of our research group in collaboration with members of Universitat Politècnica de València.

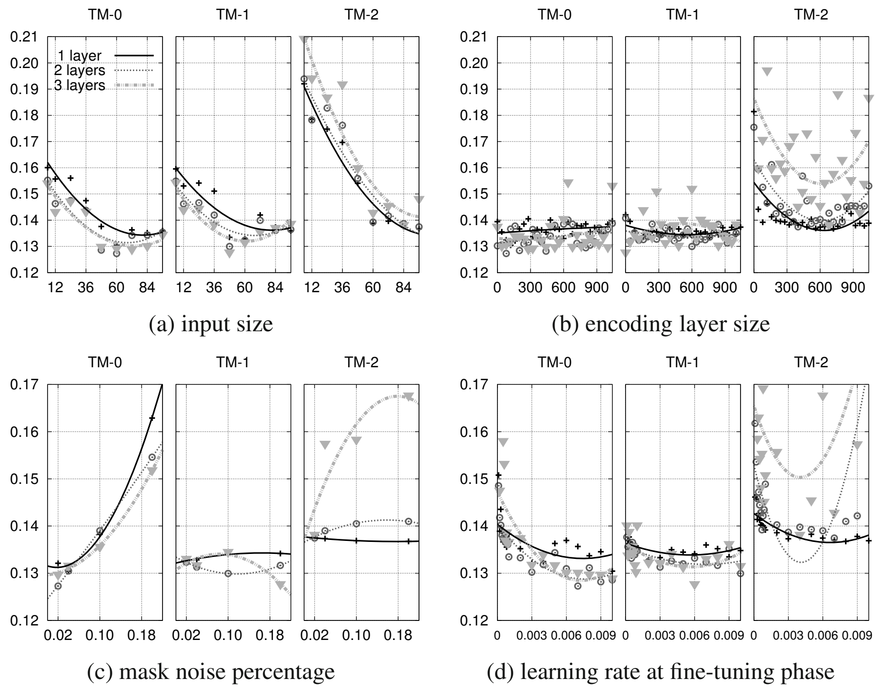


Fig. 2 Result plots of different hyper-parameters (*x*-axis) vs MAE* (*y*-axis). Only the best model for each *x* value is represented. Second order polynomial fits are also shown.

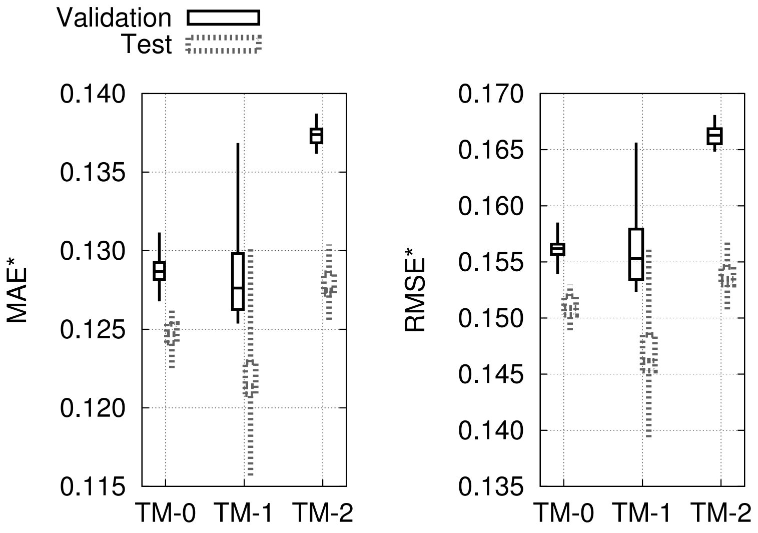


Fig. 3 Box whiskers plot computed over 20 different random initializations for each training mode (TM) and for the validation and test sets

Table 1 Mean and standard deviation for MAE* & RMSE* computed over the same 20 random initializations. Bolded numbers are best results. ETS is an exponential smoothing model.

MAE*

	Validation $(\mu \pm \sigma)$	Test $(\mu \pm \sigma)$
ETS	0.3004	0.3254
TM-0	$\mathbf{0.1289 \pm 0.0011}$	0.12482 ± 0.0010
TM-1	$\mathbf{0.1287 \pm 0.0033}$	$\mathbf{0.1223 \pm 0.0033}$
TM-2	0.1374 ± 0.0007	0.1279 ± 0.0011

RMSE*

	Validation $(\mu \pm \sigma)$	Test $(\mu \pm \sigma)$
ETS	0.3648	0.3930
TM-0	$\mathbf{0.1563 \pm 0.0011}$	0.1511 ± 0.0012
TM-1	$\mathbf{0.1565 \pm 0.0040}$	$\mathbf{0.1473 \pm 0.0039}$
TM-2	0.1663 ± 0.0009	0.1538 ± 0.0013

However, performance measured at test error is better at full pre-trained ANNs TM-1. Pre-trained ANNs TM-2 achieve worse error measures than TM-0 and TM-1. In order to compare, an exponential smoothing model [29] was trained, denoted as *ETS* at Table 1.

Finally, learning curve and test set generalization of the best configurations are shown at Figure 4. Left plot shows that as training epochs increase, TM-2 stops learning at a certain epoch, while TM-0 and TM-1 keep improving. Right plot shows that TM-0 ANN over-fits if it is trained during too much epochs, while pre-trained networks remain close to its minimum error, showing the benefits of pre-training as a regularization method.

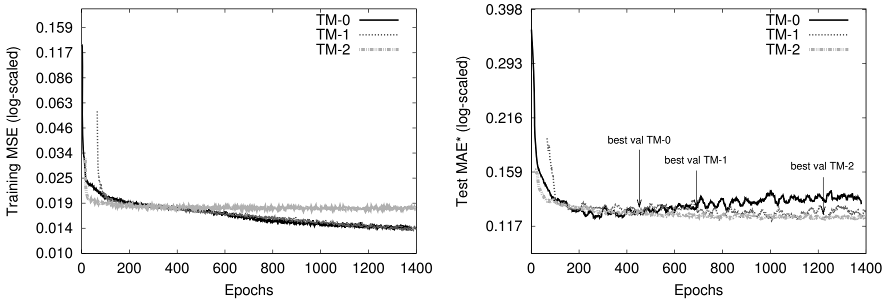


Fig. 4 Training detail for the best initialization (from the 20 random initializations tested for Figure 3). (Left) Plot of MSE at each training epoch for best ANN of each training mode. (Right) Plot of MAE* at each training epoch for best ANN of each training mode. Arrows indicates stopping point due to validation stopping criteria.

4.1.4 Conclusions

In this section, three training models have been studied: a non pre-trained model TM-0 and two pre-trained models TM-1 and TM-2. Results show that pre-trained models with a fine-tuning phase –TM-1– are able to train deep models and generalize better than non pre-trained models. However, TM-1 obtained result is not overwhelming compared to deep learning improvements in other tasks [32], but it is a promising preliminary result. The low dimensionality of the task (univariate time-series forecasting using at most 96 inputs), and the smoothness of the indoor temperature time-series, reduce the benefit of using SDAE models, as was stated in [32].

4.2 Electric Power Consumption Forecasting

In this section, other household power consumption dataset is presented. It will be studied how pre-trained and non pre-trained networks behave when learning sparse data, and the effects of covariates in these models.

4.2.1 Data Description

This second data set was also obtained from the UCI (University of California) Machine Learning Repository (`http://archive.ics.uci.edu/ml`) [1]. It is an individual household electric power consumption data set. It contains measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. Different electrical quantities and some sub-metering values are also available. The data comes from EDF R&D centre in Clamart, France.

The archive contains 2 075 259 measurements gathered between December 2006 and November 2010 (47 months). There are nine attributes that recollect the next information:

1. date: Date in format dd/mm/yyyy
2. time: time in format hh:mm:ss
3. global-active-power: household global minute-averaged active power (in kilowatt)
4. global-reactive-power: household global minute-averaged reactive power (in kilowatt)
5. voltage: minute-averaged voltage (in volt)
6. global-intensity: household global minute-averaged current intensity (in ampere)
7. sub-metering-1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
8. sub-metering-2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.

9. sub-metering-3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

To take into account the data set contains some missing values in the measurements (nearly 1,25% of the rows). All calendar timestamps are present in the data set but for some timestamps, the measurement values are missing: a missing value is represented by the absence of value between two consecutive semi-colon attribute separators. For instance, the data set shows missing values on April 28, 2007.

Anyway, for the present work, after doing a correlation study of the different attributes, it was considered to work with the month, hour, global-active-power attributes and a the day of week (where Sunday is codified as 1). Global-reactive-power, global-intensity and voltage would be possible to be introduced as covariates, because they are not really correlated between themselves. However, for simplicity, and to focus the study in one forecasting variable and its stationality, it was decided to ignore this data. Figure 5 shows the distribution of global-active-power depending within temporal variables (month, hour and day of week).

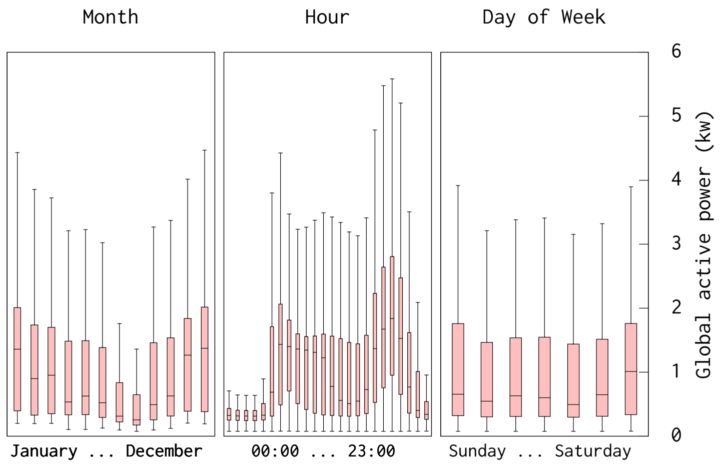


Fig. 5 Box-plot with the distribution of global-active-power within the temporal covariates

In this case, the electric active power time-series was sampled with one minute period. In order to reduce high frequency noise, it was smoothed with a low-pass filters of 5 samples. In a second step, the data was normalized to zero-mean and one-variance.

The null values in the original data were ignored. In order to simplify the problem, original data were splitted into 71 files, containing each one a bunch of sequential data.

The last six files in chronological order were reserved. The remaining files have been divided in three random partitions[2]: training (1 350 935 minutes, approximately

[2] Every file is taken as a whole, and pertains only to one of the three partitions.

2 years plus 208 days), validation (258 402 minutes, approximately 179 days) used during training to avoid over-fitting, a test set 1 with (181 641 minutes, approximately 126 days). The test set 2 contains 218 465 minutes, approximately 151 days, which corresponds to the last six files (from 2010/06/12 at 17:36 to 2010/11/26 at 21:02).

4.2.2 Experimentation Setup

In order to pre-train each layer of the ANN, a greedy algorithm was used. For each layer, as in section 4.2, over 140 experiments using random search hyper-parameter optimization were performed. Once the pre-training of a layer is over, the best hyper-parameters combination is used to generate a filter for the next layer's pre-training task. These steps are followed for each pre-trained layer. Finally, a forecasting layer is trained using as input the full stack of pre-trained layers and injecting other co-variates to the final forecasting layers using a shortcut as shown in Figure 6. These shortcuts are used in order to add the covariates without the need of adding them at the pre-training. Also, time covariates in this task are highly correlated to the power load as shown in Figure 5. Formally the forward computation of an ANN with three layers and shortcuts follows:

$$\begin{aligned}\bar{y} = b^{(3)} + \mathbf{W}^{(3)}\phi(b^{(2)} + \mathbf{W}^{(2)}\phi(b^{(1)} + \mathbf{W}^{(1)}\Omega^{j}(x_0))) + \\ + \mathbf{W}^{(M)}[:, \text{month}] + \mathbf{W}^{(H)}[:, \text{hour}] + \mathbf{W}^{(D)}[:, \text{day of week}]\end{aligned}$$

being b the biases of the ANN, $\mathbf{W}$ the weights of the ANN, ϕ the activation function, $[:, \text{month}], [:, \text{hour}], [:, \text{day of week}]$ the column value of the weights of the covariates, $\bar{y} = \langle y_1, y_2, \ldots, y_H \rangle = \langle \hat{s}_{j+1}, \hat{s}_{j+2}, \ldots, \hat{s}_{j+H} \rangle$ being the forecasted values at time j, $\bar{t} = \langle t_1, t_2, \ldots, t_H \rangle = \langle s_{j+1}, s_{j+2}, \ldots, s_{j+H} \rangle$ the true future global-active-power values at time j.

These experiments additionally use the L1 regularization term. The L1 regularization (λ) is used within the loss function together with the L2 regularization (ε), also known as weight decay. The L1 regularization forces sparsity in the model, forcing some weights to be zero. The L2 regularization avoids overfitting by forcing weights to be close to zero. Let $\bar{w} = \langle w_1, w_2, \ldots, w_n \rangle$ be the ANN weights excluding biases. Then, the loss function is computed as:

$$L = \sum_i (y_i - t_i)^2 + \lambda \sum_i |w_i| + \frac{\varepsilon}{2} \sum_i (w_i^2) \tag{17}$$

Another L2 weight penalty has been added, in this case the L2 max-norm penalty used in [19], which ensures that the L2 norm of the incoming weights for each neuron at the end of a back-propagation iteration are below a preset value, otherwise, the weights are reduced proportionally to fit the limit.

A minimum of 100 and maximum of 400 iterations over the training data are used. Training stops if ever, at iteration k, the best validation performance was observed before iteration $k/2$.

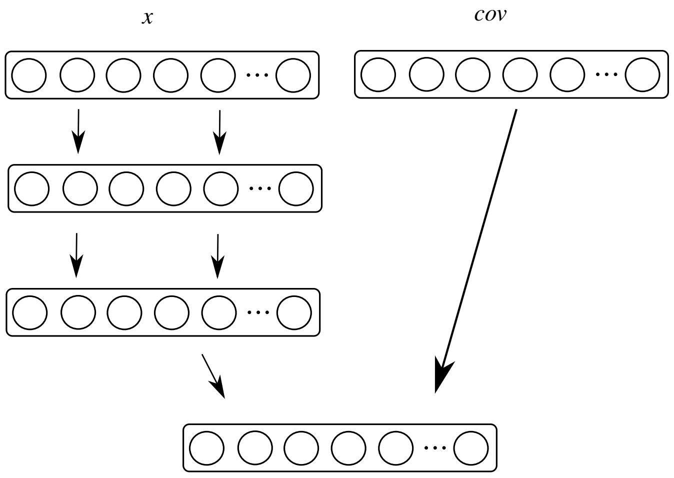


Fig. 6 Figure of covariates shortcut injected at the final forecasting layer

In this experiments, random search hyperparameter optimization [4] has been performed. 925 initial random experiments have been performed to choose an appropriate input size. These experiments are shown in figure 7. The best input size has been chosen from these experiments, which was 1440 inputs, which performed slightly better than the 2160 input size experiments.

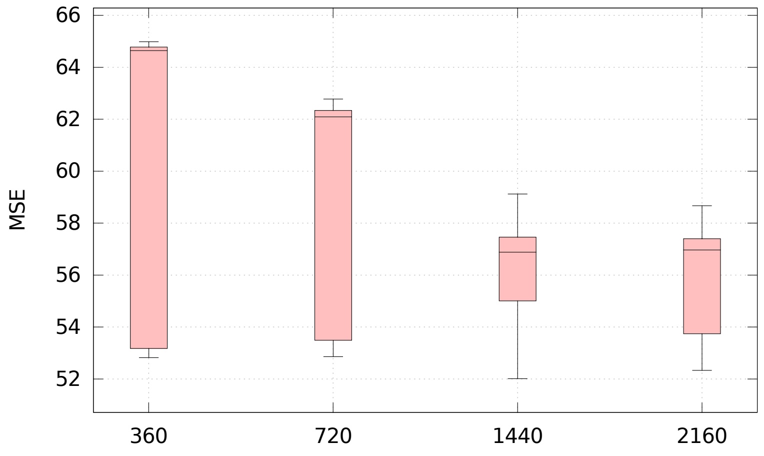


Fig. 7 MSE results box plot of 925 random initial tests to establish the best input size. The x-axis represents the values for the input size.

For the first layer pre-training task, the mini-batch size was set to 128. An input size of 1440 minutes of global-active-power was used. The activation function for the visible layer neurons was set to *linear* and *logistic* function was chosen for the hidden layer. Then, 177 random trials have been performed using following values for the randomized hyper-parameters:

- *Learning rate*: independently sampled using a log-uniform distribution within the range $[10^{-5}, 10^{-3}]$.
- *Momentum*: independently sampled using a log-uniform distribution within the range $[10^{-5}, 10^{-4}]$.
- *Weight decay*: Uniformly sampled within the range $[10^{-5}, 10^{-6}]$ in 10^{-6} steps.
- *Hidden layer sizes*: uniformly distributed within the range $[128, 2048]$, but only multipliers of 4 were taken (to avoid memory alignment issues).
- *L1 regularization*: Uniformly sampled within the values $\{0.0, 10^{-5}, 10^{-6}\}$.
- *Max norm penalty*: Uniformly sampled within the values $\{3, 4, 5, 6\}$.
- *Gaussian noise*: with zero mean and variance uniformly sampled within the values $\{0.0, 0.05, 0.1, 0.2\}$.
- *Mask noise*: uniformly sampled within the values $\{0.0, 0.05, 0.1, 0.2\}$.

The next layers' pre-training task fixed settings are both activation functions, visible and hidden, set to logistic. 148 second layer experiments and 334 third layer experiments were performed. The random values for the experiments of the next layers differences were because the first layer learning rate is set to a lower value due to the input linear function which does not converge if learning rate is set to a high value. Therefore, next layers learning rate and momentum are set to:

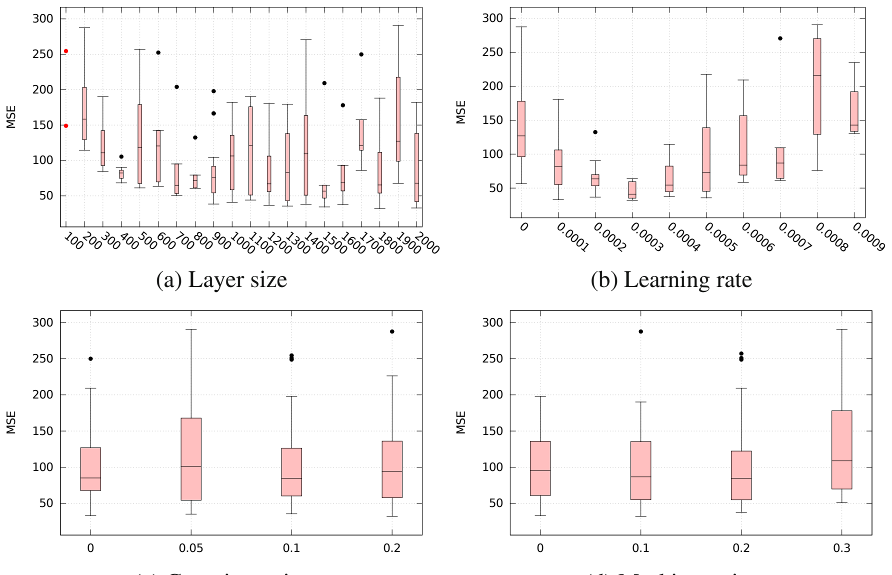
- *Learning rate*: independently sampled using a log-uniform distribution within the range $[10^{-3}, 10^{-2}]$.
- *Momentum*: independently sampled using a log-uniform distribution within the range $[10^{-4}, 10^{-3}]$.

All experiments were performed using the `April-ANN` toolkit [33], that implements SDAE and efficient ANN training algorithms.

4.2.3 Results

First layer pre-training results are shown in Figure 8. Learning rate and layer size are rounded to the closest decimal because the intrinsic randomness of the experiments makes difficult to find equal values to create boxplots. Layer size analysis suggests a first layer size close to 1500 and a learning rate of 0.0003. Gaussian noise and mask noise were selected close to 0.1 and 0.1 respectively and weight decay was chosen close to $1e-06$. momentum, L1 regularization and weight decay were chosen close to 10^{-3}, 10^{-6} and 10^{-5} respectively.

Using the first layer filter for encoding, second layer pre-training experiments have been performed. Results for this experiments are shown in Figure 9. These graphs suggest a learning rate of 0.004 and a layer size close to 1600. Also, Gaussian



(a) Layer size (b) Learning rate

(c) Gaussian noise (d) Masking noise

Fig. 8 Results for several hyper-parameters for first layer –h1– hyperparameter random search pre-training experiments

noise of 0.2 and masking noise of 0.1 will be selected. Momentum, L1 regularization and weight decay were chosen close to 10^{-4}, 10^{-6} and 10^{-6} respectively.

A third layer was also pre-trained, using the second layer filter for encoding its inputs. The experiments' results are shown at Figure 10 indicates a layer size close to 1700 and a learning rate between 0.004 and 0.006. Gaussian noise variance of 0.05 and mask noise of 0.1 was chosen. Momentum, L1 regularization and weight decay were chosen close to 10^{-4}, 10^{-6} and 10^{-6} respectively.

Once all layers have been pre-trained, the best autoencoder stack has been chosen. For the first layer, a size of 1792 neurons was chosen. For the second layer, a layer with 1596 was chosen. The last layer size was 1764. For this three layer filter, hyperparameter optimization experiments where performed with and without covariates. Non-pretrained ANNs were trained to compare their results with the pre-trained models. Experiments where performed with and without covariates. In all experiments where covariates are present, shortcuts have been used as explained at the previous section.

Analyzing the results, differences at first layer filters were detected. As shown in Figure 11, non-pretrained networks show consistently more noise than pre-trained weights. That is, weights in a pre-trained layer achieve some specialization in recognizing some few patterns while remaining nearly inactive in other patterns. Also, as this is a time series, closest point in time which is the value that the lasts neurons receive (rightmost part of the figure) usually has a high positive or negative value. The non pre-trained model results that uses covariates shown at Figure 11.b, show

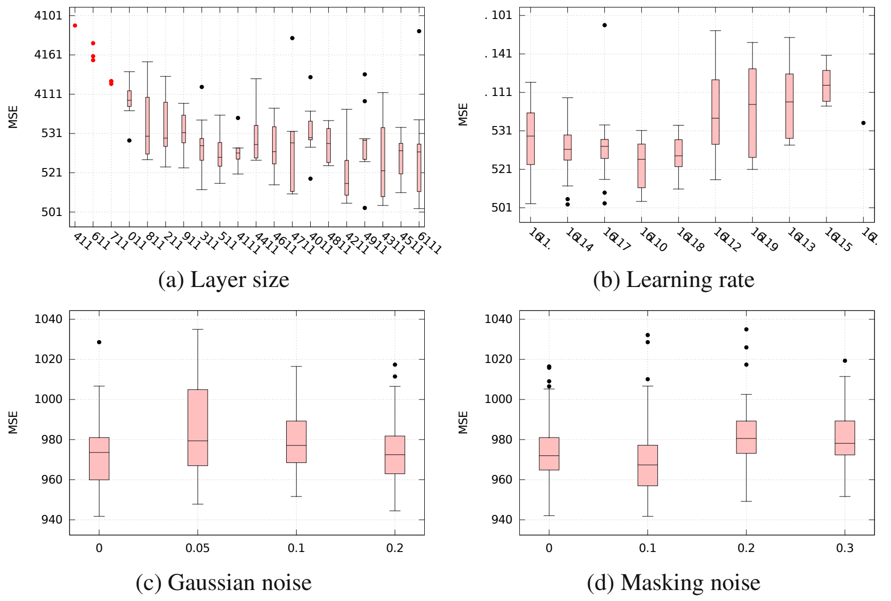

(a) Layer size
(b) Learning rate
(c) Gaussian noise
(d) Masking noise

Fig. 9 Results for several hyper-parameters for second layer –h2– hyperparameter random search pre-training experiments

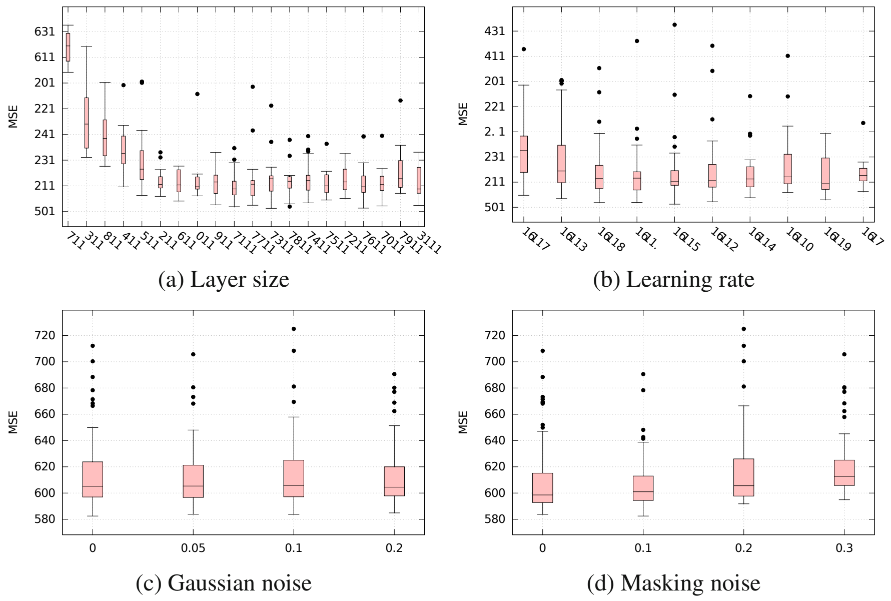
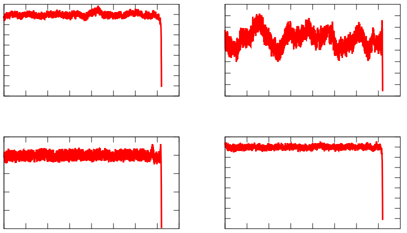
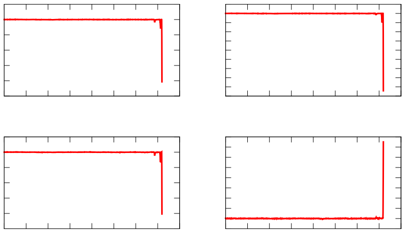
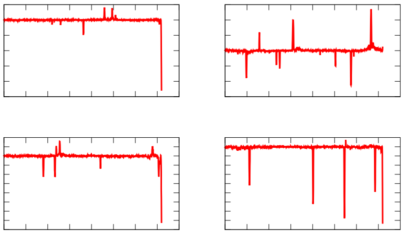

(a) Layer size
(b) Learning rate
(c) Gaussian noise
(d) Masking noise

Fig. 10 Results for several hyper-parameters for second layer –h3– hyperparameter random search pre-training experiments


(a) Non pre-trained weights


(b) Non pre-trained weights using covariates


(c) Pre-trained weights

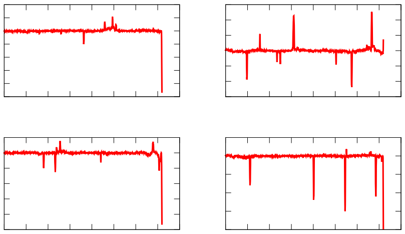

(d) Pre-trained weights using covariates

Fig. 11 First layer weights plot for single neurons in training experiments not using covariates and not pre-trained (a), using covariates and not pre-trained (b) vs pre-trained (c) and pre-trained using covariates (d)

an almost zero value in all its weights but the closest in time –rightmost part of the figure–. This effect is produced by the time covariates shortcut in the model, which gives more weight to the time covariates than the previous load values.

MSE error on the validation set was used to evaluate the best candidate to be used for the test sets. Validation results shown at Table 2 show similar performance for both approaches. Improvement differences for the pre-trained models versus the non pre-trained are shown.

Test results for each test set are shown at Table 3. Results have been divided into two main groups depending whether the test experiment was performed using covariates or not. For each test result, MAE*, MSE* and RMSE* for pre-trained and

Table 2 MAE*, MSE* and RMSE* for Validation Set using pre trained and non pre-trained networks with and without shortcuts. Percentage differences show the improvement of the pre-trained over the non pre-trained.

Pretraining	**No Covariates**			**With Covariates**		
	MAE*	*MSE**	*RMSE**	*MAE**	*MSE**	*RMSE**
No	0.5558	59.24	0.6928	0.5291	55.84	0.6713
Yes	0.5436	58.70	0.6866	0.5219	55.85	0.6706
Difference	2.23%	0.92%	0.91%	1.37%	−0.02%	0.21%

non pre-trained network are shown and the best for each one is remarked in bold. Pre-trained and non pre-trained experiments perform better in the test set 2 which was built with sequential data than in test set 1 which was built using random ordered data. Also, pre-trained experiments consistently improve the MAE* of the non pre-trained ones and more significantly when no other information (no covariates) but the global-active-power is present.

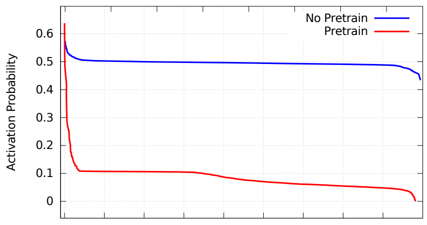
Table 3 MAE*, MSE* and RMSE* for test set 1 and test set 2 using pre trained and non pre-trained networks with and without shortcuts. Percentage differences show the improvement of the pre-trained over the non pre-trained.

test set	**Pretraining**	**No Covariates**			**With Covariates**		
		MAE*	*MSE**	*RMSE**	*MAE**	*MSE**	*RMSE**
test set 1	No	0.5808	65.88	0.7189	0.5638	**62.32**	0.7031
	Yes	**0.5660**	**65.39**	**0.7097**	**0.5555**	62.73	**0.7012**
	Difference	*2.58%*	*0.75%*	*1.28%*	*1.49%*	−0.66%	*0.27%*
test set 2	No	0.4769	41.72	0.5856	0.4596	**40.49**	**0.5770**
	Yes	**0.4584**	**41.13**	**0.5750**	**0.4570**	41.24	0.5819
	Difference	*3.95%*	*1.43%*	*1.82%*	*0.56%*	−1.83%	−0.84%

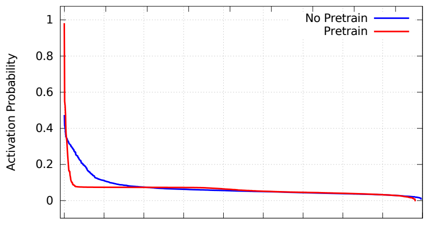
Sparsity of the last layer was analyzed implementing the same analysis used in [24]. This analysis computes the probability for each neuron to be active as its mean activation value, and then shows it in decreasing order. Figure 12 provides some insight of how sparse are the pre-trained, and non pre-trained networks. In this figure, activation probabilities for each hidden unit in the last hidden layer are plotted. The units are sorted in decreasing order of activation probability. The effect of pre-training yields the network to have sparse activations. The effect of adding injected covariates issues the same effect on non-pretrained networks, but Figure 12-(a) and (c) show that the pre-trained network non-activations (leftmost part of the figure) has more probability than the no pre-trained neurons ones. Hence, pre-trained ANNs show more sparsity for both test sets.

4.2.4 Conclusion

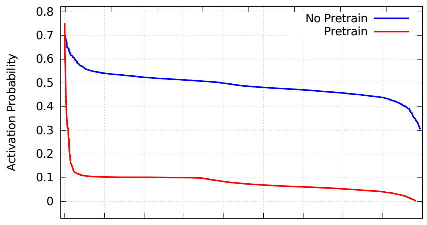
In this section a comparison with models that use covariates has been done. Although results using covariates in non pre-trained models are similar pre-trained models without covariates. The weights visualization and the activations probability analysis show that pre-trained models behave sparse in all cases, but non pre-trained models are very noisy when no covariates are present and sparse with covariates. However, the addition of covariates to non pre-trained models induces a first layer with zero weights in all positions except the last one. This behavior suggests that non pre-trained models are working in linearly with covariates, ignoring the rest of input data.



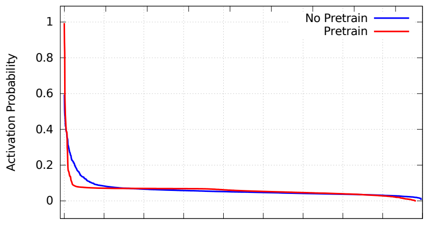
(a) Test set 1, no covariates



(b) Test set 1, covariates



(c) Test set 2, no covariates



(d) Test set 2, with covariates

Fig. 12 Last layer neuron output activation probabilities computation. Figures (a) and (b) represent test set 1. Figures (c) and (d) represent test set 2.

5 Discussion and Further Work

In this chapter, a study of deep learning of time-series forecasting techniques has been studied. Using Stacked Denoising Auto-Encoders, it is possible to disentangle complex characteristics in time series data. The effects of complete and partial fine-tuning have been studied. SDAE have proved to be able to train deeper models, and consequently to learn more complex characteristics in the data. Hence, these models are able to generalize better, as shown in the test sets. Furthermore, they are useful to learn sparse overcomplete representations of data and achieving similar results than non pre-trained models with covariates.

Further work should be done improving the pre-training process and the final fine-tuning. Also, other approaches mixing and comparing posterior injection using shortcuts presented in this chapter and adding the covariates at the input of the model should be studied. Differences between non pre-trained and pre-trained models are clear in the proposed tasks. It will be possible to study the way to exploit this differences in order to improve the deep ANN results.

Acknowledgements. This work has been supported by Banco Santander and CEU Cardenal Herrera University through the project Santander-PRCEU-UCH07/12.

References

1. Bache, K., Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
2. Ben Taieb, S., Bontempi, G., Atiya, A., Sorjamaa, A.: A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. Expert Systems with Applications (2012) (preprint)
3. Bengio, Y.: Deep learning of representations: looking forward. In: Dediu, A.-H., Martín-Vide, C., Mitkov, R., Truthe, B. (eds.) SLSP 2013. LNCS (LNAI), vol. 7978, pp. 1–37. Springer, Heidelberg (2013)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305 (2012)
5. Bergstra, J., Desjardins, G., Lamblin, P., Bengio, Y.: Quadratic polynomials learn better image features. Tech. Rep. 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal (2009)
6. Brockwell, P.J., Davis, R.A.: Introduction to Time Series and Forecasting, 2nd edn. Springer (2002)
7. Chao, J., Shen, F., Zhao, J.: Forecasting exchange rate with deep belief networks. In: The 2011 International Joint Conference on Neural Networks (IJCNN), pp. 1259–1266 (2011)
8. Cheng, H., Tan, P.-N., Gao, J., Scripps, J.: Multistep-ahead time series prediction. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 765–774. Springer, Heidelberg (2006)
9. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning, ICML 2008, pp. 160–167 (2008), doi:10.1145/1390156.1390177
10. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11, 625–660 (2010)
11. Erhan, D., Manzagol, P.A., Bengio, Y., Bengio, S., Vincent, P.: The difficulty of training deep architectures and the effect of unsupervised pre-training. Journal of Machine Learning Research 5, 153–160 (2009)
12. Escrivá-Escrivá, G., Álvarez-Bel, C., Roldán-Blay, C., Alcázar-Ortega, M.: New artificial neural network prediction method for electrical consumption forecasting based on building end-uses. Energy and Buildings 43(11), 3112–3119 (2011)
13. Ferreira, P., Ruano, A., Silva, S., Conceição, E.: Neural networks based predictive control for thermal comfort and energy savings in public buildings. Energy and Buildings 55, 238–251 (2012)
14. Gardner, J.: Exponential smoothing: The state of the art. Journal of Forecasting 4(1) (1985)
15. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research 9, 249–256 (2010)
16. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011 (2011)
17. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout Networks. ArXiv e-prints (2013)
18. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006)

19. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
20. Kuremoto, T., Kimura, S., Kobayashi, K., Obayashi, M.: Time Series Forecasting Using Restricted Boltzmann Machine. In: ICIC, pp. 17–22 (2012)
21. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th International Conference on Machine learning, ICML 2007, pp. 473–480. ACM, New York (2007), http://doi.acm.org/10.1145/1273496.1273556, doi:10.1145/1273496.1273556
22. Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. 1(4), 541–551 (1989)
23. Lee, H., Ekanadham, C., Ng, A.Y.: Sparse deep belief net model for visual area v2. In: NIPS (2007)
24. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the ICML (2013)
25. Olshausen, B.A., Fieldt, D.J.: Sparse coding with an overcomplete basis set: a strategy employed by v1. Vision Research 37, 3311–3325 (1997)
26. Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y.: Contractive auto-encoders: Explicit invariance during feature extraction. In: ICML, pp. 833–840 (2011)
27. Romeu, P., Zamora-Martínez, F., Botella-Rocamora, P., Pardo, J.: Time-Series Forecasting of Indoor Temperature Using Pre-trained Deep Neural Networks. In: Mladenov, V., Koprinkova-Hristova, P., Palm, G., Villa, A.E.P., Appollini, B., Kasabov, N. (eds.) ICANN 2013. LNCS, vol. 8131, pp. 451–458. Springer, Heidelberg (2013)
28. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1631–1642 (2013)
29. Taylor, J.: Exponential smoothing with a damped multiplicative trend. International Journal of Forecasting 19, 715–725 (2003)
30. United States Department of Energy: Solar Decathlon Europe Competition (2012), http://www.solardecathlon.gov
31. Utgoff, P.E., Stracuzzi, D.J.: Many-layered learning. Neural Comput. 14(10), 2497–2529 (2002), http://dx.doi.org/10.1162/08997660260293319, doi:10.1162/08997660260293319
32. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. 11, 3371–3408 (2010)
33. Zamora-Martínez, F., et al.: April-ANN toolkit, A Pattern Recognizer In Lua, Artificial Neural Networks module (2013), https://github.com/pakozm/april-ann
34. Zamora-Martnez, F., Romeu, P., Botella-Rocamora, P., Pardo, J.: Towards Energy Efficiency: Forecasting Indoor Temperature via Multivariate Analysis. Energies 6(9), 4639–4659 (2013), http://www.mdpi.com/1996-1073/6/9/4639, doi:10.3390/en6094639
35. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting 14(1), 35–62 (1998)
36. Zou, W.Y., Socher, R., Cer, D.M., Manning, C.D.: Bilingual word embeddings for phrase-based machine translation. In: EMNLP, pp. 1393–1398 (2013)

Author Index