Studies in Computational Intelligence 557

Taras Kowaliw
Nicolas Bredeche
René Doursat *Editors*

Growing Adaptive Machines

Combining Development and Learning in Artificial Neural Networks

Springer

Studies in Computational Intelligence

Volume 557

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

For further volumes:
http://www.springer.com/series/7092

About this Series

The series "Studies in Computational Intelligence" (SCI) publishes new developments and advances in the various areas of computational intelligence—quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life sciences, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems, and hybrid intelligent systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

Taras Kowaliw · Nicolas Bredeche
René Doursat
Editors

Growing Adaptive Machines

Combining Development and Learning in Artificial Neural Networks

Springer

Editors
Taras Kowaliw
Institut des Systèmes Complexes de Paris Île-de-France
CNRS
Paris
France

René Doursat
School of Biomedical Engineering
Drexel University
Philadelphia, PA
USA

Nicolas Bredeche
Institute of Intelligent Systems and Robotics
CNRS UMR 7222
Université Pierre et Marie Curie
Paris
France

ISSN 1860-949X ISSN 1860-9503 (electronic)
ISBN 978-3-642-55336-3 ISBN 978-3-642-55337-0 (eBook)
DOI 10.1007/978-3-642-55337-0
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014941221

Printed on acid-free paper

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

Preface

It is our conviction that the *means of construction of artificial neural network topologies* is an important area of research. The value of such models is potentially vast. From an applied viewpoint, identifying the appropriate design mechanisms would make it possible to address scalability and complexity issues, which are recognized as major concerns transversal to several communities. From a fundamental viewpoint, the important features behind complex network design are yet to be fully understood, even as partial knowledge becomes available, but scattered within different communities.

Unfortunately, this endeavour is split among different, often disparate domains. We started a workshop in the hope that there was significant room for sharing and collaboration between these researchers. Our response to this perceived need was to gather like-motivated researchers into one place to present both novel work and summaries of research portfolio.

It was under this banner that we originally organized the DevLeaNN workshop, which took place at the Complex Systems Institute in Paris in October 2011. We were fortunate enough to attract several notable speakers and co-authors: H. Berry, C. Dimitrakakis, S. Doncieux, A. Dutech, A. Fontana, B. Girard, Y. Jin, M. Joachimczak, J. F. Miller, J.-B. Mouret, C. Ollion, H. Paugam-Moisy, T. Pinville, S. Rebecchi, P. Tonelli, T. Trappenberg, J. Triesch, Y. Sandamirskaya, M. Sebag, B. Wróbel, and P. Zheng. The proceedings of the original workshop are available online, at http://www.devleann.iscpif.fr. To capitalize on this grouping of like-minded researchers, we moved to create an expanded book. In many (but not all) cases, the workshop contribution is subsumed by an expanded chapter in this book.

In an effort to produce a more complete volume, we invited several additional researchers to write chapters as well. These are: J. A. Bednar, Y. Bengio, D. B. D'Ambrosio, J. Gauci, and K. O. Stanley. The introduction chapter was also co-authored with us by S. Chevallier.

Our gratitude goes to our program committee, without whom the original workshop would not have been possible: W. Banzhaf, H. Berry, S. Doncieux, K. Downing, N. García-Pedrajas, Md. M. Islam, C. Linster, T. Menezes, J. F. Miller, J.-M. Montanier, J.-B. Mouret, C. E. Myers, C. Ollion, T. Pinville, S. Risi, D. Standage, P. Tonelli. Our further thanks to the ISC-PIF, the CNRS, and to M. Kowaliw for help with the editing process. Our workshop was made possible via a grant from the Région Île-de-France.

Enjoy!

Toronto, Canada, January 2014 — Taras Kowaliw
Paris, France — Nicolas Bredeche
Washington DC, USA — René Doursat

Contents

Chapter 1
Artificial Neurogenesis: An Introduction and Selective Review

Taras Kowaliw, Nicolas Bredeche, Sylvain Chevallier and René Doursat

Abstract In this introduction and review—like in the book which follows—we explore the hypothesis that adaptive growth is a means of producing brain-like machines. The emulation of neural development can incorporate desirable characteristics of natural neural systems into engineered designs. The introduction begins with a review of neural development and neural models. Next, artificial development—the use of a developmentally-inspired stage in engineering design—is introduced. Several strategies for performing this "meta-design" for artificial neural systems are reviewed. This work is divided into three main categories: bio-inspired representations; developmental systems; and epigenetic simulations. Several specific network biases and their benefits to neural network design are identified in these contexts. In particular, several recent studies show a strong synergy, sometimes interchangeability, between developmental and epigenetic processes—a topic that has remained largely under-explored in the literature.

T. Kowaliw (✉)
Institut des Systèmes Complexes - Paris Île-de-France, CNRS, Paris, France
e-mail: taras@kowaliw.ca

N. Bredeche
Sorbonne Universités, UPMC University Paris 06,
UMR 7222 ISIR,F-75005 Paris, France
e-mail: nicolas.bredeche@upmc.fr

N. Bredeche
CNRS, UMR 7222 ISIR,F-75005 Paris, France

S. Chevallier
Versailles Systems Engineering Laboratory (LISV), University of Versailles,
Velizy, France
e-mail: sylvain.chevallier@uvsq.fr

R. Doursat
School of Biomedical Engineering, Drexel University, Philadelphia, USA
e-mail: rene.doursat@drexel.edu

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_1,

This book is about growing adaptive machines. By this, we mean producing programs that generate neural networks, which, in turn, are capable of learning. We think this is possible because nature routinely does so. And despite the fact that animals—those multicellular organisms that possess a nervous system—are staggeringly complex, they develop from a relatively small set of instructions. Accordingly, our strategy concerns the simulation of biological development as a means of *generating*, in contrast to directly designing, machines that can learn. By creating abstractions of the growth process, we can explore their contribution to neural networks from the viewpoint of complex systems, which self-organize from relatively simple agents, and identify model choices that will help us generate functional and useful artefacts. This pursuit is highly interdisciplinary: it is inspired by, and overlaps with, computational neuroscience, systems biology, machine learning, complex systems science, and artificial life.

Through growing adaptive machines, our ambition is also to contribute to a radical reconception of engineering. We want to focus on the design of component-level behaviour from which higher-level intelligent machines can emerge. The success of this "meta-design" [63] endeavour will be measured by our capacity to generate new learning machines: machines that scale, machines that adapt to novel environments, in short, machines that exhibit the richness we encounter in animals, but presently eludes artificial systems.

This chapter and the book that it introduces are centred around developmental and learning neural networks. It is a timely topic considering the recent resurgence of the neural paradigm as a major representation formalism in many technological areas, such as computer vision, signal processing, and robotic controllers, together with rapid progress in the modelling and applications of complex systems and highly decentralized processes. Researchers generally establish a distinction between *structural* design, focusing on the network topology, and *synaptic* design, defining the weights of the connections in a network [278]. This book examines how one could create a biologically inspired network structure capable of synaptic training, and blend synaptic and structural processes to let functionally suitable networks self-organize. In so doing, the aim is to recreate some of the natural phenomena that have inspired this approach.

The present chapter is organized as follows: it begins with a broad description of neural systems and an overview of existing models in computational neuroscience. This is followed by a discussion of *artificial development* and *artificial neurogenesis* in general terms, with the objective of presenting an introduction and motivation for both. Finally, three high-level strategies related to artificial neurogenesis are explored: first, *bio-inspired representations*, where network organization is inspired by empirical studies and used as a template for network design; then, *developmental simulation*, where networks grow by a process simulating biological embryogenesis; finally, *epigenetic simulation*, where learning is used as the main step in the design of the network. The contributions gathered in this book are written by experts in the field and contain state-of-the-art descriptions of these domains, including reviews of original research. We summarize their work here and place it in the context of the meta-design of developmental learning machines.

1 The Brain and Its Models

1.1 Generating a Brain

Natural reproduction is, to date, the only one known way to generate true "intelligence". In humans, a mere six million (6×10^6) base pairs, of which the majority is not directly expressed, code for an organism of some hundred trillion (10^{14}) cells. Assuming that a great part of this genetic information concerns neural development and function [253], it gives us a rough estimate of a brain-to-genome "compression ratio". In the central nervous system of adult humans, which contains approximately 8.5×10^{10} neural cells and an equivalent number of non-neural (mostly glial) cells [8], this ratio would be of the order of 10^4. However, the mind is not equal to its neurons, but considered to emerge from the specific synaptic connections and transmission efficacies between neurons [234, 255]. Since a neural cell makes contacts with 10^3 other cells on average,[1] the number of connections in the brain reaches 10^{14}, raising our compression ratio to 10^8, a level beyond any of today's compression algorithms.

From there, one is tempted to infer that the brain is not as complex as it appears based solely on the number of its components, and even that something similar might be generated via a relatively simple parallel process. The brain's remarkable structural complexity is the result of several dynamical processes that have emerged over the course of evolution and are often categorized on four levels, based on their time scale and the mechanisms involved:

level	time scale	change
phylogenic	generations	genetic: randomly mutated genes propagate or perish with the success of their organisms
ontogenic	days to years	cellular: cells follow their genetic instructions, which make them divide, differentiate, or die
epigenetic	seconds to days	cellular, connective: cells respond to external stimuli, and behave differently depending on the environment; in neurons, these changes include contact modifications and cell death
inferential	milliseconds to seconds	connective, activation: neurons send electrical signals to their neighbours, generating reactions to stimuli

However, a strict separation between these levels is difficult in neural development and learning processes.[2] Any attempt to estimate the phenotype-to-genotype com-

[1] Further complicating this picture are recent results showing that these connections might themselves be information processing units, which would increase this estimation by several orders of magnitude [196].

[2] By epigenetic, we mean here any heritable and non-genetic changes in cellular expression. (The same term is also used in another context to refer strictly to DNA methylation and transcription-level mechanisms.) This includes processes such as learning for an animal, or growing toward a light source for a plant. The mentioned time scale represents a rough average over cellular responses to environmental stimuli.

pression ratio must also take into account epigenetic, not just genetic, information. More realistic or bio-inspired models of brain development will need to include models of environmental influences as well.

1.2 Neural Development

We briefly describe in this section the development of the human brain, noting that the general pattern is similar in most mammals, despite the fact that size and durations vastly differ. A few weeks after conception, a sheet of cells is formed along the dorsal side of the embryo. This neural plate is the source of all neural and glial cells in the future body. Later, this sheet closes and creates a neural tube whose anterior part develops into the brain, while the posterior part produces the spinal cord. Three bulges appear in the anterior part, eventually becoming the forebrain, midbrain, and hindbrain. A neural crest also forms on both sides of the neural tube, giving rise to the nervous cells outside of the brain, including the spinal cord. After approximately eight weeks, all these structures can be identified: for the next 13-months they grow in size at a fantastic rate, sometimes generating as many as 500,000 neurons per minute.

Between three to six months after birth, the number of neurons in a human reaches a peak. Nearly all of the neural cells used throughout the lifetime of the individual have been produced [69, 93]. Concurrently, they disappear at a rapid rate in various regions of the brain as programmed cell death (apoptosis) sets in. This overproduction of cells is thought to have evolved as a competitive strategy for the establishment of efficient connectivity in axonal outgrowth [34]. It is also regional: for instance, neural death comes later and is less significant in the cortex compared to the spinal cord, which loses a majority of its neurons before birth.

Despite this continual loss of neurons, the total brain mass keeps increasing rapidly until the age of three in humans, then more slowly until about 20. This second peak marks a reversal of the trend, as the brain now undergoes a gradual but steady loss of matter [53]. The primary cause of weight increase can be found in the connective structures: as the size of the neurons increase, so does their dendritic tree and glial support. Most dendritic growth is postnatal, but is not simply about adding more connections: the number of synapses across the whole brain also peaks at eight months of age. Rather, mass is added in a more selective manner through specific phases of neural, dendritic, and glial development.

These phenomena of maturation—neural, dendritic, and glial growth, combined with programmed cell death—do not occur uniformly across the brain, but regionally. This can be measured by the level of myelination, the insulation provided by glial cells that wrap themselves around the axons and greatly improve the propagation of membrane potential. Taken as an indication of more permanent connectivity, myelination reveals that maturation proceeds in the posterior-anterior direction: the

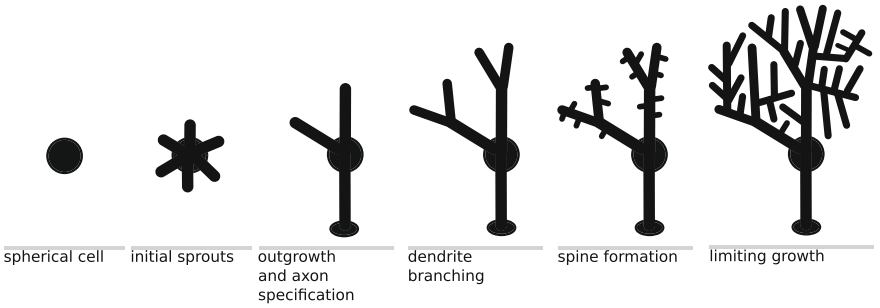

Fig. 1 Illustration of the general steps in neural dendritic development

spinal cord and brain stem (controlling vital bodily function) are generally mature at birth, the cerebellum and midbrain mature in the few months following birth, and after a couple of years the various parts of the forebrain also begin to mature. The first areas to be completed concern sensory processing, and the last ones are the higher-level "association areas" in the frontal cortex, which are the site of myelination and drastic reorganization until as late as 18-years old [69]. In fact, development in mammals never ends: dendritic growth, myelination, and selective cell death continue throughout the life of an individual, albeit at a reduced pace.

1.2.1 Neuronal Morphology

Neurons come in many types and shapes. The particular geometric configuration of a neural cell affects the connectivity patterns that it creates in a given brain region, including the density of synaptic contacts with other neurons and the direction of signal propagation. The shape of a neuron is determined by the outgrowth of *neurites*, an adaptive process steered by a combination of genetic instructions and environmental cues.

Although neurons can differ greatly, there are general steps in dendritic and axonal development that are common to many species. Initially, a neuron begins its life as a roughly spherical body. From there, neurites start sprouting, guided by *growth cones*. Elongation works by addition of material to relatively stable spines. Sprouts extend or retract, and one of them ultimately self-identifies as the cell's axon. Dendrites then continue to grow out, either from branching or from new dendritic spines that seem to pop up randomly along the membrane. Neurites stop developing, for example, when they have encountered a neighbouring cell or have reached a certain size. These general steps are illustrated in Fig. 1 [230, 251].

Dendritic growth is guided by several principles, generally thought to be controlled regionally: a cell's dendrites do not connect to other specific cells but, instead, are drawn to regions of the developing brain defined by diffusive signals. Axonal growth

tends to be more nuanced: some axons grow to a fixed distance in the direction of a simple gradient; others grow to long distances in a multistage process requiring a large number of guidance cells. While dendritic and axonal development is most active during early development, by no means does it end at maturity. The continual generation of dendritic spines plays a crucial role throughout the lifetime of an organism.

Experiments show that neurons isolated in cultures will regenerate neurites. It is also well known that various extracellular molecules can promote, inhibit, or otherwise bias neurite growth. In fact, there is evidence that in some cases context alone can be sufficient to trigger differentiation into specific neural types. For example, the introduction of catalysts can radically alter certain neuron morphologies to the point that they transform into other morphologies [230]. This has important consequences on any attempt to classify and model neural types [268].

In any case, the product of neural growth is a network possessing several key properties that are thought to be conducive to learning. It is an open question in neuroscience how much of neural organization is a result of genetic and epigenetic targeting, and how much is pure randomness. However, it is known that on the mesoscopic scale, seemingly random networks have consistent properties that are thought to be typical of effective networks. For instance, in several species, cortical axonal outgrowth can be modelled by a gamma distribution. Moreover, cortical structures in several species have properties such as relatively high clustering along certain axes, but not other axes [28, 146]. Cortical connectivity patterns are also "small-world" networks (with high local specialization, and minimal wiring lengths), which provide efficient long-range connections [263] and are probably a consequence of dense packing constraints inside a small space.

1.2.2 Neural Plasticity

There are also many forms of plasticity in a nervous system. While neural cell behaviour is clearly different during development and maturity (for instance, the drastic changes in programmed cell death), many of the same mechanisms are at play throughout the lifetime of the brain. The remaining differences between developmental and mature plasticity seem to be regulated by a variety of signals, especially in the extracellular matrix, which trigger the end of sensitive periods and a decrease in spine formation dynamics [230].

Originally, it was Hebb who postulated in 1949 what is now called *Hebbian learning*: repeated simultaneous activity (understood as mean-rate firing) between two neurons or assemblies of neurons reinforces the connections between them, further encouraging this co-activity. Since then, biologists have discovered a great variety of mechanisms governing synaptic plasticity in the brain, clearly establishing reciprocal causal relations between wiring patterns and firing patterns. For example, long-term potentiation (LTP) and long-term depression (LTD) refer to ositiveor negative

changes in the probability of successful signal transmission from a resynapticaction potential to the generation of a postsynaptic potential. These "long-term" changes can last for several minutes, but are generally less pronounced over hours or days [230]. Prior to synaptic efficacies, synaptogenesis itself can also be driven by activity-dependent mechanisms, as dendrites "seek out" appropriate partner axons in a process that can take as little as a few hours [310]. Other types of plasticity come from glial cells, which stabilize and accelerate the propagation of signals along mature axons (through myelination and extracellular regulation), and can also depend on activity [135].

Many others forms and functions of plasticity are known, or assumed, to exist. For instance, "fast synaptic plasticity", a type of versatile Hebbian learning on the 1-ms time scale, was posited by von der Malsburg [286–288]. Together with a neural code based on temporal correlations between units rather than individual firing rates, it provides a theoretical framework to solve the well-known "binding problem", the question of how the brain is able to compose sensory information into multi-feature concepts without losing relational information. In collaboration with Bienenstock and Doursat, this assumption led to a format of representation using graphs, and models of pattern recognition based on *graph matching* [19–21]. Similarly, "spike-timing dependent plasticity" (STDP) describes the dependence of transmission efficacies between connected neurons on the *ordering* of neural spikes. Among other effects, this allows for pre-synaptic spikes which precede post-synaptic spikes to have greater influence on the resulting efficacy of the connection, potentially capturing a notion of causality [183]. It is posited that Hebbian-like mechanisms also operate on non-neural cells or neural groups [310]. "Metaplasticity" refers to the ability of neurons to alter the threshold at which LTP and LTD occur [2]. "Homeostatic plasticity" refers to the phenomenon where groups of neurons self-normalize their own level of activity [208].

1.2.3 Theories of Neural Organization

Empirical insights into mammalian brain development have spawned several theories regarding neural organization. We briefly present three of them in this section: *nativism*, *selectivism*, and *neural constructivism*.

The nativist view of neural development posits a strong genetic role in the construction of cognitive function. It claims that, after millions of years of evolutionary shaping, development is capable of generating highly specialized, innate neural structures that are appropriate for the various cognitive tasks that humans accomplish. On top of these fundamental neural structures, details can be adjusted by learning, like parameters. In cognitive science, it is argued that since children learn from a relative poverty of data (based on single examples and "one-shot learning"), there must be a native processing unit in the brain that preexists independently of environmental influence. Famously, this hypothesis led to the idea of a "universal grammar" for language [36], and some authors even posit that *all* basic concepts are innate [181]. According to a neurological (and controversial) theory, the cortex

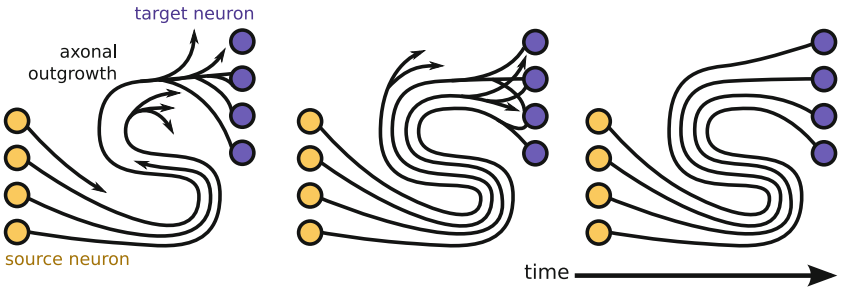

Fig. 2 Illustration of axonal outgrowth: initial overproduction of axonal connections and competitive selection for efficient branches leads to a globally efficient map (adapted from [294])

is composed of a repetitive lattice of nearly identical "computational units", typically identified with cortical columns [45]. While histological evidence is unclear, this view seems to be supported by physiological evidence that cortical regions can adapt to their input sources, and are somewhat interchangeable or "reusable" by other modalities, especially in vision- or hearing-impaired subjects. Recent neuro-imaging research on the mammalian cortex has revived this perspective. It showed that cortical structure is highly regular, *even across species*: fibre pathways appear to form a rectilinear 3D grid containing parallel sheets of interwoven paths [290]. Imaging also revealed the existence of arrays of assemblies of cells whose connectivity is highly structured and predictable *across species* [227]. Both discoveries suggest a significant role for regular and innate structuring in cortex layout (Fig. 2).

In contrast to nativism, selectivist theories focus on *competitive mechanisms* as the lead principle of structural organization. Here, the brain initially overproduces neurons and neural connections, after which plasticity-based competitive mechanisms choose those that can generate useful representations. For instance, theories such as Changeux's "selective stabilization" [34] and Katz's "epigenetic population matching" [149] describe the competition in growing axons for postsynaptic sites, explaining how the number of projected neurons matches the number of available cells. The quantity of axons and contacts in an embryo can also be artificially decreased or increased by excising target sites or by surgically attaching supernumerary limbs [272]. This is an important reason for the high degree of evolvability of the nervous system, since adaptation can be easily obtained under the same developmental mechanisms without the need for genetic modifications.

The regularities of neocortical connectivity can also be explained as a self-organization process during pre- and post-natal development via epigenetic factors such as ongoing biochemical and electrophysiological activity. These principles have been at the foundation of biological models of "topographically ordered mappings", i.e. the preservation of neighborhood relationships between cells from one sheet to another, most famously the bundle of fibers of the "retinotopic projection" from the retina to the visual cortex, via relays [293]. Bienenstock and Doursat have also proposed a model of selectivist self-structuration of the cortex [61, 65],

showing the possibility of simultaneous emergence of ordered chains of synaptic connectivity together with wave-like propagation of neuronal activity (also called "synfire chains" [1]). Bednar discusses an alternate model in Chap. 7.

A more debated selectivist hypothesis involves the existence of "epigenetic cascades" [268], which refer to a series of events driven by epigenetic population-matching that affect successive interconnected regions of the brain. Evidence for phenomena of epigenetic cascades is mixed: they seem to exist in only certain regions of the brain but not in others. The selectivist viewpoint also leads to several intriguing hypotheses about brain development over the evolutionary time scale. For instance, Ebbesson's "parcellation hypothesis" [74] is an attempt to explain the emergence of specialized brain regions. As the brain becomes larger over evolutionary time, the number of inter-region connections increases but due to competition and geometric constraints, these connections will preferentially target neighbouring regions. Therefore, the increase in brain mass will tend to form "parcels" with specialized functions. Another hypothesis is Deacon's "displacement theory" [51], which tries to account for the differential enlargement and multiplication of cortical areas.

More recently, the neural constructivism of Quartz and Sejnowski [234] casts doubt on both the nativist and selectivist perspectives. First, the developing cortex appears to be free of functionally specialized structures. Second, finer measures of neural diversity, such as type-dependent synapse counts or axonal/dendritic arborization, provide a better assessment of cognitive function than total quantities of neurons and synapses. According to this view, development consists of a long period of dendritic development, which slowly generates a neural structure mediated by, and appropriately biased toward, the environment.

These three paradigms highlight principles that are clearly at play in one form or another during brain development. However, their relative merits are still a subject of debate, which could be settled through modelling and computational experiments.

1.3 Brain Modelling

Computational neuroscience promotes the theoretical study of the brain, with the goal of uncovering the principles and mechanisms that guide the organization, information-processing and cognitive abilities of the nervous system [278]. A great variety of brain structures and functions have already been the topic of many modelling and simulation works, at various levels of abstraction or data-dependency. Models range from the highly detailed and generic, where as many possible phenomena are reproduced in as much detail as possible, to the highly abstract and specific, where the focus is one particular organization or behaviour, such as feed-forward neural networks. These different levels and features serve different motivations: for example, concrete simulations can try to predict the outcome of medical treatment, or demonstrate the generic power of certain neural theories, while abstract systems are the tool of choice for higher-level conceptual endeavours.

In contrast with the majority of computational neuroscience research, our main interest with this book, as exposed in this introductory chapter, resides in the potential to *use brain-inspired mechanisms for engineering challenges*.

1.3.1 Challenges in Large-Scale Brain Modelling

Creating a model and simulation of the brain is a daunting task. One immediate challenge is the scale involved, as billions of elements are each interacting with thousands of other elements nonlinearly. Yet, there have already been several attempts to create large-scale neural simulations (see reviews in [27, 32, 95]). Although it is a hard problem, researchers remain optimistic that it will be possible to create a system with sufficient resources to mimic all connections in the human brain within a few years [182]. A prominent example of this trend is the Blue Brain project, whose ultimate goal is to reconstruct the entire brain numerically at a molecular level. To date, it has generated a simulation of an array of cortical columns (based on data from the rat) containing approximately a million cells. Among other applications, this project allows generating and testing hypotheses about the macroscopic structures that result from the collective behaviours of instances of neural models [116, 184]. Other recent examples of large-scale simulations include a new proof-of-concept using the Japanese K computer simulating a (non-functional) collection of nearly 2×10^9 neurons connected via 10^{12} synapses [118], and Spaun, a more functional system consisting of 2.5×10^6 neurons and their associated connections. Interestingly, Spaun was created by top-down design, and is capable of executing several different functional behaviours [80]. With the exception of one submodule, however, Spaun does not "learn" in a classical sense.

Other important challenges of brain simulation projects, as reviewed by Cattell and Parker [32], include neural diversity and complexity, interconnectivity, plasticity mechanisms in neural and glial cells, and power consumption. Even more critically, the fast progress in computing resources able to support massive brain-like simulations is not any guarantee that such simulations will behave "intelligently". This requires a much greater understanding of neural behaviour and plasticity, at the individual and population scales, than what we currently have. After the recent announcements of two major funded programs, the EU Human Brain Project and the US Brain Initiative, it is hoped that research on large-scale brain modelling and simulation should progress rapidly.

1.3.2 Machine Learning and Neural Networks

Today, examples of abstract learning models are legion, and *machine learning* as a whole is a field of great importance attracting a vast community of researchers. While some learning machines bear little resemblance to the brain, many are inspired by their natural source, and a great part of current research is devoted to reverse-engineering natural intelligence.

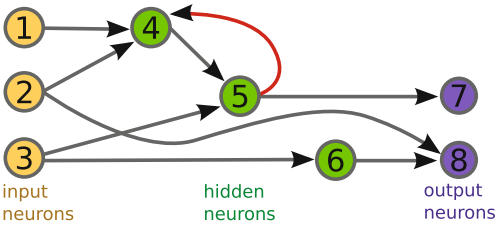

Fig. 3 Example of neural network with three *input neurons*, three *hidden neurons*, two *output neurons*, and nine connections. One feedback connection (*5→4*) creates a cycle. Therefore, this is a recurrent NN. If that connection was removed, the network would be feed-forward only

Chapter 2: A brief introduction to probabilistic machine learning and its relation to neuroscience.

In Chap. 2, Trappenberg provides an overview of the most important ideas in modern machine learning, such as support vector machines and Bayesian networks. Meant as an introduction to the probabilistic formulation of machine learning, this chapter outlines a contemporary view of learning theories across three main paradigms: unsupervised learning, close to certain developmental aspects of an organism, supervised learning, and reinforcement learning viewed as an important generalization of supervised learning in the temporal domain. Beside general comments on organizational mechanisms, the author discusses the relations between these learning theories and biological analogies: unsupervised learning and the development of filters in early sensory cortical areas, synaptic plasticity as the physical basis of learning, and research that relates models of basal ganglia to reinforcement learning theories. He also argues that, while lines can be drawn between development and learning to distinguish between different scientific camps, this distinction is not as clear as it seems since, ultimately, all model implementations have to be reflected by some morphological changes in the syste [279].

In this book, we focus on neural networks (NNs). Of all the machine learning algorithms, NNs provide perhaps the most direct analogy with the nervous system. They are also highly effective as engineering systems, often achieving state-of-the-art results in computer vision, signal processing, speech recognition, and many other areas (see [113] for an introduction). In what follows, we introduce a summary of a few concepts and terminology.

For our purposes, a neural network consists of a graph of neurons indexed by i. A connection $i \rightarrow j$ between two neurons is directed and has a weight w_{ij}. Typically, input neurons are application-specific (for example, sensors), output neurons are desired responses (for example, actuators or categories), and hidden neurons are information processing units located in-between (Fig. 3).

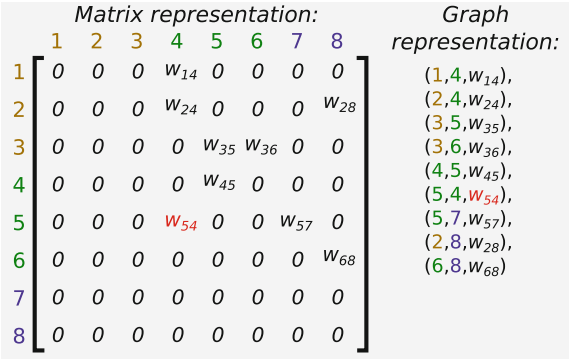

Fig. 4 Two representations for the neural network of Fig. 3

A neural network typically processes signals propagating through its units: a vector of floating-point numbers, s, originates in input neurons and resulting signals are transmitted along the connections. Each neuron j generates an output value v_j by collecting input from its connected neighbours and computing a weighted sum via an *activation function*, φ:

$$v_j(s) = \varphi\left(\sum_{i \mid (i \to j)} w_{ij}\, v_i(s)\right)$$

where $\varphi(x)$ is often a sigmoid function, such as $\tanh(x)$, making the output nonlinear. For example, in the neural network of Fig. 3, the output of neuron 8 can be written in terms of input signals v_1, v_2, v_3 as follows:

$$\begin{aligned} v_8(s) &= \varphi(w_{28}\, v_2 + w_{68}\, v_6) \\ &= \varphi(w_{28}\, v_2 + w_{68}\, \varphi(w_{36}\, v_3)) \end{aligned}$$

Graph topologies without cycles are known as *feedforward* NNs, while topologies with cycles are called *recurrent* NNs. The former are necessarily stateless machines, while the latter might possess some memory capacity. With sufficient size, even simple feed-forward topologies can approximate any continuous function [44]. It is possible to build a Turing machine in a recurrent NN [260].

A critical question in this chapter concerns the *representation* format of such a network. Two common representations are adjacency matrices, which list every possible connection between nodes, and graph-based representations, typically represented as a list of nodes and edges (Fig. 4). Given sufficient space, any NN topology and set of weights can be represented in either format.

Neural networks can be used to solve a variety of problems. In classification or regression problems, when examples of input-output pairs are available to the network during the learning phase, the training is said to be *supervised*. In this scenario, the fitness function is typically a mean square error (MSE) measured between the

network outputs and the actual outputs over the known examples. With feedback available for each training signal sent, NNs can be trained through several means, most often via gradient descent (as in the "backpropagation" algorithm). Here, a error or "loss function" E is defined between the desired and actual responses of the network, and each weight is updated according to the derivative of that function:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}}$$

where η is the learning rate. Generally, this kind of approach assumes a fixed topology and its goal is to optimize the weights.

On the other hand, *unsupervised learning* concerns cases where no output samples are available and data-driven self-organization mechanisms are at work, such as Hebbian learning. Finally, *reinforcement learning* (including neuroevolution) is concerned with delayed, sparse and possibly noisy rewards. Typical examples include robotic control problems, decision problems, and a large array of inverse problems in engineering. These various topics will be discussed later.

1.3.3 Brain-Like AI: What's Missing?

It is generally agreed that, at present, artificial intelligence (AI) is not "brain-like". While AI is successful at many specialized tasks, none of them shows the versatility and adaptability of animal intelligence. Several authors have compiled a list of "missing" properties, which would be necessary for brain-like AI. These include: the capacity to engage in a behavioural tasks; control via a simulated nervous system; continuously changing self-defined representations; and embodiment in the real world [165, 253, 263, 292]. Embodiment, especially, is viewed as critical because by exploiting the richness of information contained in the morphology and the dynamics of the body and the environment, intelligent behaviour could be generated with far less representational complexity [228, 291].

The hypothesis explored in this book is that *the missing feature is development*. The brain is not built from a blueprint; instead, it grows *in situ* from a complex multicellular process, and it is this adaptive growth process that leads to the adaptive intelligence of the brain. Our goal is not to account for all properties observed in nature, but rather to *identify the relevance of a developmental approach with respect to an engineering objective* driven by performance alone. In the remainder of this chapter, we review several approaches incorporating developmentally inspired strategies into artificial neural networks.

2 Artificial Development

There are about 1.5 million known species of multicellular organisms, representing an extraordinary diversity of body plans and shapes. Each individual grows from the division and self-assembly of a great number of cells. Yet, this developmental

process also imposes very specific constraints on the space of possible organisms, which restricts the evolutionary branches and speciation bifurcations. For instance, bilaterally symmetric cellular growth tends to generate organisms possessing pairs of limbs that are equally long, which is useful for locomotion, whereas asymmetrical organisms are much less frequent.

While the "modern synthesis" of genetics and evolution focused most of the attention on selection, it is only during the past decade that analyzing and understanding variation by comparing the developmental processes of different species, at both embryonic and genomic levels, became a major concern of evolutionary development, or "evo-devo". To what extent are organisms also the product of self-organized physicochemical developmental processes not necessarily or always controlled by complex underlying genetics? Before and during the advent of genetics, the study of developmental structures had been pioneered by the "structuralist" school of theoretical biology, which can be traced back to Goethe, D'Arcy Thompson, and Waddington. Later, it was most actively pursued and defended by Kauffman [150] and Goodwin [98] under the banner of *self-organization*, argued to be an even greater force than natural selection in the production of viable diversity.

By *artificial development* (AD), also variously referred to as artificial embryogeny, generative systems, computational ontogeny, and other equivalent expressions (see early reviews in [107, 265]), we mean the attempt to reproduce the constraints and effects of self-organization in automated design. Artificial development is about creating a growth-inspired process that will bias design outcomes toward useful forms or properties. The developmental engineer engages in a form of "meta-design" [63], where the goal is not to design a system directly but rather set a framework in which human design or automated search will specify a process that can generate a desired result. The benefits and effectiveness of development-based design, both in natural and artificial systems, became an active topic of research only recently and are still being investigated.

Assume for now that our goal is to generate a design which maximizes an objective function, o: $\Phi \rightarrow \mathbb{R}^n$, where Φ is the "phenotypic" space, that is, the space of potential designs, and $\mathbb{R}^n$ is a collection of performance assessments, as real values, with $n \geq 1$ ($n = 1$ denotes a single-objective problem, while $n > 1$ denotes a multiobjective problem). A practitioner of AD will seek to generate a lower-level "genetic" space Γ, a space of "environments" E in which genomes will be expressed, and a dynamic process δ that transforms the genome into a phenotype:

$$\Gamma \times E \xrightarrow{\delta} \Phi \xrightarrow{o} \mathbb{R}^n$$

In many cases, only one environment is used, usually a trivial or empty instance from the phenotypic space. In these cases, we simply write:

$$\Gamma \xrightarrow{\delta} \Phi \xrightarrow{o} \mathbb{R}^n$$

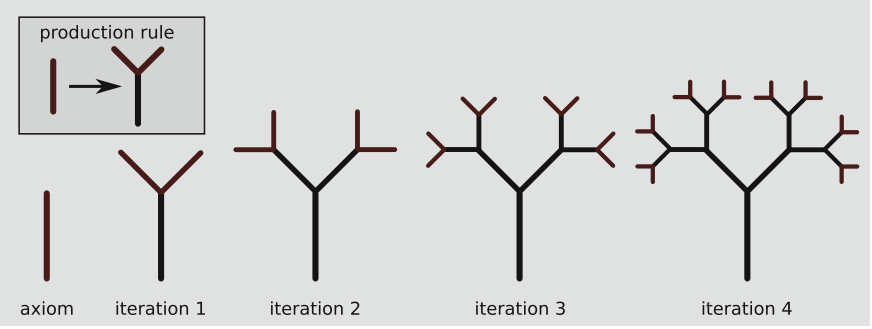

Fig. 5 Visualization of an L-System. *Top-left* a single production rule (the "genome"). *Bottom-left* the axiom (initial "word"). Recursive application of the production rule generates a growing structure (the "phenotype"). In this case, the phenotype develops exponentially with each application of the production rule

The dynamic process δ is inspired by biological embryogenesis, but need not resemble it. Regardless, we will refer to it as growth or development, and to the quadruple $(\Gamma, E, \delta, \Phi)$ as an *AD system*.

Often, the choice of phenotypic space Φ is dictated by the problem domain. For instance, to design neural networks, one might specify Φ as the space of all adjacency matrices, or perhaps as all possible instances of some data structure corresponding to directed, weighted graphs. Or to design robots, one might define Φ as all possible lattice configurations of a collection of primitive components and actuators. Sometimes there is value in restricting Φ, for example to exclude nonsensical or dangerous configurations. It is the engineer's task to choose an appropriate Φ and to "meta-design" the Γ, E, and δ parts that will help import the useful biases of biological growth into evolved systems.

A famous class of AD systems are the so-called L-Systems. These are formal grammars originally developed by Lindenmayer as a means of generating model plants [231]. In their simplest form, they are context-free grammars, consisting of a starting symbol, or "axiom", a collection of variables and constants, and at most one production rule per variable. By applying the production rules to the axiom, a new and generally larger string of symbols, or "word", is created. Repeated application of the production rules to the resulting word simulates a growth process, often leading to gradually more complex outputs. One such grammar is illustrated in Fig. 5, where a single variable (red stick) develops into a tree-like shape. In this case, the space of phenotypes Φ is the collection of all possible words (collections of sticks), the space of genotypes Γ is any nonambiguous set of context-free production rules, the environment E is the space in which a phenotype exists (here trivially 2D space), and the dynamic process δ is the repeated application of the rules to a given phenotype.

There are several important aspects to the meta-design of space of representations Γ and growth process δ. Perhaps the most critical requirement is that the chosen entities be "evolvable". This term has many definitions [129] but generally means that

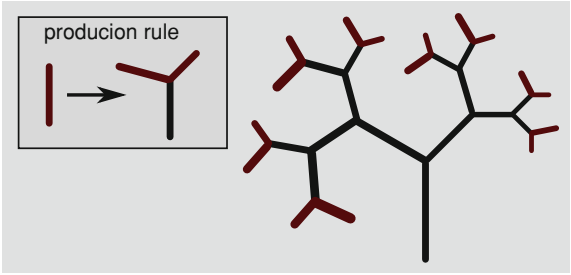

Fig. 6 A mutation of the production rule in Fig. 5, and the output after four iterations of growth

Fig. 7 McCormack's evolved L-Systems, inspired by, but exaggerating, Australian flora

the space of representations should be easily searchable for candidates that optimize some objective. A generally desirable trait is that small changes in a representation should lead to small changes in the phenotype—a "gentle slope" allowing for incremental search techniques. In AD systems, however, due to the nonlinear dynamic properties of the transformation process, it is not unusual for small genetic changes to have large effects on the phenotype [87].

For instance, consider in Fig. 6 a possible *mutation* of the previous L-System. Here, the original genome has undergone a small change, which has affected the resulting form. The final phenotypes from the original and the mutated version are

similar in this case: they are both trees with an identical topology. However, it is not difficult to imagine mutations that would have catastrophic effects, resulting in highly different forms, such as straight lines or self-intersections. Nonlinearity of the genotype-to-phenotype mapping δ can be at the same time a strength and a weakness in design tasks.

There is an important distinction to be made here between our motivations and those of systems biology or computational neuroscience. In AD, we seek means of creating engineered designs, not simulating or reproducing biological phenomena. Perhaps this is best illustrated via an example: McCormack, a computational artist, works with evolutionary computation and L-Systems (Fig. 7). Initially, this involved the generation of realistic models of Australian flora. Later, however, he continued to apply evolutionary methods to create exaggerations of real flora, artefacts that he termed "impossible nature" [187, 188]. McCormack's creations retain salient properties of flora, especially the ability to inspire humans, but do not model any existing organism.

2.1 Why Use Artificial Development?

Artificial development is one way of approaching *complex systems engineering*, also called "emergent engineering" [282]. It has been argued that the traditional state-based approach in engineering has reached its limits, and the principles underlying complex systems—self-organization, nonlinearity, and adaptation—must be accommodated in new engineering processes [11, 203]. Incorporating complex systems into our design process is necessary to overcome our present logjam of complexity, and open new areas of productivity. Perhaps the primary reason for the interest in simulations of development is that natural embryogenesis is a practical example of complex systems engineering, one which achieves designs of scale and functionality that modern engineers aspire to. There are several concrete demonstrations of importing desirable properties from natural systems into artificial counterparts. The key property of evolvability, which we have already discussed, is linked to a notion of scalability. Other related properties include robustness via self-repair and plasticity.

2.1.1 Scalability

Perhaps the best studied property of AD systems is the ability to scale to several sizes. This is a consequence of a general decoupling of the complexity of the genome (what we are searching for) from the phenotype (the final product). In many models, the size of the phenotype is controlled via a single parameter, which can be the number of repetitions of a module, the number of iterations in an L-System, or a single variable

controlling the amount of available resources. In these cases, a minimal change in the size of the genome might have exponential effects on the size of the resulting phenotype.

This property—the capacity to scale—brings to mind the notion of "Kolmogorov complexity", or the measurement of the complexity of a piece of data by the shortest computer program that generates it. With the decision to use AD, we make the assumption that there exists a short computer program that can generate our desired data, i.e. that the Kolmogorov complexity of our problem is small. This implies that AD will succeed in cases where the data to be generated is sufficiently large and non-random. Unfortunately, in the general case, finding such a program for some given data is an uncomputable problem, and to date there is no good approximation other than enumerating all possible programs, a generally untenable solution [173].

In many highly relevant domains of application, the capacity for scaling has been successfully demonstrated by AD systems. Researchers will often compare their AD model to a *direct encoding* model, in which each component of the solution is specified in the genome independently. Abstract studies have confirmed our intuition that AD systems are often better for *large* phenotypes and *nonrandom* data [40, 108]. This has also been demonstrated in neural networks [86, 104, 153], virtual robotics [161]; engineering design [127], and other domains [17, 243].

2.1.2 Robustness and Self-repair

Another desirable property of biological systems is the capacity for robustness. By this, we mean a "canalization" or the fact that a resulting phenotype is resistant to environmental perturbations, whether they are obstacles placed in the path of a developing organism, damage inflicted, or small changes to external factors affecting cellular expression, such as temperature or sources of nutrient. In biology, this ability is hypothesized to result from a huge number of almost identical cells, a redundancy creating tolerance toward differences in cellular arrangement, cell damage, or the location of organizers [152]. Several AD systems have been shown to import robustness, which can be selected for explicitly [18]. More interestingly, robustness is often imported without the inclusion of selection pressure [86, 161, 243]. In many cases, this property seems to be a natural consequence of the use of an adaptive growth process as a design step.

An extreme example of robustness is the capacity for self-repair. Many authors have conducted experiments with AD systems in which portions of an individual are damaged (e.g. by scrambling or removing components). In these cases, organisms can often self-repair, reconfiguring themselves to reconstruct the missing or altered portions and optimize the original objective. For instance, this has been demonstrated in abstract settings [5, 42, 145, 197], digital circuits [224], and virtual robotics [275]. Interestingly, in most of these cases, the self-repair capacity is not explicitly selected for in the design stage.

2.1.3 Plasticity

Another property of AD systems is plasticity, also referred to as polymorphism or polyphenism (although these terms are not strictly equivalent). By this, we mean the ability of organisms to be influenced by their environment and adopt as a result any phenotype from a number of possibilities. Examples in nature are legion [94], and most striking in the tendency of plants to grow toward light or food, or the ability of nervous systems to adapt to new stimuli. While robustness means reaching the same genotype under perturbation, plasticity means reaching *different* phenotypes under perturbation. Both, however, serve to improve the ultimate fitess of the organism in a variety of environments.

In classical neural systems, plasticity is the norm and is exemplified by well-known training methods: Hebbian learning, where connections between neurons are reinforced according to their correlation under stimuli [114], and backpropagation, where connection weights are altered according to an error derivative associated with incoming stimuli [245]. These classic examples focus on synaptic structure, or the weighting of connections in some predetermined network topology. While this is certainly an element of natural self-organization, it is by no means a complete characterization of the role that plasticity plays in embryogenesis. Environmental stimuli in animal morphogenesis include other neural mechanisms, such as the constant reformation and re-connection of synapses. Both selectivist and constructivist theories of brain development posit a central role for environmental stimuli in the generation of neural morphology. Furthermore, plasticity plays a major role in other developmental processes as well. In plants, the presence or absence of nutrients, light, and other cues will all but determine the coarse morphology of the resulting form. In animals, cues such as temperature, abundance of nutrients, mechanical stress, and available space are all strong influences. Indeed, the existence of plasticity is viewed as a strong factor in the evolvability of forms: for instance, plastic mechanisms in the development of the vascular system allow for a sort of "accidental adaptation", where novel morphological structures are well served by existing genetic mechanisms for vasculogenesis, despite never being directly selected for in evolutionary history [99, 177].

Most examples of artificial neural systems exploit plasticity mechanisms to tune parameters according to some set of "training" stimuli. Despite this, the use of environmentally induced plasticity in AD systems is rare. Only a few examples have shown that environmental cues can be used to reproduce plasticity effects commonly seen in natural phenomena, such as: virtual plant growth [87, 252], circuit design [280], or other scenarios [157, 190]. In one case, Kowaliw et al. experimented with the growth of planar trusses, a model of structural engineering. They initially showed that the coarse morphology of the structures could be somewhat controlled by the choice of objective function—however, this was also a difficult method of morphology specification [163]. Instead, the authors experimented with external constraints, which consisted of growing their structures in an environment that had the shape of the desired morphology. Not only was this approach generally successful in the sense of generating usable structures of the desired overall shape, but it

also spontaneously generated results indicating evolvability. A few of the discovered genomes could grow successful trusses not only in the specific optimization environment but also in *all* the other experimental environments, thus demonstrating a capacity for accidental adaptation [162].

2.1.4 Other Desirable Natural Properties

Other desirable natural properties are known to occasionally result from AD systems. These include: graceful degradation, i.e. the capacity for systems performance to fail continuously with the removal of parts [18]; adaptation to previously unseen environments, thought to be the result of repetitions of phenotypic patterns capturing useful regularities (see, for instance, Chap. 9 [206]); and the existence of "scafolding", i.e. a plan for the construction of the design in question, based on the developmental growth plan [241].

2.2 Models of Growth

An AD system requires a means of converting a representation into a design. This conversion typically involves a dynamic process that generates an arrangement of "cells", where these cells can stand for robotic components, structural members, neurons, and so on. Several models of multi-component growth have been investigated in detail:

- **Induced representational bias**: the designer adds a biologically inspired bias to an otherwise direct encoding. Examples include very simple cases, such as mirroring elements of the representation to generate symmetries in the phenotype [256], or enforcing a statistical property inspired by biological networks, such as the density of connections in a neural system [258].
- **Graph rewriting**: the phenotype is represented as a graph, the genome as a collection of graph-specific actions, and growth as the application of rules from the genome to some interim graph. Examples of this paradigm include L-Systems and dynamic forms of genetic programming [109, 122].
- **Cellular growth models**: the phenotype consists of a collection of cells on a lattice or in continuous space. The genome consists of logic that specifies associations between cell neighbourhoods and cell actions, where the growth of a phenotype involves the sum of the behaviours of cells. Cellular growth models are sometimes based on variants of *cellular automata*, a well-studied early model of discrete dynamics [161, 197]. This choice is informed by the success of cellular automata in the simulation of natural phenomena [56]. Other models involve more plausible physical models of cellular interactions, where cells orient themselves via intercellular physics [25, 62, 76, 144, 249]

- **Reaction-diffusion models**: due to Turing [281], they consist of two or more simulated chemical agents interacting on a lattice. The chemical interactions are modelled as nonlinear differential equations, solved numerically. Here, simple equations quickly lead to remarkable examples of self-organized patterns. Reaction-diffusion models are known to model many aspects of biological development, including overall neural organization [172, 259] and organismal behaviour [47, 298].
- Other less common but viable choices include: the **direct specification of dynamical systems**, where the genome represents geometric components such as attractors and repulsors [267]; the use of **cell sorting**, or the simulation of random cell motion among a collection of cells with various affinities for attraction, which can be used to generate a final phenotype [107].

A major concern for designers of artificial development (and nearly all complex systems) is how to find the micro-rules which will generate a desired macro-scale pattern. Indeed, this problem has seen little progress despite several decades of research, and in the case of certain generative machines such as cellular automata, it is even known to be impossible [133]. The primary way to solve this issue is using a machine learner as a search method. Evolutionary computation is the general choice for this machine learner, mostly due to the flexibility of genomic representations and objective functions, and the capacity to easily incorporate conditions and heuristics. In this case, the phenotype of the discovered design solution will be an unpredictable, emergent trait of bottom-up design choices, but one which meets the needs of the objective function. Various authors have explored several means of ameliorating this approach, in particular by controlling or predicting the evolutionary output [213, 214].

2.3 Why Does Artificial Development Work?

The means by which development improves the evolvability of organisms is a critical question. In biology, the importance of developmental mechanisms in organismal organization has slowly been acknowledged. Several decades ago, Gould (controversially) characterized the role of development as that of a "constraint", or a "fruitful channelling [to] accelerate or enhance the work of natural selection" [99]. Later authors envisioned more active mechanisms, or "drives" [7, 152]. More recently, discussion has turned to "increased evolvability", partly in recognition that no simple geometric or phenotypic description can presently describe all useful phenotypic biases [115]. At the same time, mechanisms of development have gained in importance in theoretical biology, spawning the field of evo-devo [31] mentioned above, and convincing several researchers that the emergence of physical epigenetic cellular mechanisms capable of supporting robust multicellular forms was, in fact, the "hard" part of the evolution of today's diversity of life [212].

Inspired by this related biological work, practitioners of artificial development have hypothesized several mechanisms as an explanation for the success of artificial development, or as candidates for future experiments:

- **Regularities**: this term is used ambiguously in the literature. Here, we refer to the use of simple geometrically based patterns over space as a means of generating or biasing phenotypic patterns, for example relying on Wolpert's notion of gradient-based positional information [295]. This description includes many associated biological phenomena, such as various symmetries, repetition, and repetition with variations. Regularities in artificial development are well studied and present in many models; arguably the first AD model, Turing's models of chemical morphogenesis, relied implicitly on such mechanisms through chemical diffusion [281]. A recent and popular example is the Compositional Pattern Producing Network (CPPN), an attempt to reproduce the beneficial properties of development without explicit multicellular simulation [266] (see also Sect. 5.4 and Chap. 5).
- **Modularity**: this term implies genetic reuse. Structures with commonalities are routine in natural organisms, as in the repeated vertebrae of a snake, limbs of a centipede, or columns in a cortex [29]. As Lipson points out, modules need not even repeat in a particular organism or design, as perhaps they originate from a meta-processes, such as the wheel in a unicycle [174]. Despite this common conception, there is significant disagreement on how to define modularity in neural systems. In cognitive science, a module is a functional unit: a specialized and encapsulated unit of function, but not necessarily related to any particular low-level property of neural organization [89, 233]. In molecular biology, modules are measured as either information-theoretic clusters [121], or as some measure of the clustering of network nodes [147, 211, 289]. These sorts of modularity are implicated in the separation of functions within a structure, allowing for greater redundancy in functional parts, and for greater evolvability through the separation of important functions from other mutable elements [229]. Further research shows that evolution, natural and artificial, induces modularity in some form, under pressures of dynamic or compartmentalized environments [23, 24, 39, 121, 147], speciation [82], and selection for decreased wiring costs [39]. In some cases, these same measures of modularity are applied to neural networks [23, 39, 147]. Beyond modularity, hierarchy (i.e. the *recursive composition of a structure and/or function* [64, 124, 174]) is also frequently cited as a possibly relevant network property.
- **Phenotypic properties**: Perhaps the most literal interpretation of biological theory comes from Matos et al., who argue for the use of measures on phenotypic space. In this view, an AD system promotes a bias on the space of phenotypic structures that can be reached, which might or might not promote success in some particular domain. By enumerating several phenotypic properties (e.g. "the number of cells produced") they contrast several developmental techniques, showing the bias of AD systems relative to the design space [185]. While this approach is certainly capable of adapting to the problem at hand, it requires *a priori* knowledge of the interesting phenotypic properties—something not presently existing for large neural systems;

- **Adaptive feedback and learning**: Some authors posit adaptive feedback during development as a mechanism for improved evolvability. The use of an explicit developmental stage allows for the incorporation of explicit cues in the resulting phenotype, a form of structural plasticity which recalls natural growth. These cues include not only a sense of the environment, as was previously discussed, but also interim indications of the eventual success of the developing organism. This latter notion, that of a continuous measure of viability, can be explicitly included in AD system, and has been shown in simple problems to improve efficacy and efficiency [12, 157, 158, 190]. A specialized case of adaptive feedback is *learning*, by which is meant the reaction to stimuli by specialized plastic components devoted to the communication and processing of inter-cellular signals. This important mechanism is discussed in the next section.

3 Artificial Neurogenesis

By artificial neurogenesis, we mean *a developmentally inspired process that generates neural systems for use in a practical context*. These contexts include tasks such as supervised learning, computer vision, robotic control, and so on. The definition of developmentally inspired processes in this chapter is also kept broad on purpose: at this early stage, we do not want to exclude the possibility that aspects of our current understanding of development are spurious or replaceable.

An interesting early example of artificial neurogenesis is Gruau's *cellular encoding* [103]. Gruau works with directed graph structures: each neural network starts with one input and one output node, and a hidden "mother" cell connected between them. The representation, or "genome", is a tree encoding that lists the successive cell actions taken during development. The mother cell has a reading head pointed at the top of this tree, and executes any cellular command found there. In the case of a division, the cell is replaced with two connected children, each with reading heads pointed to the next node in the genome. Other cellular commands change registers inside cells, by adding bias or changing connections. A simple example is illustrated in Fig. 8.

Through this graph-based encoding, Gruau et al. designed and evolved networks solving several different problems. Variants of the algorithm used learning as a midstep in development and encouraged modularity in networks through the introduction of a form of genomic recursion [103, 104]. The developed networks showed strong phenotypic organization and modularity (see Fig. 9 for samples).

3.1 The Interplay Between Development and Learning

A critical difference between artificial neurogenesis and AD is the emphasis on learning in the latter. Through the modelling of neural elements, a practitioner includes

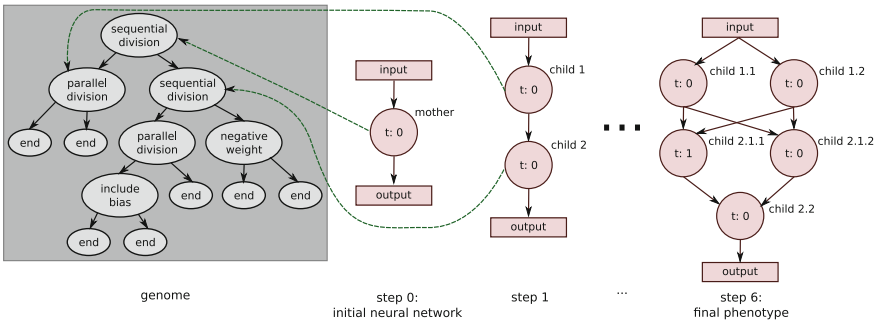

Fig. 8 Simple example of a neural network generated via cellular encoding (adapted from [103]). On the *left*, an image of the genome of the network. On the *right*, snapshots of the growth of the neural network. The *green arrows* show the reading head of the active cells, that is, which part of the genome they will execute next. This particular network solves the XOR problem. Genomic recurrence (not shown) is possible through the addition of a recurrence node in the genomic tree

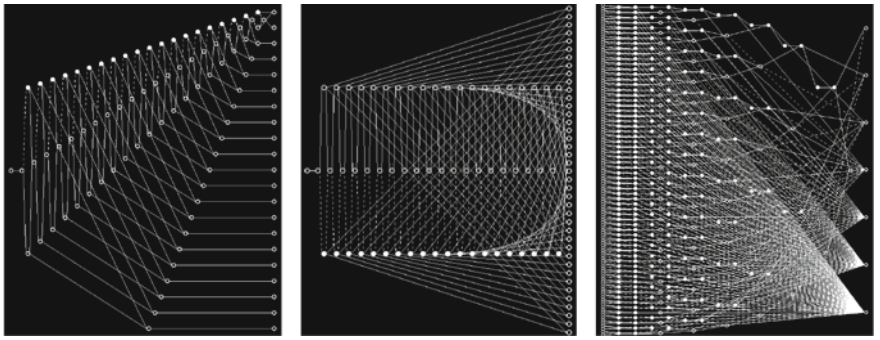

Fig. 9 Sample neural networks generated via cellular encoding: *left* a network solving the 21-bit parity problem; *middle* a network solving the 40-bit symmetry problem; *right* a network implementing a 7-input, 128-output decoder (reproduced with permission from [103])

any number of plasticity mechanisms that can effectively incorporate environmental information.

One such hypothetical mechanism requiring the interplay between genetics and epigenetics is the *Baldwin effect* [9]. Briefly, it concerns a hypothesized process that occurs in the presence of both genetic and plastic changes and accelerates evolutionary progress. Initially, one imagines a collection of individuals distributed randomly over a fitness landscape. As expected, the learning mechanism will push some, or all, of these individuals toward local optima, leading to a population more optimally distributed for non-genetic reasons. However, such organisms are under "stress" since they must work to achieve and maintain their epigenetically induced location in the fitness landscape. If a population has converged toward a learned optimum, then in subsequent generations, evolution will operate to lower this stress, by finding genetic

means of reducing the amount of learning required. Thus, learning will identify an optimum, and evolution will gradually adapt the genetic basis of the organism to fit the discovered optimum. While this effect is purely theoretical in the natural world, it has long been known that it can be generated in simple artificial organisms [120]. Accommodating developmental processes in these artificial models is a challenge, but examples exist [72, 103]. Other theories of brain organization, such as displacement theory, have also been tentatively explored in artificial systems [70, 71].

3.2 Why Use Artificial Neurogenesis?

There is danger in the assumption that all products of nature were directly selected for their contribution to fitness; this Panglossian worldview obscures the possibility that certain features of natural organisms are the result of non-adaptive forces, such as genetic drift, imperfect genetic selection, accidental survivability, side-effects of ontogeny or phylogeny, and others [100]. In this spirit, we note that while a computer simulation might show a model to be *sufficient* for the explanation of a phenomenon, it takes more work to show that it is indeed *necessary*. Given the staggering complexity of recent neural models, even a successful recreation of natural phenomena does not necessarily elucidate important principles of neural organization, especially if the reconstructed system is of size comparable to the underlying data source. A position of many practitioners working with bio-inspired neural models, as in artificial intelligence generally, is that an *alternative path to understanding neural organization is the bottom-up construction of intelligent systems*. The creation of artefacts capable of simple behaviours that we consider adaptive or intelligent gives us a second means of "understanding" intelligent systems, a second metric through which we can eliminate architectural overfitting from data-driven models, and identify redundant features of natural systems.

A second feature of many developmental neural networks is the reliance on local communication. Practitioners of AD will often purposefully avoid global information (e.g. in the form of coordinate spaces or centralized controllers) in order to generate systems capable of emergent global behaviour from purely local interactions, as is the case in nature. Regardless of historic motivations, this attitude brings potential benefits in engineered designs. First, it assumes that the absence of global control contributes to the scalability of developed networks (a special form of the robustness discussed in Sect. 2.1.1). Second, it guarantees that the resulting process can be implemented in a parallel or distributed architecture, ideally based on physically asynchronous components. Purely local controllers are key in several new engineering application domains, for instance: a uniform array of locally connected hardware components (such as neuromorphic engineering), a collection of modules with limited communication (such as a swarm of robots, or a collection of software modules over a network), or a group of real biological cells executing engineered DNA (such as synthetic biology).

3.3 Model Choices

A key feature in artificial neurogenesis is the level of simulation involved in the growth model. It can range from highly detailed, as is the case for models of cellular physics or metabolism, to highly abstract, when high-level descriptions of cellular groups are used as building blocks to generate form. While realism is the norm in computational neuroscience, simpler and faster models are typical in machine learning. An interesting and open question is whether or not this choice limits the capacity of machine learning models to solve certain problems. For artificial neurogenesis, relevant design decisions include: spiking versus non-spiking neurons, recurrent versus feed-forward networks, the level of detail in neural models (e.g. simple transmission of a value versus detailed models of dendrites and axons), and the sensitivity of neural firing to connection type and location.

Perhaps the most abstract models come from the field of *neuroevolution*, which relies on static feed-forward topologies and nonspiking neurons. For instance, Stanley's HyperNEAT model [49] generates a pattern of connections from another lattice of feed-forward connections based on a composition of geometric regularities. This model is a highly simplified view of neural development and organization, but can be easily evolved (see Chap. 5, [48]). A far more detailed model by Khan et al. [151] provides in each neuron several controllers that govern neural growth, the synaptogenesis of dendrites and axons, connection strength, and other factors. Yet, even these models are highly abstract compared to other works from computational neuroscience, such as the modelling language of Zubler et al. [311]. The trade-offs associated with this level of detailed modelling are discussed in depth by Miller (Chap. 8, [198]).

Assuming that connectivity between neurons depends on their geometric location, a second key question concerns the level of stochasticity in the placement of those elements. Many models from computational neuroscience assume that neural positions are at least partially random, and construct models that simply overlay preformed neurons according to some probability law. For instance, Cuntz et al. posit that synapses follow one of several empirically calculated distributions, and construct neural models based on samples from those distributions [41]. Similarly, the Blue Brain project assumes that neurons are randomly scattered: this model does, in fact, generate statistical phenomena which resemble actual brain connectivity patterns [116].

A final key decision for artificial neurogenesis is the level of detail in the simulation of neural plasticity. These include questions such as:

- Is plasticity modelled at all? In many applications of neuroevolution (Sect. 4.3), it is not: network parameters are determined purely via an evolutionary process.
- Does plasticity consist solely of the modification of connection weights or firing rates? This is the case in most classical neural networks, where a simple, almost arbitrary network topology is used, such as a multilayer perceptron. In other cases, connection-weight learning is applied to biologically motivated but static network topologies (Sects. 4.1 and 4.2, Chap.7 [13]).

- How many forms of plasticity are modelled? Recent examples in reservoir computing show the value of including several different forms (Sect. 6.1).
- Does the topology of the network change in response to stimuli? Is this change based on a constructive or destructive trigger (Sect. 6.2)? Is the change based on model cell-inspired synaptogenesis (Sect. 5)?

The plethora of forms of plasticity in the brain suggests different functional roles in cognition. For instance, artificial neural networks are prone to a phenomenon known as "catastrophic forgetting", that is, a tendency to rapidly forget all previously learned knowledge when presented with new data sources for training. Clearly, such forgetfulness will negatively impact our capacity to create multi-purpose machines [90]. Miller and Khan argue, however, that re-introducing metaphors for developmental mechanisms, such as dendritic growth, overcomes this limitation [201].

3.4 Issues Surrounding Developmental Neural Network Design

The use of a developmentally inspired representation or growth routine in neural network design implies a scale of network rarely seen in other design choices. Indeed, development is associated with the generation of large structures and is not expected to be useful below a minimal number of parts. This leads to several related issues for practitioners:

- Large networks are difficult to train via conventional means. This is mainly due to computational complexity, as training procedures such as backpropagation grow with the number of connections in a network.
- A more specific issue of size, *depth*, refers to the number of steps between the input and output of the network. It is known that there are exponentially more local optima in "deep" networks than "shallow" ones, and this has important consequences for the success of a gradient-descent technique in a supervised learning task. Despite these difficulties, depth is found to be useful because certain problems can be represented in exponentially smaller formats in deep networks [16].

These issues can be ameliorated via several new and highly promising neural techniques. On such technique is *reservoir computing*, where only a small subset of a large network is trained (Sect. 4.2). A second such technique is *deep learning*, where a deep network is preconditioned to suit the data source at hand (Sect. 4.1).

In much of statistical learning, there is a drive toward finding the most parsimonious representation possible for a solution. This is usually the case in constructive and pruning networks (Sect. 6.2), in which a smaller network is an explicit metric of success. Obviously, simpler solutions are more efficient computationally and can be more easily understood. However, it is further claimed that parsimonious solutions will also perform better on previously unseen data, essentially based on the *bias/variance trade-off* argument by Geman et al. [92]. They show that for a simple, fully connected network topology, the number of hidden nodes controls the level

of bias and variance in a trained classifier. Too many nodes lead to a network with excessive variance and overfitting of the training data. They conclude that the hard part of a machine learning problem is finding a representational structure that can support a useful "bias" toward the problem at hand. It means that a heuristic architectural search must precede the exploration and optimization of network parameters. Perhaps inspired by this and similar studies on limited representations, and the hope that smaller representations will have less tendencies to overfit, parsimony is often an explicit goal in optimization frameworks. Yet, we take here a different view: for us, *certain forms of redundancy in the network might in fact be one of the architectural biases that support intelligence*. In AD, redundancy is often celebrated for increasing resilience to damage, allowing graceful degradation, and creating neutral landscapes, or genetic landscapes that encourage evolvability [239, 248, 305].

4 Bio-Inspired Representations

Many neural models do not explicitly simulate any developmental process, yet they are substantially informed by biology through the observation the network structure of natural neural systems (or systems from computational neuroscience), and the inclusion of an explicit "bias" containing similar properties. Several of these approaches have proven tremendously successful in recent years, contributing to the so-called "second neural renaissance" that has reinvigorated research in artificial neural networks. We summarize below some of these bio-inspired representations.

4.1 Deep Learning

With the advent of deep learning, neural networks have made headlines again both in the machine learning community and publicly, to the point that "deep networks" could be seen on the cover of the New York Times. While deep learning is primarily applied to image and speech recognition [15, 46, 171], it is also mature enough today to work out of the box in a wide variety of problems, sometimes achieving state-of-the-art performance. For example, the prediction of molecular activity in the Kaggle challenge on Merck datasets (won by the Machine Learning group of the University of Toronto), or collaborative filtering and preference ranking in the Netflix movie database [246] both used deep learning.

These impressive results can be explained by the fact that deep learning very efficiently learns simple features from the data and combines them to build high-level detectors, a crucial part of the learning task. The features are learned in an unsupervised way and the learning methods are scalable: they yield the best results on the ImageNet problem [52, 166, 170], a dataset comprising 1,000 classes of common object images, after a training process that ran on a cluster of tens of thousands of CPUs and several millions of examples. Even through purely unsupervised training

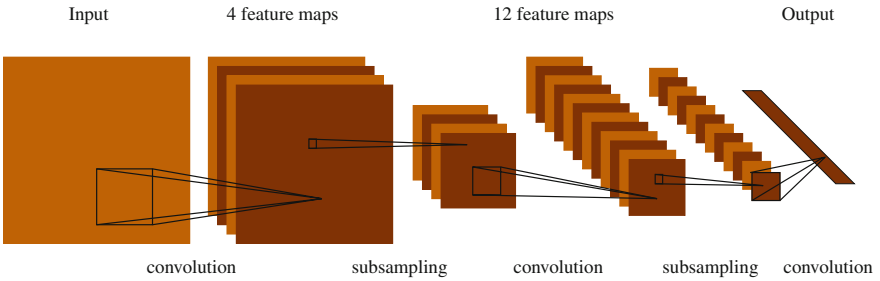

Fig. 10 Architecture of a convolution neural network, as proposed by LeCun in [171]. The convolutional layers alternate with subsampling (or pooling) layers

on YouTube images, the features learned are specialized enough to serve as face detectors or cat detectors. A straightforward supervised tuning of these unsupervised features often leans to highly effective classifiers, typically outperforming all other techniques.

Deep networks are similar to the classical multilayer perceptrons (MLP). MLPs are organized into "hidden layers", which are rows of neurons receiving and processing signals in parallel. These hidden layers are the actual locus of the computation, while the input and output layers provide the interface with the external world. Before deep learning, most multilayered neural nets contained only one hidden layer, with the notable exception of LeCun's convolutional network [171] (see below). One reason comes from the theoretical work of Håstad [112], who showed that all boolean circuits with $\ell + 1$ layers could be simulated with ℓ layers, at the cost of an exponentially larger number of units in each layer. Therefore, to make the model selection phase easier, for example chosing the number of units per layer, a common practice was to consider a single hidden layer. Another reason is that networks with more than one or two hidden layers were notoriously difficult to train [274], and the very small number of studies found in the literature that involve such networks is a good indicator of this problem.

Pioneering work on deep learning was conducted by LeCun [171], who proposed a family of perceptrons with many layers called convolutional networks (Fig. 10). These neural networks combine two important ideas for solving difficult tasks: shift-invariance, and reduction of dimensionality of the data. A convolution layer implements a filtering of its input through a kernel function common to all neurons of the layer. This approach is also called weight sharing, as all neurons of a given layer always have the same weight pattern. Convolution layers alternate with "pooling layers", which implement a subsampling process. The activation level of one neuron in a pooling layer is simply the average of the activity of all neurons from the previous convolution layer. In the first layer, the network implements a filter bank whose output is subsampled then convolved by the filter implemented in the next layer.

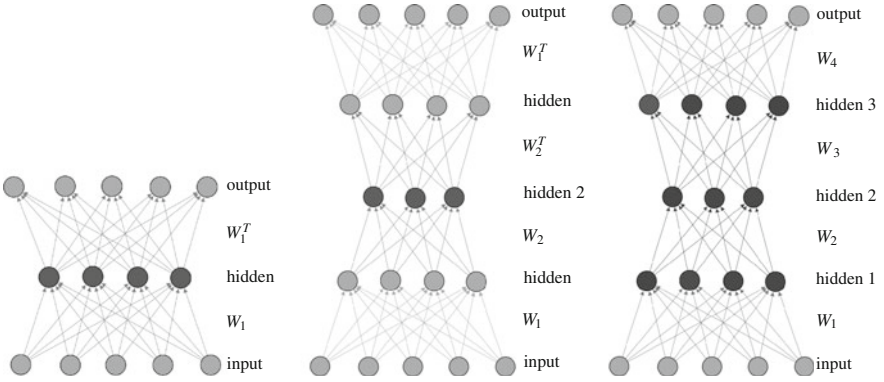

Fig. 11 Layer-wise unsupervised training in a deep architecture: *left* training of the first hidden layer, shown in *black*; *center* training of the second hidden layer, shown in *black*. Hidden layers and associated weights that are not subject to learning are shown in *grey*

Therefore, each pair of layers extracts a set of features from the input, which in turn feed into the next pair of layers, eventually building a whole hierarchy of features. Interesting variants of convolutional networks include L2-pooling, in which the L_2 norm of a neuron's activation in the previous layer is used instead of the maximum or the average [141], and contrast normalization, where the activities of the pooling neurons are normalized.

Hierarchical combination of features is the key ingredient of deep networks. In convolutional networks, the weight sharing technique allows learning a specific filter for each convolution map, which drastically reduces the number of variables required, and also explains why convolutional networks converge by simple stochastic gradient descent. On the other hand, weight sharing also limits the expressivity of the network, as each filter must be associated to a feature map and too many feature maps could negatively affect the convergence of the learning algorithm.

To overcome this trade-off, the method proposed by deep learning is to build the network step by step and ensure the learning of a feature hierarchy while maintaining good expressivity [81]. This is implemented via layer-wise unsupervised training, followed by a fine tuning phase that uses a supervised learning algorithm, such as gradient descent (Fig. 11). The idea of relying on unsupervised learning to train a network for a supervised task has been advocated by Raina et al. [235] in their work about self-taught learning. It is known that adding unlabelled examples to the training patterns improves the accuracy of the classifiers, an approach called "semi-supervised" learning [217]. In self-taught learning, however, any example and any signal can be used to improve the classifier's accuracy.

The underlying hypothesis is that recurring patterns in the input signal can be learned from any of the signal classes, and these typical recurrent patterns are helpful to discriminate between different signal classes. In other words, when the signal space

is large, it is possible to learn feature detectors that lie in the region containing most of the signal's energy, and then, classifiers can focus on this relevant signal space.

The layer-wise unsupervised objective of a deep network is to minimize the reconstruction error between the signal given on the input layer of the network and the signal reconstructed on the output layer. In the autoencoder framework, this first learning step, also called generative pretraining, focuses on a pair of parameters, the weight matrix W and the bias b of an encoder-decoder network. The encoder layer is a mapping f from the input signal x to an internal representation y:

$$y = f(x) = s(Wx + b) \tag{1}$$

where b is a bias vector and s is a non-linear function, usually a sigmoidal function. The decoder is a mapping from the internal state to a reconstructed signal z:

$$z = g(y) = s(W^T x + b^T) \tag{2}$$

In the left part of Fig. 11, the input vector x activates the neurons of the input layer, the internal state y of the hidden layer is expressed by Eq. (2) and the output layer is $z = g(f(x))$. The reconstructed error to minimize is then:

$$L(x, z) \propto -\log p(x|z) \tag{3}$$

A deep network can be built by stacking networks on top of each other. The most common are the autoencoders, also called auto-associators, which are often constrained to either ensure sparsity (sparse autoencoders) [15, 169, 236], or enforce generalization by purposefully corrupting the input signals, as with denoising autoencoders [81, 285]. Another widely investigated type of network is the restricted Boltzmann machine (RBM) [119], which is based on latent variables and a probabilistic formulation. A bound on accuracy ensures that stacking RBMs could only improve the accuracy of the whole architecture. Recent developments have shown that it is possible to use many different classifiers as the building blocks of a deep architecture. Nonetheless, the neural networks and their training have been sufficiently investigated to be integrated into a toolbox and applied without prior knowledge to nearly any pattern recognition problem.

The incremental method used in deep learning can be construed as a type of simplified evolutionary process, in which a first layer is set up to process certain inputs until it is sufficiently robust, then a second layer uses as input the output of the first layer and re-processes it until convergence, and so on. In a sense, this mimics an evolutionary process based on the "modularity of the mind" hypothesis [89], which claims that cognitive functions are constructed incrementally using the output of previous modules leading to a complex system. Another evolutionary perspective on deep learning, in relation with cultural development, is proposed by Bengio [14].

Chapter 3: Evolving culture versus local minima.

In Chap. 3, Bengio [14] provides a global view of the main hypotheses behind the training of deep architectures. It describes both the difficulties and the benefits of deep learning, in particular the ability to capture higher-level and more abstract relations. Bengio relates this challenge to human learning, and proposes connections to culture and language. In his theory, language conveys higher-order representations from a "teacher" to a "learner" architecture, and offers the opportunity to improve learning by carefully selecting the sequence of training examples—an approach known as Curriculum Learning. Bengio's theory is divided into several distinct hypotheses, each with proposed means of empirical evaluation, suggesting avenues for future research. He further postulates cultural consequences for his theory, predicting, for instance, an increase in collective intelligence linked to better methods of memetic transmission, such as the Internet.

From a computational viewpoint, signals acquired from natural observations often reside on a low-dimension manifold embedded in a higher-dimensional space. Deep learning aims at learning local features that characterize the neighbourhood of observed manifold elements. A connection could be made with sparse coding and dictionary learning algorithms, as described in [222], since all these data-driven approaches construct over-complete bases that capture most of the signal's energy. This line of research is elaborated and developed in Chap. 4 by Rebecchi, Paugam-Moisy and Sebag [236].

Chapter 4: Learning sparse features with an auto-associator.

In Chap. 4, Rebecchi, Paugam-Moisy and Sebag [236] review the recent advances in sparse representations, that is, mappings of the input space to a high-dimensional feature space, known to be robust to noise and facilitate discriminant learning. After describing a dictionary-based method to build such representations, the authors propose an approach to regularize auto-associator networks, a common building block in deep architectures, by constraining the learned representations to be sparse. Their model offers a good alternative to denoising auto-associator networks, which can efficiently reinforce learning stability when the source of noise is identified.

To deal with multivariate signals and particularly complicated time-series, several deep learning systems have been proposed. A common choice is to replicate and connect deep networks to capture temporal aspect of signals, using learning rules such as backpropagation through time. However, since these networks are recurrent, the usual gradient descent search does not converge. Consequently, "vanishing" or "exploding" gradient descents have also been the subject of an intense research

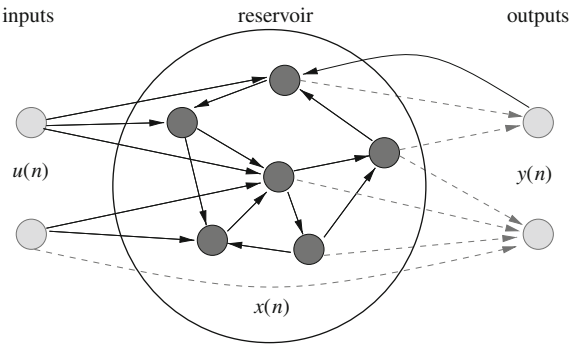

Fig. 12 A time-series $u(n)$ is assigned to *input* neurons. These *input* neurons are connected to the "reservoir", a recurrent neural network $x(n)$. Only a subset of $x(n)$ is connected to the output neurons $y(n)$

effort and have led to the development of reservoir computing approaches, which are detailed in the next section.

4.2 Reservoir Computing

Reservoir computing is an approach and family of models that rely on a recurrent neural network, called a reservoir, to generate a high-dimensional and dynamical representation of a given input, often a time series. Typically, the connections and weights in the network are randomly distributed and can produce a large number of nonlinear patterns from an input stream. Rather than modifying the reservoir with a supervised learning algorithm, however, the dynamical state of the reservoir is "read out" by a simple classifier, for example a linear regression or support vector machine, connected to a fraction of its neurons. This idea has been instantiated in different neural models, the two best known being Jaeger's Echo State Network (ESN) [137] and Maass' Liquid State Machine (LSM) [180, 210] (Fig. 12).

The ESN formulation uses a common sum of the weight / nonlinearity neurons as a neural model for the reservoir, such as McCulloch and Pitts neurons [189] or sigmoidal units. A good reservoir should produce a rich dynamics to facilitate the separability of its activity traces by the readout classifier, thus it is usually large, sparsely and randomly connected. It should also possess the *echo state property*, meaning that the effect of a previous state of the network should vanish asymptotically—in other words, the network should "forget" its previous state in a finite amount of time. To ensure this property, a common practice is to verify that the spectral radius $|\lambda_{\max}|$ of the weight matrix of the reservoir W is less than 1, and close to 1 for tasks requiring long memories. Other strategies have also been explored, taking into account more specialized neural types [303].

In other setups, the reservoir relies on more realistic neuronal models, such as integrate-and-fire or spiking neurons. While such temporal models endow the network with a richer dynamics, they also come at the expense of computational

efficiency, which can be noticeably reduced even in the case of simple spiking neurons [27]. Nonetheless, this type of reservoir form the basis of LSMs [210] and "cortical microcircuits" [180], which have been employed less frequently than ESNs. In any case, both types have been applied to a great variety of tasks and are theoretically linked [138].

One of the major difficulties of the LSM paradigm is the choice of the *readout classifier*. A simple linear regression achieves correct results and has been used to demonstrate theoretical characterizations [180], yet it ignores the fact that spiking neurons convey information through precise spike timings. Several propositions have been made to exploit this temporal information: encoding patterns with transient synchrony, as shown by Hopfield and Brody [123], or applying a margin classifier based on connection delays [226].

Whether LSMs or ESNs, another key element is the *topology* of the reservoir. The spectral radius $|\lambda_{\max}|$ of the weight matrix W plays a crucial role in determining the dynamics that will take place in the recurrent network. Other factors, such as small-world degree, scale-free regimes, and bio-inspired axonal growth patterns, have also been shown to positively influence the capabilities of the reservoir [242]. On the other hand, a recent theoretical analysis by Zhang et al. argues that all random reservoir topologies asymptotically converge to the same distribution of eigenvalues, implying that the topology is relatively indifferent after all [307]. Finer investigations of the dynamics are also possible [226] but they have not yet been applied in this context.

Beyond fixed topologies, a topic of great relevance to this chapter concerns endowing the reservoir with *plasticity*. Applying an unsupervised learning procedure to the weights allows the reservoir to adapt to very constrained topologies, although a theoretical analysis in this case becomes problematic [250]. The use of plasticity in reservoir computing will be discussed further in Sect. 6. Another widely investigated aspect is the influence of an *external loop*, by which the readout classification results are reinjected in the reservoir. This feedback adds another level of cognition to the network, as the system can now utilize its own capacity for prediction in input. An in-depth review of reservoir computing challenges and common practice can be found in a special issue of Neural Network [139], and a comprehensive explanation of the ongoing approaches is proposed in [179].

4.3 Neuroevolution

In evolutionary computation, there has been an long-standing interest in artificial neural networks for classification and regression, as well as control problems. The term "neuroevolution" is now well established and covers a large range of approaches (evolving weights, evolving topologies, learning rules, developmental processes) and applications. In this framework, the design of a particular neural network for solving a task is driven by its performance with respect to the defined task. This performance is itself described in terms of fitness value(s), and the evolutionary algorithm targets incremental improvements of fitness evaluations. To this aim, it

produces new candidate solutions (i.e. particular configurations of neural networks) from previously tried ones through the action of various mutation and recombination operators [79].

For example, neural networks have long been the method of choice of evolutionary robotics when performing "policy search" in reinforcement learning problems [219, 271, 273]. As a formalism for controller representation, they exhibit interesting features such as robustness (with respect to noisy sensory inputs), evolvability (in the sense that small weight changes give rise to small behavioral changes), and ease of implementation (since update time varies only linearly with the number of links). This is also true in situations with limited hardware specifications, such as onboard robotic systems.

Three important decisions may impact the choice of a learning method to train the network: (a) the definition of the neural network to be considered (with/without recurrent connections), (b) the choice of variables to learn (the weights of a fixed topology, and/or the topology), and (c) how these variables will be learnt (how to encode such a network, how to navigate through the search space). While there exist several methods in the literature for evolving weights only, such as classic multi-layered perceptrons or echo state networks [110], things become more challenging when evolving entire topologies. On the one hand, the choice of a particular search space relies for a great part on the programmer's expertise, and a poor guess may hinder the whole process. On the other hand, learning both the weights and the topology opens up a much larger search space and may well lead to performance normally unreachable through pure synaptic modification. Due to the versatility and robustness of evolutionary algorithms, they are considered promising candidates in the exploration of configuration spaces.

Evolutionary algorithms (EAs) quickly appeared as a relevant approach toward NN learning by the end of the 1990s (see [302] for a detailed survey). Although EAs can be useful for the optimization of the weights of a feedforward NN, they have instead been mainly used for their flexibility in handling complex search spaces. Many algorithms modifying the structure of neural networks through dedicated variation operators have been proposed.

Notable works and models in this field include: GNARL [6], which uses a direct encoding of the neural network to build a robot controller; EANT [148], which evolves the structure and weights via distinct processes; SANE (Symbiotic Adaptive Neuro-Evolution) [204] and ESP (Enforced Sub-Population) [96, 97], which evolve a population of neurons (rather than a network) and combine these neurons to form effective neural networks; and GASNET [132], which combines the optimization of the position of neurons in an Euclidean space through diffusion of chemicals. More recently, NEAT (Neuro Evolution of Augmenting Topologies) [264] has set new standards for neuroevolution algorithms in pure performance and speed of convergence based on classical benchmarks from evolutionary robotics.

It has been known for a long time [194] that the choice of a representation, i.e. search space, is crucial for the success of any evolutionary algorithm. This led to the exploration of genotype-to-phenotype maps using a more compact representation, which should theoretically enable the evolution of more complex neural networks.

One approach is the inclusion of an induced representational bias. In these cases, forms of structural organization known to be employed by natural neural systems are added to the otherwise directly encoded network. An illuminating study by Seys and Beer considers the value of several forms of induced symmetry on the generation of NNs [256]. Evolved genomes are "unpacked" by creating symmetric copies of the evolved network substructure, and evaluated via the whole unpacked network. The authors contrast their results against nonsymmetric networks, showing that the inclusion of symmetry makes NNs more evolvable, even compared to nonsymmetric networks of smaller size. A more practical example inspired by computational neuroscience models comes from Doncieux et al. [59]. Finally, recent works have explored the benefits of particular topological properties, such as regularity and modularity, whether a priori designed or evolved [33, 37, 284]. Another approach to the genotype-to-phenotype map strategy consists of including a simulation of development, a process that serves to construct the phenotype in a time-based fashion. The next section covers these strategies in greater detail.

5 Developmental Systems

This section gives an overview of neural *developmental systems*, which are about the abstraction of a developmental process to obtain artificial neural networks from simpler representations. Since, in the vast majority of cases, adequate representations (genomes) are optimized via evolutionary computation, we will use terminology from that field. Due to their metaphorical inspiration from developmental biology, developmental systems have received several names, including computational embryogeny [17], artificial ontogeny [25] and artificial embryogeny [265]. While all these terms emphasize the biological metaphor, we think that a broader phrasing of "evolution of developmental neural networks", or *evo-devo-NN* for short, would be more appropriate for this section.

The idea of combining evolution and development for designing artificial neural networks was first put to the test in 1990. Kitano [153] criticized direct encoding methods and proposed exploring indirect encodings as a promising solution to address the challenge of scalability. In this setup, the evolved genotypic description of a solution undergoes a reformulation process (or *mapping*) in order to obtain a usable phenotype. As noted in later works, this would enable it to evolve compact representations and possibly exploit properties such as modularity and hierarchy.

The debate about the relevance of evolving developmental neural networks has been, and still is, very much alive. In the first decade after Kitano's original claim regarding scalability, his results were first confirmed [75] (in a different context) then challenged [258] (in the original context). We now review the various approaches and works conducted in this area.

5.1 Grammar-Based Encoding

In his seminal work, Kitano [153] described the first approach with indirect encoding for generating artificial neural networks. Kitano used a genetic algorithm to evolve L-System grammar rules, a work which was later extended with lifelong neurogenesis [154]. His original contribution also fostered the emergence of a whole new field, which explored various approaches to developmental neural networks and addressed various challenges, some of them still open. Since then, evolutionary L-Systems have been further applied to the generation of neural networks [22, 225] or the co-evolution of artificial creatures [126]. While similar methods have been proposed to evolve morphologies, Hornby's GenRe system [126] relied on L-Systems for generating both body *and* brain (i.e. neural network) of virtual and real robots.

As previously discussed, in 1992 Gruau designed an original approach to the evolution of graph-rewriting rules called Cellular Encoding [101, 102]. His model was based on genetic programming to evolve a list of instructions that an original cell could follow to determine its fate. This cell would undergo several transformations (such as cell division) until a graph was built. A major contribution of this work was to provide the first-ever neural controller of hexapodal robot gait [103]. It was further extended [155, 178] and reused [25, 164] by several authors.

Other studies have explored the evolution of rewriting rules to generate neural networks. In the early 1990s, Nolfi and Parisi evolved direct encodings of neuron locations on a 2D substrate, then applied a heuristic for the simulation of axon growth (using previously evolved parameters) to obtain full-grown networks that were executing a robot navigation task. This work was later extended with cell division and migration [30], and lifetime adaptation through environment-triggered axon growth [218]. In 1994, Sims' "virtual creatures" also relied on evolved graph-rewriting rules both in the neural networks *and* in the morphologies [261]. Most recently, Mouret and Doncieux proposed Modular Encoding for Neural Networks based on Attribute Grammars (MENNAG), a general approach to evo-devo-NN based on the definition of grammar-based constraints [207], and the EvoNeuro method [205], which takes inspiration from computational neuroscience in order to generate large-scale and highly regular neural networks. Other applications exist as well [3].

5.2 Genetic Regulatory Networks

A major topic of interest in theoretical biology todayf is the modelling of *gene regulatory networks* (GRNs), which represent the interactions among genes and transcription products (mainly DNA-binding proteins) governing cell behaviour and maintenance. Generally, theoretical models are chosen based on their ability to

replicate specific patterns of expression found in natural systems [4, 150], or based on approximations of the molecular mechanisms of gene regulation [10]. In all cases, GRN simulations comprise a set of differential equations describing the dynamics of a various product concentrations. These models have been explored for their computational properties [144, 176, 215].

Offering a different approach to developmental systems, GRNs became a strong source of inspiration for researchers in computer science for obvious reasons. Starting from a relatively compact description such as a string of symbols, GRNs made it possible to build an entire network topology and function by defining interaction patterns among parts of the original representation. The possible benefits for control systems were first explored in 1994 by Dellaert and Beer [55] and quickly used to generate neural networks [54]. Their work combined a boolean GRN and a cell division process, alternating regulation and division over N iterations, in order to iteratively grow a full neural network. Although evolution was only discussed, some of the resulting networks were tested on a simplified robot navigation task as a proof of concept.

This first attempt was soon followed by others that had the same dual objective of taking inspiration from biology to achieve compact representations and, at the same time, addressing evolutionary robotics challenges. Jakobi [140] described a method to evolve a bit-string, acting as a GRN to grow a neural network for robot control. Eggenberger proposed a similar approach, stressing scalability as the main motivation, and applied it to robot morphogenesis [76] and pattern recognition in neural networks [78].

An interesting alternative came from Reisinger and Miikkulainen [238], who used an evolved GRN structure directly as a neural network architecture. Applying their system to game playing, the authors contrasted their GRN-based NN against several non-developmental alternatives, and found favourable results. Their analysis mentions several reasons for this success: their representation was significantly more compact, more evolvable under a simple mutation operator, and pushed phenotypes toward larger, more recurrent network motifs, typical of networks in nature.[3]

A more recent instance of GRN-inspired neural model is the AGE (Analog Genetic Encoding) model by Mattiussi and Floreano [73, 186], in which a string of symbols (rather than bits) represents a genotypic description, while the "coding" parts of the genome (i.e. syntactically correct with respect to the gene definitions) build a neural network. As with other GRN abstractions, the AGE process is a one-step transformation from the representation to the network, i.e. self-regulation is abstracted as a one-pass process. Wróbel et al. later proposed a system called GReaNs (Genetic Regulatory evolving artificial Networks), which shares many similarities with AGE. Among several applications, GReaNs has been used to evolve spiking neural networks [296]. In this scope, each gene stands for a node, and the connection between

[3] The authors point out a similarity between their developed NNs and natural networks, specifically the existence of higher-order network triads. However, Milo et al. [202] attribute the existence of such triads in natural networks to the minimization of information processing time, a factor which was not relevant to the NNs. Hence, we consider this similarity unexplained.

nodes is determined by their relative euclidian distance to one another—as is the case with AGE. The sign of this connection, however, is evolved seperately.

Chapter 6: Using the Genetic Regulatory evolving artificial Networks (GReaNs) platform for signal processing, animat control, and artificial multicellular development.

In Chap. 6, Wróbel and Joachimczak present their bio-inspired model of pattern formation and morphogenesis, GReaNs, and show that it can support an evo-devo approach to complex neural networks. The topology of a GReaN is encoded in a linear genome composed of genetic modules, including regulatory factors, regions of promoting or repressing elements, and inputs and outputs. The resulting genetic network is evolved toward the control of the behaviour of a cell, coupling the chemical simulation with mechanical outputs. The authors review the results of previous experiments in which GReaNs have been used to design single-celled animats, 2D soft-bodied animats, and 3D morphologies, as well as more recent work where spiking neuron models emerge from the GRNs. They conclude by laying out their vision for the evolution of plausible neural control mechanisms from genetic origins [297].

5.3 Cellular Automata Models

Also inspired from biology, several authors have explored models of multicellularity. Defined as a particular kind of cellular automaton (CA), each cell is capable of processing information, either by triggering further growth of the network or by relaying information as a neuronal cell. The seminal work from De Garis followed this metaphor to design the CAM-brain (Cellular Automata Machine), a two-dimensional CA in which a source cell could develop into a full organism capable of transmitting and manipulating information like a regular neural network [50]. This kind of approach raises the question of the halting problem, i.e. when and how development should stop [57]. Astor and Adami applied a similar approach based on a hexagonal grid, which addressed the halting problem by setting boundary cells to limit the total number of possible neurons. Adding a self-limiting mechanism allowed development to terminate before the environment was saturated with cells.

Early CA-based works were mostly limited to proof-of-concept experiments where evolution was merely discussed but not exploited. By contrast, Federici et al. designed a continuous CA implementation, which they termed *cell chemistry*, to generate spiking neural networks for solving a robotic navigation task in a discrete environment [84, 85]. In a later work, this approach was shown to outperform a direct encoding approach on a pattern recognition task [244].

5.4 HyperNEAT

In 2007, Stanley et al. presented the first version of HyperNEAT (Hybercube-based NeuroEvolution of Augmenting Topologies [49, 266]. Since then, it has become very popular and has been applied in many domains, including the evolution of neural networks. One of the key principles behind HyperNEAT is a high level of abstraction that emphasizes the expressivity of the genotype-phenotype mapping in terms of composed transformation functions, instead of a temporal development process.

Chapter 5: HyperNEAT: the first five years.
In Chap. 5, D'Ambrosio, Gauci, and Stanley summarize recent work on generating patterns with properties such as regularity, symmetry, and repetition with variations. This chapter successively considers spatial pattern generation using CPPNs (compositional pattern-producing networks), NEAT (NeuroEvolution of Augmenting Topologies), and neural connectivity pattern generation using the many flavours of HyperNEAT (Hybercube-based NEAT). The basic idea behind this work is to define the search space as a set of compositions of simple functions, each function with particular behaviours (e.g. favouring symmetries, repetitions, etc.). To some extent, HyperNEAT is an abstraction of the developmental process, mapping a compact representation to a possibly large phenotype, but removing the temporal aspects of such a process. The benefits of the HyperNEAT approach are presented and discussed: the compact encoding of large networks which posses relevant structural properties, the ability to generate solutions in various sizes and resolutions, and the exploitation of the geometric properties of the problem at hand. Finally, a short review of existing applications across several fields is given, from image generation to robotic control, from visual discrimination to playing chess and Go [48].

HyperNEAT has also inspired other works in various ways. The HybridID ("Hybridization of Indirect and Direct Encodings") algorithm [38, 40] tries to integrate the best of indirect encodings and direct encodings by successively applying HyperNEAT and FT-NEAT to refine the last steps of evolution. The DSE ("Developmental Symbolic Encoding") model [270] takes inspiration both from HyperNEAT and Cellular Encoding, retaining interesting properties such as the ability to create neural networks with regularity, modularity and scalability. DSE also provides an interesting complement to existing approaches as it focuses on specific problems for which scale-free network topologies are relevant. Alternatively, the NEON ("NeuroEvolution with ONtogeny") algorithm [134] extends the traditional NEAT algorithm with a developmental process.

5.5 *Beyond Artificial Neural Networks*

From the start, there have been strong interactions among subfields of artificial development and evolution: from evolvable hardware [175, 199, 200] to simulated dynamics of genetic regulatory networks [10, 64, 237]; from artificial models of morphogenesis [57, 63, 158, 161] to agent control [191, 192]. In robotics (virtual or real), the integration of development with morphofunctional machines shows great promise toward faster and more innovative methodologies of automated design. Under the name of *brain-body co-evolution*, an emerging trend of evolutionary computation argues that structure and function should not be predefined and optimized separately, but simultaneously as a whole, and based on the same genome. The work of Sims offered the first results with simulated robots [261], and was soon followed by researchers exploring various grammar-based encoding for a similar purpose, such as the works lead by Eggenberger [77], Hornby [126] and Bongard [23].

Recent research in this domain has also seen a division between works targeting engineering and more fundamental research. On the one hand, developmental systems for morphogenesis is illustrated by physical systems from Hornby [125], Hiller [117] and Rieffel [240], where more recent works benefit from the advent of versatile 3D printing machines. On the other hand, several authors have either explored virtual creatures [35, 66, 142, 143, 156, 193, 249], or considered a less robot-oriented interpretation of simulated morphofunctional machines [43, 62, 63, 247]. Doursat et al. [67, 68] propose a new approach encompassing these trends: "Morphogenetic Engineering" aims to reconcile engineering with decentralized complex systems. It explores new methodologies to model and create precise architectures that self-organize from a swarm of heterogeneous agents, in particular by development. It can also describe brain representations based on dynamic "neural shapes" in phase space, formed by myriads of correlated spikes [60].

From artificial neural networks to robotics, this shared interest in the developmental paradigm can be explained by the need for features (such as modularity, regularity, or hierarchy) that are considered relevant for functional or morphological reasons. Moreover, scalability stands as a critical issue in all these domains, and the combination of a compact genotype with a dedicated developmental process remains a promising track to achieve large phenotypes, as long as evolvability as a property is successfully retained.

6 Epigenetic Simulation

In this section, we consider algorithms that rely primarily on the simulation of *epigenetic mechanisms*, in the sense that they build neural networks from transient information provided by stimuli. From a certain perspective, this is already the norm

in artificial neural nets, where "classic" techniques involve simple and often fixed network topologies trained via stimulus-based methods such as backpropagation. Here, by contrast, we consider cases in which the structural design of the network is strongly influenced by the environment, or where a more biologically motivated synaptic plasticity mechanism has a significant effect on the topology.

Perhaps the best argumentation that development and epigenetics are both necessary in artificial networks comes from a study by Valsalam et al. [283]. In this work, the authors were concerned with exploring the role of prenatal and postnatal learning on the generation of a network. Under this viewpoint, development is modelled by non-environmentally induced stimuli, that is, patterns produced genetically rather than coming from the task at hand. Valsalam et al. explored three groups of models, all applied to hand-written character recognition and relying on a simple static network *(terminology ours)*:

- **learn**: in the first group, networks were trained by competitive Hebbian learning using input samples
- **evo**: in the second group, networks evolved through a simple neuro-evolutionary technique
- **pre-learn-evo**: in the third group, networks were trained by competitive Hebbian learning, first using genetically defined pretraining samples, then using input samples.

In summary, these three groups represented pure learning, pure genetic control, and a technique combining prenatal development and learning. The authors found that the two evolutionary models, "evo" and "pre-learn-evo", were far superior to "learn" in classifying hand-written characters. Furthermore, the "pre-learn-evo" type completed the task in a fraction of the time taken by "evo". They argued that the prenatal learning stage could be replaced with alternative forms of development for similar results. Valsalam et al. concluded that their prenatal stage implemented a form of bias on the space of neural models, which could be adjusted by evolution to adapt the particular network to the problem at hand. A more recent study by Tonelli and Mouret also shows that a combination of development (via map-based and HyperNEAT-like encodings) and plasticity can lead to improved learning efficacy, which they attribute to the increased propensity toward the generation of symmetric networks [277] (see also Chap. 9).

These studies are perfectly in line with the view of *development as a means of achieving useful phenotypic biases*, in this case via Hebbian learning. In a sense, some of these algorithms pose a challenge to the existence of developmental modelling in general, with the suggestion that very simple static topologies might be sufficient for intelligent behaviour when subjected to proper epigenetic mechanisms. Perhaps one of the most striking examples is given by the work of Bednar and colleagues:

Chapter 7: Constructing complex systems via activity-driven unsupervised Hebbian self-organization.

In Chap. 7, Bednar summarizes his recent work on exploring the use of Hebbian learning as a mechanism for the recreation of phenomena associated with the visual cortex. Starting from highly regular topologies, a simple form of Hebbian learning is applied. Through learning and the appropriate design of simple and complex cell layers, the major functional properties of the primary visual cortex emerge: receptive fields, selective topographic maps, surround modulation, visual contrast, and temporal responses. This impressive array of functional responses is notable for emerging simultaneously from a highly simple neural model, a result which suggest that most of the development and function of the first layer of the primary visual cortex can be viewed as an instance of unsupervised learning. Bednar goes on to discuss the lessons available from his model for the design of complex data-processing systems [13].

As in biology, it is difficult to determine whether certain phenomena should considered "strictly" developmental (in the sense of genetic control), or whether they depend on epigenetic processes. In reality, almost all scenarios integrate both mechanisms in a tight feedback loop. No amount of genetic information can control the fate and behavior of each cell, therefore a great many details have to depend on their interactions with one another and with environmental stimuli (which, for the most part, arise from the cell assembly itself). This is why a combination of developmental and epigenetic mechanisms will also be necessary in the simulation of intelligent networks. We summarize below three active areas of research that we characterize as epigenetic models: Hebbian pretraining, constructive and pruning algorithms, and epigenetic neuroevolution.

6.1 Hebbian Pretraining

Several recent models have explored the addition of Hebbian learning to bio-inspired representations. These have used reservoir computing instead of simpler feed-forward networks, and have concentrated on how to initialize and pretrain the reservoir.

Self-Organizing Recurrent Neural Network (SORN) is a model by Lazar et al. [168], which develops a recurrent neural network reservoir with a particular connectivity for inhibitory neurons. It includes three plasticity mechanisms: intrinsic plasticity, STDP, and synaptic normalization. SORN is trained via an echo state approach, and contrasted against static reservoirs and more limited forms of the model. The authors show that the conjunction of the three forms of plasticity outperform other configurations on simple learning tasks: counting and occluding. There is further suggestion that the organization of neural systems might be predictable from

the model. Zheng et al. [309] have constructed a version of SORN in which structure was a result only of internal plasticity (i.e. no external inputs), and tested over a range of parameters. They discovered that an emergent consequence of the model was a log-normal weight distribution, which resembles the organization found in nature, and has been implicated in the computational capacities of network in general. This suggests that the plasticity mechanisms alone in the absence of environmental stimuli are capable of generating useful organizational principles in a model cortex.

Yin et al. consider the addition of Hebbian learning mechanisms to a recurrent reservoir approach [304]. In this model, a genetic regulatory network specifies Hebbian and anti-Hebbian learning to generate plasticity parameters. The role of the genome here is to create a particular form of plasticity suitable for the problem at hand. An initially complete reservoir is then pruned according to the interplay between input and the GRN, leading to a sparse and pretrained reservoir. The networks are trained via "backpropagation through time" (BPTT), and evaluated on a collection of vision-based tasks with favourable results. Similar work has also been shown to have value in Liquid State Machines [220, 221].

6.2 Constructive and Pruning Algorithms

Closely related to the notion of using simulations of neural development are domains such as constructive neural networks (CoNNs) and pruning networks. Both are families of network design algorithms that operate by gradually changing a network structure in response to training data. They are designed to explore artificial versions of neural organization starting from two opposite viewpoints: CoNNs instantiate a form of *constructivist* process, whereas pruning networks illustrate a *selectivist* process.

In constructive algorithms, a small initial network (sometimes a single hidden neuron) is gradually transformed into a large network in a series of iterations. The network is trained until convergence or until some other stopping criterion has been met. Based on output from this training, the algorithm either terminates or adds more neurons or connections to the network. Once a global termination criterion is reached, the final, larger network is returned, possibly for additional training.

Perhaps the most popular CoNN algorithm is the cascade-correlation architecture [83], which has spawned numerous variants. In a recent review, Nicoletti et al. [58] have compiled a list of models and design decisions which characterize different CoNN approaches. More recent work has concentrated on network growth based on sensitivity analysis [106], adaptive neural activation functions [257], extreme learning machines [308], and extending CoNN to reinforcement learning [131].

In contrast, pruning algorithms start with a large network and gradually remove nodes. Initially, some large network is generated and trained. Next, particular neurons are selected as unimportant, and those neurons are deleted or merged. This process iterates until some global stopping criterion is reached, and finally, a smaller network is returned. Pruning algorithms are less restrained than CoNN algorithms, as

pruning plays a role in many different forms of neural networks and at different times. Deciding which neurons to prune can be made in many ways: for example, neurons that are highly correlated, neurons connected via weak weights, neurons with little influence over outputs, or neurons identified by more complex procedures, such as Optimal Brain Damage [300] and the Optimal Brain Surgeon [111]. A general disadvantage to pruning is that the use of large networks as a starting point tends to require significant computational effort. Recent work on pruning algorithms has included decisions to prune based on sensitivity analysis [167], component analysis [216], and competitive pressures [269]. Extensions to extreme learning machines [195] and other applications [306] have also been tried.

In their simplest forms, both types of algorithms are greedy and can fall prey to "architectural local optima" [6]. However, modern variants are more complex and less easily characterized. One such example, AMGA (adaptive merging and growing algorithm) comes from Islam et al. [136]. AMGA generates a network by both construction and pruning, using adaptive rules as triggers. Between iterations, a given network is trained via backpropagation. Construction occurs by splitting an existing hidden node, which results in the preservation of the behavioural linkages among neurons. Pruning occurs by merging highly correlated hidden nodes. AMGA is highly effective at supervised learning, outperforming several other neural and SVM techniques. The authors hypothesize that constructive-pruning hybrid techniques successfully avoid the local optima that hindered previous algorithms. Many such hybrid techniques have been explored [26, 105, 128, 130, 209, 232, 299].

CoNN and pruning algorithms are inspired by development, although motivations differ somewhat from those of developmental systems. They generally target the most parsimonious network possible, and show little interest for a parallel implementation of the algorithms, since they often rely on global data structures such as inter-neural correlations or Hessian matrices. Regardless, these techniques provide insight into how to execute ontogenic and epigenetic processes simultaneously.

6.3 Epigenetic Neuroevolution

Other authors have explored techniques that could be characterized as *epigenetic neuroevolution* as they offer a combination of evolutionary algorithms and learning techniques operating in tandem. Researchers in this category hope that such combination might return the best of both worlds: the high accuracy associated with epigenetic training and the capacity to explore a wide space of possible networks associated with neuroevolution, leading together to the ability to generalize to new environmental stimuli. Some authors also hope to avoid the architectural local minima generated by other non-evolutionary techniques[4]

[4] Caveat: while neuroevolution is known to be more versatile than, for instance, classic CoNN algorithms, it is also known that evolutionary computation will be often hindered by local optima in the fitness landscape, suggesting a possibly different sort of suboptimality.

A simple early example of epigenetic neuroevolution is outlined in Yao and Liu [301]. It is based on an evolutionary algorithm that controls the topology and weights of a network (via a form of genetic programming), while a learning routine is applied at mid-step to trigger the addition or deletion of neurons in the network. Training accuracy was used as the fitness of an individual during evolution. As the authors later argue, the issue with such a naive approach is that learning techniques based on gradient descent, when initialized with random weights, tend to be very noisy to the point of negatively affecting the evolutionary process. Using an averaged success over several independent learning sessions may provide a solution, but also tends to be too computationally expensive to serve as a fitness function [302].

To alleviate these problems, several authors including Yao and Liu have explored hybrid algorithms where the evolutionary search is global while the learning methods work on a local level. A recent example comes from Oong and Isa [223], who evolved a direct representation of the network via an adjacency matrix. This matrix, however, is augmented with a secondary genetic representation, a "node vector", which applies structural changes to the network topology. The interim success of the network is used to compute a measure of generalization loss, which in turn serves to control the weight of the evolutionary mutation. Thus, for Oong and Isa, instead of letting epigenetic information directly control the change of a network, it is a cue for the meta-process (evolution) to adjust the degree of exploration versus exploitation. Other forms of hybrid evolutionary-epigenetic algorithms, including the use of constructive techniques, have been explored [91].

Chapter 8: Neuro-centric and holocentric approaches to the evolution of developmental neural networks.

In Chap. 8, Miller explores two strategies of generating neural networks via neuroevolution. The first, a *neurocentric* approach, involves the detailed modelling of cells in a dynamic, time-based process. In this case, several independent control mechanisms are created, ones which emulate detailed sub-cellular behaviours. The second strategy, a *holocentric* approach, operates on a whole neural network. Here, network-specific operations make changes to sub-graphs of neurons and connections. By contrasting these two approaches, the author explores the value of the inclusion of a detailed and more plausible model of growth and plasticity relative to the additional computational costs involved. The chapter closes with design advice for practitioners [198].

In some cases, an explicit developmental process and a later epigenetic process are both included, which can make development occur twice: as a genotype-phenotype mapping process, and as a plastic property during operation, closely related to learning. Autonomous robotics is one prominent area of application, where neural network controllers are grown from compact genotypes (Sect. 4), and modification to the actual controller may occur during the robot's lifetime, whether the objective is long-term adaptation to the environment [151, 218], memorizing events [88], or learning new capabilities [262, 276].

Chapter 9: Artificial evolution of plastic neural networks: a few key concepts.

In Chap. 9, Mouret and Tonelli consider the use of neuroevolution to find plastic neural networks for reinforcement learning. A neuroevolutionary process produces a network topology, which is then trained via Hebbian learning in a (possibly reward-based) environment. Two key motivations for this type of approach are the promotion of behavioural robustness and reward-based behavioural change, concepts which suffer from inconsistent terminology in the literature. The authors provide new definitions of both concepts, and turn their attention to a key issue: the response of a neural network to previously unseen scenarios. To promote research on the topic, they define and discuss relevant concepts, such as the capacity for general and transitive learning, then theorize about the benefits of a developmental stage in terms of general learning [206].

7 Summary

In this introduction, we have explored the central hypothesis of this book that *adaptive growth is a means of producing brain-like machines*. The emulation of neural development can incorporate desirable characteristics of natural neural systems into engineered designs. We have reviewed several strategies for performing this "meta-design", which also involves identifying specific network biases and their benefits. In particular, we have seen that several recent studies show a strong synergy, sometimes interchangeability, between developmental and epigenetic processes—a topic that has remained largely under-explored in the literature. The chapters that follow in this book describe some of the most important works in this area, offering a state-of-the-art review of intelligent machine design.

Recent accelerating progress in observation and modelling techniques in neuroscience, and systems biology in general, ensures the continued generation of novel insights into brain organization. This new collection of "biases" should be further explored and exploited in neural networks over the coming years, suggesting that artificial neurogenesis is a promising avenue of research.

References

1. M. Abeles, *Local Cortical Circuits: An Electrophysiological Study*, vol. 6 (Springer, New York, 1982)
2. W.C. Abraham, M.F. Bear, Metaplasticity: the plasticity of synaptic plasticity. Trends Neurosci. **19**(4), 126–130 (1996)

3. I. Aho, H. Kemppainen, K. Koskimies, E. Makinen, T. Niemi, Searching neural network structures with l systems and genetic algorithms. Int. J. Comput. Math. **73**(1), 55–75 (1999)
4. U. Alon, *An Introduction to Systems Biology: Design Principles of Biological Circuits* (CRC Press, Boca Raton, 2007)
5. T. Andersen, R. Newman, T. Otter, Development of virtual embryos with emergent self-repair. in *AAAI Fall Symposium* (2006)
6. P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Netw. **5**(1), 54–65 (1994)
7. W. Arthur, The effect of development on the direction of evolution: toward a twenty-first century consensus. Evol. Dev. **6**(4), 282–288 (2004)
8. F.A.C. Azevedo, L.R.B. Carvalho, L.T. Grinberg, J.M. Farfel, R.E.L. Ferretti, R.E.P. Leite, W.J. Filho, R. Lent, S. Herculano-Houzel, Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. J. Comp. Neurol. **513**(5), 532–541 (2009)
9. J.M. Baldwin, A new factor in evolution. Am. Nat. **30**, 441–451 (1896)
10. W. Banzhaf, On the dynamics of an artificial regulatory network. in *European Conference on Artificial Life (ECAL 2003)* (Springer, Berlin, 2003), pp. 217–227
11. W. Banzhaf, N. Pillay, Why complex systems engineering needs biological development. Complexity **13**(2), 12–21 (2007)
12. J. Beal, Functional blueprints: an approach to modularity in grown systems. Swarm Intell. **5**(3–4), 257–281 (2011)
13. J.A. Bednar, *Constructing Complex Systems Via Activity-Driven Unsupervised Hebbian Self-organization*. in ed. by Kowaliw et al. [160], pp. 216–241
14. Y. Bengio, *Evolving Culture Versus Local Minima*. in ed. by Kowaliw et al. [160], pp. 112–143
15. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks. in *Advances in Neural Information Processing Systems* (2007)
16. Y. Bengio, Y. LeCun, Scaling learning algorithms towards AI. in *Large Scale Kernel Machines* (MIT Press, Cambridge, 2007)
17. P. Bentley, Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. in *Conference on Genetic and Evolutionary Computation* (1999), pp. 35–43
18. P. Bentley, Investigations into graceful degradation of evolutionary developmental software. Nat. Comput. **4**(4), 417–437 (2005)
19. E. Bienenstock, R. Doursat, Spatio-temporal coding and the compositionality of cognition. in *Temporal Correlations and Temporal Coding in the Brain* (1990), pp. 42–47
20. E. Bienenstock, C. von der Malsburg, A neural network for invariant pattern recognition. Europhys. Lett. **4**(1), 121–126 (1987)
21. E. Bienenstock, R. Doursat, A shape-recognition model using dynamical links. Netw. Comput. Neural Syst. **5**(2), 241–258 (1994)
22. E.J.W. Boers, H. Kuiper, *Biological Metaphors and the Design of Modular Artificial Neural Networks* Technical report (Leiden University, 1992)
23. J.C. Bongard, Evolving modular genetic regulatory networks. in *IEEE Congress on Evolutionary Computation (CEC)* (2002), pp. 1872–1877
24. J.C. Bongard, Spontaneous evolution of structural modularity in robot neural network controllers. in *Conference on Genetic and Evolutionary Computation (GECCO)* (Springer, 2011)
25. J.C. Bongard, R. Pfeifer, Evolving complete agents using artificial ontogeny. in ed. by F. Hara, R. Pfeifer *Morpho-functional Machines: The New Species (Designing Embodied Intelligence)* (Springer, 2003), pp. 237–258
26. M. Bortman, M. Aladjem, A growing and pruning method for radial basis function networks. IEEE Trans. Neural Netw. **20**(6), 1039–1045 (2009)
27. R. Brette, M. Rudolph, N.T. Carnevale, M.L. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris Jr, M. Zirpe, T. Natschläger, D. Pecevski, G.B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Viéville, E. Muller, A.P. Davison, S. El Boustani, A. Destexhe, Simulation of networks of spiking neurons: A review of tools and strategies. J. Comput. Neurosci. **23**(3), 349–398 (2007)

28. D.J. Cahalane, B. Clancy, M.A. Kingsbury, E. Graf, O. Sporns, B.L. Finlay, Network structure implied by initial axon outgrowth in rodent cortex: empirical measurement and models. PLoS ONE **6**(1), 01 (2011)
29. W. Callebaut, D. Rasskin-Gutman, *Modularity: Understanding the Development and Evolution of Natural Complex Systems* (MIT Press, 2005)
30. A. Cangelosi, D. Parisi, S. Nolfi. Cell division and migration in a genotype for neural networks. Conf. Comput. Netw. 497–515 (1994)
31. S. Carroll, J. Grenier, S. Weatherbee, *From DNA to Diversity: Molecular Genetics and the Evolution of Animal Design* 2nd edn (Blackwell Publishing, 2005)
32. R. Cattell, A. Parker, Challenges for brain emulation: why is it so difficult? Nat. Intell. INNS Mag. **1**(3), 17–31 (2012)
33. L. Cazenille, N. Bredeche, H. Hamann, J. Stradner, Impact of neuron models and network structure on evolving modular robot neural network controllers. in *Conference on Genetic and evolutionary computation (GECCO)*, (ACM Press, New York, 2012), p. 89
34. J.P. Changeux, A. Danchin, Nature **264**, 705–712 (1976)
35. N. Cheney, R. Maccurdy, J. Clune, H. Lipson, Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. in *Genetic and Evolutionary Computation Conference (GECCO)* (2013), pp. 167–174
36. N. Chomsky, *Aspects of the Theory of Syntax* (MIT Press, 1965)
37. J. Clune, B.E. Beckmann, P.K. McKinley, C. Ofria, Investigating whether hyperNEAT produces modular neural networks. in *Conference on Genetic and Evolutionary Computation (GECCO)*, (ACM Press, New York, 2010), pp. 1523–1530
38. J. Clune, B.E. Beckmann, R.T. Pennock, C. Ofria, HybrID: A hybridization of indirect and direct encodings for evolutionary computation. in *European Conference on Artificial Life (ECAL)* (2009), pp. 134–141
39. J. Clune, J.B. Mouret, H. Lipson, The evolutionary origins of modularity. Proc. Roy. Soc. B Biol. Sci. **280**(1755), 20122863 (2013)
40. J. Clune, K.O. Stanley, R.T. Pennock, C. Ofria, On the performance of indirect encoding across the continuum of regularity. IEEE Trans. Evol. Comput. **15**(3), 346–367 (2011)
41. H. Cuntz, F. Forstner, A. Borst, M. Häusser, One rule to grow them all: a general theory of neuronal branching and its practical application. PLoS Comput. Biol. **6**(e1000877), 08 (2010)
42. S. Cussat-Blanc, H. Luga, Y. Duthen, Cell 2Organ: Self-repairing artificial creatures thanks to a healthy metabolism. in *IEEE Congress on Evolutionary Computation (CEC)* (2009), pp. 2708–2715
43. S. Cussat-Blanc, J. Pascalie, S. Mazac, H. Luga, Y. Duthen, A synthesis of the cell2organ developmental model. in Doursat et al. [67], pp. 353–381
44. G. Cybenko, Approximations by superpositions of sigmoidal functions. Math. Control Sig. Syst. **4**(2), 303–314 (1989)
45. N.M. da Costa, K.A.C. Martin, Whose cortical column would that be? Front. Neuroanat. **4**(16), (2010)
46. G. Dahl, M. Ranzato, A. Mohamed, G.E. Hinton, Phone Recognition with the mean-covariance restricted Boltzmann machine. in *Advances in Neural Information Processing Systems* (2010), pp. 469–477
47. K. Dale, P. Husbands, The evolution of reaction-diffusion controllers for minimally cognitive agents. Artif. Life **16**, 1–19 (2010)
48. D.B. DAmbrosio, J. Gauci, K.O. Stanley, HyperNEAT: the first five years. in ed. by Kowaliw et al. [160], pp. 167–197
49. D.B. D'Ambrosio, K.O. Stanley, A novel generative encoding for exploiting neural network sensor and output geometry. in *Conference on Genetic and Evolutionary Computation (GECCO)* (ACM Press, New York, 2007), pp. 974–982
50. H. De Garis, Growing an artificial brain: the genetic programming of million-neural-net-module artificial brains within trillion cell cellular automata machines. in *Proceedings of the Third Annual Conference on Evolutionary Programming* (1994), pp. 335–343
51. T.W. Deacon, Rethinking mammalian brain evolution. Am. Zool. **30**(3), 629–705 (1990)

52. J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A. Ng, Large scale distributed deep networks. in ed. by P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger. *Advances in Neural Information Processing Systems 25* (2012), pp. 1232–1240
53. A.S. Dekaban, D. Sadowsky, Changes in brain weights during the span of human life: Relation of brain weights to body heights and body weights. Ann. Neurol. **4**(4), 345–356 (1978)
54. F. Dellaert, R.D. Beer, *Co-evolving Body and Brain in Autonomous Agents Using a Developmental Model*. Technical report, Department of Computer Engineering and Science (Case Western Reserve University, Cleveland, 1994)
55. F. Dellaert, R.D. Beer, Toward an evolvable model of development for autonomous agent synthesis. in *Proceedings of the fourth International Workshop on the Synthesis and Simulation of Living Systems (ALIFE Workshop)* (1994)
56. A. Deutsch, S. Dormann, *Cellular Automaton Modelling of Biological Pattern Formation: Characterization, Applications and Analysis* (Birkhauser, 2005)
57. A. Devert, N. Bredeche, M. Schoenauer, Robustness and the halting problem for multicellular artificial ontogeny. IEEE Trans. Evol. Comput. **15**(3), 387–404 (2011)
58. M. do Carmo Nicoletti, J. Bertini, D. Elizondo, L. Franco, J. Jerez, Constructive neural network algorithms for feedforward architectures suitable for classification tasks. in ed. by L. Franco, D. Elizondo, J. Jerez. *Constructive Neural Networks, Studies in Computational Intelligence*, vol. 258 (Springer, Heidelberg, 2009), pp. 1–23
59. S. Doncieux, J.-B. Mouret, T. Pinville, P. Tonelli, B. Girard, The evoneuro approach to neuroevolution. in Kowaliw et al. [159], pp. 10–14
60. R. Doursat, Bridging the mind-brain gap by morphogenetic neuron flocking: The dynamic self-organization of neural activity into mental shapes. in *2013 AAAI Fall Symposium Series* (2013)
61. R. Doursat, Contribution à l'étude des représentations dans le système nerveux et dans les réseaux de neurones formels. PhD thesis, Université Pierre et Marie Curie (Paris 6), 1991
62. R. Doursat, Facilitating evolutionary innovation by developmental modularity and variability. in *Conference on Genetic and Evolutionary Computation (GECCO)* (ACM, 2009), pp. 683–690
63. R. Doursat, Organically grown architectures: creating decentralized, autonomous systems by embryomorphic engineering. in ed. by R.P. Würtz. *Organic computing, Understanding Complex Systems* (Springer, 2008), pp. 167–199
64. R. Doursat, The growing canvas of biological development: multiscale pattern generation on an expanding lattice of gene regulatory networks. InterJournal Complex Syst. 1809 (2006)
65. R. Doursat, E. Bienenstock, Neocortical self-structuration as a basis for learning. in *5th International Conference on Development and Learning (ICDL 2006)* (2006), pp. 1–6
66. R. Doursat, C. Sánchez, R. Dordea, D. Fourquet, T. Kowaliw, Embryomorphic engineering: emergent innovation through evolutionary development. in ed. by Doursat et al. [67], pp. 275–311
67. R. Doursat, H. Sayama, O. Michel (eds.), *Morphogenetic Engineering: Toward Programmable Complex Systems. Understanding Complex Systems* (Springer, 2012)
68. R. Doursat, H. Sayama, O. Michel, A review of morphogenetic engineering. *Nat. Comput.* 1–19 (2013)
69. J.E. Dowling, *The Great Brain Debate: Nature or Nurture?* (Princeton University Press, 2007)
70. K. Downing, A neural-group basis for evolving and developing neural networks. in *AAAI-Devp* (2006)
71. K. Downing, Supplementing evolutionary developmental systems with abstract models of neurogenesis. in *9th Genetic and Evolutionary Computation Conference (GECCO)* (2007), pp. 990–996
72. K. Downing, The Baldwin effect in developing neural networks. in *Genetic and Evolutionary Computation Conference (GECCO)* (2010), pp. 555–562
73. P. Durr, C. Mattiussi, D. Floreano, Neuroevolution with analog genetic encoding. in *Parallel Problem Solving from Nature (PPSN)* (2006), pp. 671–680

74. S.O.E. Ebbesson, The parcellation theory and its relation to interspecific variability in brain organization, evolutionary and ontogenetic development, and neuronal plasticity. Cell Tissue Res. **213**(2), 179–212 (1980)
75. P. Eggenberger Hotz, Creation of neural networks based on developmental and evolutionary principles. in *International Conference on Artificial, Neural Networks* (1997), pp. 337–342
76. P. Eggenberger Hotz, Evolving morphologies of simulated 3D organisms based on differential gene expression. in *European Conference on Artificial Life (ECAL)* (MIT Press, 1997), pp. 205–213
77. P. Eggenberger Hotz, Evolving morphologies of simulated 3D organisms based on differential gene expression. in *European Conference on Artificial Life (ECAL)* (1997), pp. 205–213
78. P. Eggenberger Hotz, G. Gomez, R. Pfeiffer, Evolving the morphology of a neural network for controlling a foveating retina and its test on a real robot. in *Artificial Life 8* (2002), pp. 243–251
79. A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing* (Springer, 2003)
80. C. Eliasmith, T.C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, D. Rasmussen, A large-scale model of the functioning brain. Science **338(20)**, 1202-1205 (2012)
81. D. Erhan, Y. Bengio, A. Courville, P.A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help dDeep learning? J. Mach. Learn. Res. **11**, 625–660 (2010)
82. C. Espinosa-Soto, A. Wagner, Specialization can drive the evolution of modularity. PLoS Comput. Biol. **6**(3), e1000719 (2010)
83. S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture. in ed. by D.S. Touretzky. *Advances in Neural Information Processing Systems 2*, (Morgan Kaufmann, 1990), pp. 524–532
84. D. Federici, Evolving a neurocontroller through a process of embryogeny. in *Proceeding of Simulation of Adaptive Behavior (SAB)* (2004), pp. 373–384
85. D. Federici, Evolving developing spiking neural networks. in *IEEE Congress on Evolutionary Computation* (2005), pp. 43–550
86. D. Federici, K. Downing, Evolution and development of a multicellular organism: Scalability, resilience, and neutral complexification. Artif. Life **12**(3), 381–409 (2006)
87. J.D. Fernández, D. Lobo, G.M. Martín, R. Doursat, F.J. Vico, Emergent diversity in an open-ended evolving virtual community. Artif. Life **18**(2), 199–222 (2012)
88. D. Floreano, J. Urzelai, Neural morphogenesis, synaptic plasticity, and evolution. Theory Biosci. **120**(3–4), 225–240 (2001)
89. J.A. Fodor, *Modularity of Mind: An Essay on Faculty Psychology* (MIT Press, 1983)
90. R.M. French, Catastrophic forgetting in connectionist networks. Trends Cogn. Sci. **3**(4), 128–135 (1999)
91. N. García-Pedrajas, D. Ortiz-Boyer, A cooperative constructive method for neural networks for pattern recognition. Pattern Recogn. **40**(1), 80–98 (2007)
92. S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992)
93. S.F. Gilbert, *Developmental Biology* 8 edn. (Sinauer Associates, 2008)
94. S.F. Gilbert, D. Epel, *Ecological Developmental Biology* 1 edn. (Sinauer Associates, 2008)
95. B. Goertzel, R. Lian, I. Arel, H. de Garis, S. Chen, A world survey of artificial brain projects, part II: biologically inspired cognitive architectures. Neurocomputing **74**(1–3), 30–49 (2010)
96. F. Gomez, R. Miikkulainen, Solving non-markovian control tasks with neuro-evolution. in *IJCAI* (1999), pp. 1356–1361
97. F. Gomez, R. Miikkulainen, Incremental evolution of complex general behavior. Adapt. Behav. **5**, 317–342 (1997)
98. B.C. Goodwin, *How the Leopard Changed Its Spots: The Evolution of Complexity* (Scribner, 1994)
99. S.J. Gould, *The Structure of Evolutionary Theory* (The Belknap Press of Harvard University Press, 2002)
100. S.J. Gould, R. Lewontin, The spandrels of san marco and the panglossian paradigm: a critique of the adaptationist programme. Proc. Roy. Soc. London Ser. B Biol. Sci. **205**(1161), 581–598 (1979)

101. F. Gruau, Cellular encoding as a graph grammar. in *Grammatical Inference: IEE Colloquium on Theory, Applications and Alternatives* (1993), pp. 1–17
102. F. Gruau, Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. in *Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks* (IEEE Computer Society Press, 1992), pp. 55–74
103. F. Gruau, *Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm* (PhD thesis, Université Claude Bernard-Lyon, 1994)
104. F. Gruau, D. Whitley, L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks. in *Conference on Genetic Programming* (1996), pp. 81–89
105. H.-G. Han, J.-F. Qiao, A structure optimization algorithm for feedforward neural network construction. Neurocomputing **99**, 347–357 (2012)
106. H.-G. Han, J.-F. Qiao, A repair algorithm for radial basis function neural network and its application to chemical oxygen demand modeling. Int. J. Neural Syst. **20**(01), 63–74 (2010)
107. S. Harding, W. Banzhaf, Artificial development. in *Organic Computing, Understanding Complex Systems* (Springer, Heidelberg, 2008), pp. 201–219
108. S.L. Harding, J.F. Miller, The dead state: A comparison between developmental and direct encodings (updated version). in *Workshop on Complexity through Development and Self-Organizing Representations (CODESOAR), Genetic and Evolutionary Computation Conference (GECCO)* (2006)
109. S.L. Harding, J.F. Miller, W. Banzhaf, Self-modifying cartesian genetic programming, in ed. by J.F. Miller *Cartesian Genetic Programming*, Natural Computing Series (Springer, Berlin, 2011), pp. 101–124
110. C. Hartland, N. Bredeche, M. Sebag, Memory-enhanced evolutionary robotics: the echo state network approach. in *IEEE Congress on Evolutionary Computation, 2009 (CEC)* (2009), pp. 2788–2795
111. B. Hassibi, D.G. Stork, Second order derivatives for network pruning: Optimal brain surgeon. in *Advances in Neural Information Processing Systems* (1993), pp. 164–164
112. J. Hastad, Almost optimal lower bounds for small depth circuits. in *Proceedings of the Eighteenth Annual ACM Aymposium on Theory of Computing, STOC '86* (ACM, New York, 1986), pp. 6–20
113. S. Haykin, *Neural Networks and Learning Machines* 3 edn. (Pearson Inc., 2009)
114. D.O. Hebb, *The Organization of Behavior* (Wiley, New York, 1949)
115. J.L. Hendrikse, T.E. Parsons, B. Hallgrímsson, Evolvability as the proper focus of evolutionary developmental biology. Evol. Dev. **9**(4), 393–401 (2007)
116. S.L. Hill, Y. Wang, I. Riachi, F. Schürman, H. Markram, Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits. vol. 18 *Proceedings of the National Academy of Sciences* (2012)
117. J. Hiller, H. Lipson, Automatic design and manufacture of soft robots. IEEE Trans. Robot. **28**, 457–466 (2012)
118. R. Himeno, J. Savin, Largest neuronal network simulation achieved using K computer @ONLINE. http://www.riken.jp/en/pr/press/2013/20130802_1/. Accessed: 09/2013
119. G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006)
120. G.E. Hinton, S.J. Nowlan, How learning can guide evolution. Complex Syst. **1**, 495–502 (1987)
121. A. Hintze, C. Adami, Evolution of complex modular biological networks. PLoS Comput. Biol. **4**(2), 1–12 (2008)
122. T.-H. Hoang, R.I. McKay, D. Essam, N.X. Hoai, On synergistic interactions between evolution, development and layered learning. IEEE Trans. Evol. Comput. **15**(3), 287–312 (2011)
123. J.J. Hopfield, C.D. Brody, What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. Proc. Natl. Acad. Sci. **98**(3), 1282–1287 (2001)
124. G.S. Hornby, Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2007), pp. 1729–1736

125. G.S. Hornby, H. Lipson, J.B. Pollack, Generative representations for the automated design of modular physical robots. IEEE Trans. Robot. Autom. **19**(4), 703–719 (2003)
126. G.S. Hornby, J.B. Pollack, Creating high-level components with a generative representation for body-brain evolution. Artif. Life **8**(3), 223–246 (2002)
127. P.E. Hotz, Comparing direct and developmental encoding schemes in artificial evolution: a case study in evolving lens shapes. in *Congress on Evolutionary Computation (CEC)* (2004), pp. 752–757
128. C.-F. Hsu, Adaptive growing-and-pruning neural network control for a linear piezoelectric ceramic motor. Eng. Appl. Artif. Intell. **21**(8), 1153–1163 (2008)
129. T. Hu, W. Banzhaf, Evolvability and speed of evolutionary algorithms in light of recent developments in biology. J. Artif. Evol. Appl. **1–28**, 2010 (2010)
130. D.-S. Huang, J.-X. Du, A constructive hybrid structure optimization methodology for radial basis probabilistic neural networks. IEEE Trans. Neural Netw. **19**(12), 2099–2115 (2008)
131. A. Huemer, M. Gongora, D. Elizondo, A robust reinforcement based self constructing neural network. in *International Joint Conference on Neural Networks (IJCNN)* (2010), pp. 1–7
132. P. Husbands, T. Smith, N. Jakobi, M. O'Shea, Better living through chemistry: evolving GasNets for robot control. Connection Sci. **10**(3–4), 185–210 (1998)
133. A. Ilachinski, *Cellular Automata: A Discrete Universe* (World Scientific, 2001)
134. B. Inden, Neuroevolution and complexifying genetic architectures for memory and control tasks. Theory Biosci. **127**(2), 187–194 (2008)
135. T. Ishibashi, K. Dakin, B. Stevens, P. Lee, S. Kozlov, C. Stewart, R. Fields, Astrocytes promote myelination in response to electrical impulses. Neuron **49**(6), 823–832 (2006)
136. M.M. Islam, A. Sattar, F. Amin, Xin Yao, K. Murase, A new adaptive merging and growing algorithm for designing artificial neural networks. IEEE Trans. Syst. Man Cyber. Part B Cybern. **39**(3), 705–722 (2009)
137. H. Jäeger, H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
138. H. Jäeger, M. Lukoševičius, D. Popovici, U. Siewert, Optimization and applications of echo state networks with leaky- integrator neurons. Neural Netw. **20**(3), 335–352 (2007)
139. H. Jäeger, W. Maass, J. Principe, Introduction to the special issue on echo state networks and liquid state machines. Neural Netw. **20**(3), 287–289 (2007)
140. N. Jakobi, Harnessing morphogenesis. in *International Conference on Information Processing in Cells and Tissues* (1995), pp. 29–41
141. K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition? in *Proceedings of International Conference on Computer Vision (ICCV)* (2009), pp. 2146–2153
142. M. Joachimczak, T. Kowaliw, R. Doursat, B. Wróbel, Brainless bodies: controlling the development and behavior of multicellular animats by gene regulation and diffusive signals. in *Conference on the Simulation and Synthesis of Living Systems (ALife)*, (2012), pp. 349–356
143. M. Joachimczak, T. Kowaliw, R. Doursat, B. Wróbel, Controlling development and chemotaxis of soft-bodied multicellular animats with the same gene regulatory network. in *Advances in Artificial Life (ECAL)* (MIT Press, 2013), pp. 454–461
144. M. Joachimczak, B. Wróbel, Processing signals with evolving artificial gene regulatory networks. in *Conference on the Simulation and Synthesis of Living Systems (ALife)* (MIT Press, 2010), pp. 203–210
145. M. Joachimczak, B. Wróbel, Evolution of robustness to damage in artificial 3-dimensional development. Biosystems **109**(3), 498–505 (2012)
146. M. Kaiser, C.C. Hilgetag, A. von Ooyen, A simple rule for axon outgrowth and synaptic competition generates realistic connection lengths and filling fractions. Cereb. Cortex **19**(12), 3001–3010 (2009)
147. N. Kashtan, U. Alon, Spontaneous evolution of modularity and network motifs. Proc. Natl. Acad. Sci. **102**(39), 13773 (2005)
148. Y. Kassahun, G. Sommer, Evolution of neural networks through incremental acquisition of neural structures. Technical Report Number 0508, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Juni 2005

149. M.J. Katz, R.J. Lasek, Evolution of the nervous system: Role of ontogenetic mechanisms in the evolution of matching populations. Proc. Natl. Acad. Sci. **75**(3), 1349–1352 (1978)
150. S.A. Kauffman, *The Origins of Order: Self Organization and Selection in Evolution* (Oxford University Press, Oxford, 1993)
151. G.M. Khan, J.F. Miller, D.M. Halliday, Evolution of cartesian genetic programs for development of learning neural architecture. Evol. Comput. **19**(3), 469–523 (2011)
152. M.W. Kirschner, J.C. Gerhart, *The Plausibility of Life: Resolving Darwin's Dilemma* (Yale University Press, 2005)
153. H. Kitano, Designing neural networks using genetic algorithms with graph generation system. Complex Syst. **4**, 461–476 (1990)
154. H. Kitano, A Simple Model of Neurogenesis and Cell Differentiation based on Evolutionary Large-Scale Chaos. Artif. Life **2**, 79–99 (1995)
155. J. Kodjabachian, J.-A. Meyer, Evolution and development of neural networks controlling locomotion, gradient-following and obstacle avoidance in artificial insects. IEEE Trans. Neural Netw. **9**(5), 796–812 (1998)
156. M. Komosinski, The world of framsticks: simulation, evolution, interaction. in *Virtual Worlds* (2000), pp. 214–224
157. T. Kowaliw, W. Banzhaf, Augmenting artificial development with local fitness. in ed. by A. Tyrrell *IEEE Congress on Evolutionary Computation (CEC)* (2009), pp. 316–323
158. T. Kowaliw, W. Banzhaf, Mechanisms for complex systems engineering through artificial development. in ed. by Doursat et al. [67], pp. 331–351
159. T. Kowaliw, N. Bredeche, R. Doursat (eds.), *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks* (Springer, 2014)
160. T. Kowaliw, N. Bredeche, R. Doursat (eds.), *Proceedings of DevLeaNN: A Workshop on Development and Learning in Artificial Neural Networks* (Paris, France, 2011)
161. T. Kowaliw, P. Grogono, N. Kharma, Bluenome: A novel developmental model of artificial morphogenesis. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2004), pp. 93–104
162. T. Kowaliw, P. Grogono, N. Kharma, Environment as a spatial constraint on the growth of structural form. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2007), pp. 1037–1044
163. T. Kowaliw, P. Grogono, N. Kharma, The evolution of structural form through artificial embryogeny. in *IEEE Symposium on Artificial Life (ALIFE)* (2007), pp. 425–432
164. J.R. Koza, D. Andre, F.H Bennett III, M. Keane, *Genetic Programming 3: Darwinian Invention and Problem Solving* (Morgan Kaufman, 1999)
165. J.L. Krichmar, G.M. Edelman, Brain-based devices for the study of nervous systems and the development of intelligent machines. Artif. Life **11**(1–2), 63–77 (2005)
166. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks. in ed. by P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger *Advances in Neural Information Processing Systems 25* (2012), pp. 1106–1114
167. P. Lauret, E. Fock, T.A. Mara, A node pruning algorithm based on a fourier amplitude sensitivity test method. IEEE Trans. Neural Netw. **17**(2), 273–293 (2006)
168. A. Lazar, G. Pipa, J. Triesch, SORN: a self-organizing recurrent neural network. Front. Comput. Neurosci. **3**(23), 1–9 (2009)
169. Q. Le, A. Karpenko, J. Ngiam, A.Y. Ng, Ica with reconstruction cost for efficient overcomplete feature learning. in ed. by J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, K.Q. Weinberger *Advances in Neural Information Processing Systems 24* (2011), pp. 1017–1025
170. Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, A. Ng, Building high-level features using large scale unsupervised learning, in ed. by J. Langford, J. Pineau *Proceedings of the 29th International Conference on Machine Learning (ICML-12), ICML '12* (Omnipress, New York, 2012), pp. 81–88
171. Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and time series. in *The Handbook of Brain Theory and Neural Networks* (MIT Press, 1998)

172. J. Lefèvre, J.-F. Mangin, A reaction-diffusion model of human brain development. PLoS Comput. Biol. **6**(4) e1000749 (2010)
173. M. Li, P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications* 3rd edn. (Springer, 2008)
174. H. Lipson, Principles of modularity, regularity, and hierarchy for scalable systems. J. Biol. Phys. Chem. **7**, 125–128 (2007)
175. J. Lohn, G. Hornby, D. Linden, Evolutionary antenna design for a NASA spacecraft. in *Genetic Programming Theory and Practice II* Chap. 18 (Springer, Ann Arbor, 2004), pp. 301–315
176. R.L. Lopes, E. Costa, The regulatory network computational device. Genetic Program. Evolvable Mach. **13**, 339–375 (2012)
177. C.J. Lowe, G.A. Wray, Radical alterations in the roles of homeobox genes during echinoderm evolution. Nature **389**, 718–721 (1997)
178. S. Luke, L. Spector, Evolving graphs and networks with edge encoding : preliminary report. in *Late Breaking Papers at the Genetic Programming 1996 Conference* (1996), pp. 117–124
179. M. Lukoševičius, H. Jaeger, Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
180. W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput. **14**(11), 2531–2560 (2002)
181. J.M. Mandler, *The Foundations of Mind: Origins of Conceptual Thought* (Oxford University Press, Oxford, 2004)
182. H. Markram, A brain in a supercomputer. www.ted.com. Accessed: 27/12/2012
183. H. Markram, J. Lübke, M. Frotscher, B. Sakmann, Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. Science **275**(5297), 213–215 (1997)
184. H. Markram, The blue brain project. Nat. Rev. Neurosci. **7**(2), 153–160 (2006)
185. A. Matos, R. Suzuki, T. Arita, Heterochrony and artificial embryogeny: A method for analyzing artificial embryogenies based on developmental dynamics. Artif. Life **15**(2), 131–160 (2009)
186. C. Mattiussi, D. Floreano, Analog genetic encoding for the evolution of circuits and networks. IEEE Trans. Evol. Comput. **11**(5), 596–607 (2007)
187. J. McCormack, Aesthetic evolution of L-Systems revisited. in *Applications of Evolutionary Computing (Evoworkshops)* (2004), pp. 477–488
188. J. McCormack, *Impossible Nature: the Art of Jon McCormack*, Australian Centre for the Moving Image (2004)
189. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. **5**(4), 114–133 (1943)
190. N.F. McPhee, E. Crane, S.E. Lahr, R. Poli, Developmental plasticity in linear genetic programming. in *conference on Genetic and Evolutionary Computation (GECCO)* (2009), pp. 1019–1026
191. T. Menezes, E. Costa, Artificial brains as networks of computational building blocks. in *European Conference on Complex Systems* (2008)
192. T. Menezes, E. Costa, The gridbrain: an heterogeneous network for open evolution in 3d environments. in *IEEE Symposium on Artificial Life* (2007), pp. 155–162
193. Y. Meng, Y. Zhang, Y. Jin, Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model. IEEE Comput. Intell. Mag. **6**(1), 43–54 (2011)
194. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs* 1st–3rd edn. (Springer, New-York, 1992–1996)
195. Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, Op-elm: Optimally pruned extreme learning machine. IEEE Trans. Neural Netw. **21**(1), 158–162 (2010)
196. K.D. Micheva, B. Busse, N.C. Weiler, N. O'Rourke, S.J. Smith, Single-synapse analysis of a diverse synapse population: Proteomic imaging methods and markers. Neuron **68**(4), 639–653 (2004)
197. J.F. Miller, Evolving a self-repairing, self-regulating, french flag organism. in *Conference on Genetic and Evolutionary Computation (GECCO)* (Springer, 2004), pp. 129–139

198. J.F. Miller, Neuro-centric and holocentric approaches to the evolution of developmental neural networks. in ed. by Kowaliw et al. [160], pp. 242–268
199. J.F. Miller, W. Banzhaf, Evolving the program for a cell:fFrom french flags to boolean circuits. in *On Growth, Form and Computers* (2003), pp. 278–301
200. J.F. Miller, P. Thomson, A developmental method for growing graphs and circuits. in *Evolvable Systems: From Biology to Hardware* (2003), pp. 93–104
201. J.F. Miller, G.M. Khan, Where is the brain inside the brain? Memetic Comput. **3**, 217–228 (2011)
202. R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, U. Alon, Superfamilies of evolved and designed networks. Science **303**(5663), 1538–1542 (2004)
203. A.A. Minai, D. Braha, Y. Bar-Yam, Complex engineered systems: Science meets technology. in ed. by D. Braha, Y. Bar-Yam, A.A. Minai *Complex Engineered Systems: Science Meets Technology, Chapter Complex Engineered Systems: A New Paradigm* (Springer, 2006), pp. 1–21
204. D.E. Moriarty, *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*, Ph.D. Thesis (University of Texas at Austin, USA, 1998)
205. J.-B. Mouret, S. Doncieux, B. Girard, Importing the computational neuroscience toolbox into neuro-evolution-application to basal ganglia. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2010), pp. 587–595
206. J.-B. Mouret, P. Tonelli, Artificial evolution of plastic neural networks: a few key concepts. en ed. by Kowaliw et al. [160], pp. 269–280
207. J.-B. Mouret, S. Doncieux, MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars. Evol. Intell. **1**(3), 187–207 (2008)
208. T.D. Mrsic-Flogel, S.B. Hofer, K. Ohki, R.C. Reid, T. Bonhoeffer, M. Hbener, Homeostatic regulation of eye-specific responses in visual cortex during ocular dominance plasticity. Neuron **54**, 961–972 (2007)
209. P.L. Narasimha, W.H. Delashmit, M.T. Manry, J. Li, F. Maldonado, An integrated growing-pruning method for feedforward network training. Neurocomputing **71**(13–15), 2831–2847 (2008)
210. T. Natschläger, W. Maass, H. Markram, The "liquid computer": A novel strategy for real-time computing on time series. Spec Issue Found. Inf. Proc. TELEMATIK **8**, 39–43 (2002)
211. M.E.J. Newman, Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
212. S.A. Newman, G. Forgacs, G.B. Müller, Before programs: the physical origination of multicellular forms. Int. J. Dev. Biol. **50**, 289–299 (2006)
213. S. Nichele, G. Tufte, Genome parameters as information to forecast emergent developmental behaviors. in ed. by J. Durand-Lose, N. Jonoska *Unconventional Computation and Natural Computation (UCNC)* (Springer, 2012), pp. 186–197
214. S. Nichele, G. Tufte, Trajectories and attractors as specification for the evolution of behaviour in cellular automata. in *IEEE Congress on Evolutionary Computation (CEC)* (2010), pp. 1–8
215. M. Nicolau, M. Schoenauer, W. Banzhaf, Evolving genes to balance a pole. in ed. by A. Esparcia-Alczar, A. Ekárt, S. Silva, S. Dignum, A. Uyar *Genetic Programming, Lecture Notes in Computer Science*, vol. 6021 (Springer, Berlin, 2010), pp. 196–207
216. A.B. Nielsen, L.K. Hansen, Structure learning by pruning in independent component analysis. Neurocomputing **71**(10–12), 2281–2290 (2008)
217. K. Nigam, A.K. Mccallum, S. Thrun, T. Mitchell, Text classification from labeled and unlabeled documents using EM. Mach. Learn. **39**(2–3), 103–134 (2000)
218. S. Nolfi, O. Miglino, D. Parisi, Phenotypic plasticity in evolving neural networks. in *From Perception to Action (PerAc)* (1994), pp. 146–157
219. S. Nolfi, D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines* (MIT Press/Bradford Books, Cambridge, 2000)
220. D. Norton, D. Ventura, Improving liquid state machines through iterative renement of the reservoir. Neurocomputing **73**, 2893–2904 (2010)

221. D. Norton, D. Ventura, Preparing more effective liquid state machines using hebbian learning. in *International Joint Conference on Neural Networks (IJCNN)* (2006), pp. 8359–8364
222. B.A. Olshausen, D.J. Field, Sparse coding with an overcomplete basis set: a strategy employed by V1? Vis. Res. **37**(23), 3311–3325 (1997)
223. T.H. Oong, N.A.M.M. Isa, Adaptive evolutionary artificial neural networks for pattern classification. IEEE Trans. Neural Netw. **22**, 1823–1836 (2011)
224. C. Öztürkeri, M.S. Capcarrere, Self-repair ability of a toroidal and non-toroidal cellular developmental model. in *European conference on Advances in Artificial Life (ECAL)* (Springer, 2005), pp. 138–148
225. M.E. Palmer, Evolved neurogenesis and synaptogenesis for robotic control: the L-brain model. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2011), pp. 1515–1522
226. H. Paugam-Moisy, R. Martinez, S. Bengio, Delay learning and polychronization for reservoir computing. Neurocomputing **71**(7–9), 1143–1158 (2008)
227. R. Perin, T.K. Berger, H. Markram, A synaptic organizing principle for cortical neuronal groups. Proc. Natl. Acad. Sci. **108**, 5419–5424 (2011)
228. R. Pfeifer, J. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence* (Bradford Books, 2006)
229. M. Pigliucci, Is evolvability evolvable? Nat. Rev. Genet. **9**, 75–82 (2008)
230. D.J. Price, A.P. Jarman, J.O. Mason, P.C. Kind, Building brains: an introduction to neural development. 2nd edn. (Wiley, 2009)
231. P. Prusinkiewicz, A. Lindenmayer, *The Algorithmic Beauty of Plants* (Springer, 1990)
232. W.J. Puma-Villanueva, E.P. dos Santos, F.J. Von Zuben, A constructive algorithm to synthesize arbitrarily connected feedforward neural networks. Neurocomputing **75**(1), 14–32 (2012)
233. Z.W. Pylyshyn, Is vision continuous with cognition? the case for cognitive impenetrability of visual perception. Behav. Brain Sci. **22**, 341–423 (1999)
234. S.R. Quartz, T.J. Sejnowski, H. Hughes, The neural basis of cognitive development: a constructivist manifesto. Behav. Brain Sci. **20**, 537–596 (1997)
235. R. Raina, A. Battle, H. Lee, B. Packer, A.Y. Ng, Self-taught learning: transfer learning from unlabeled data. in *ICML '07: Proceedings of the 24th International Conference on Machine Learning* (ACM, New York, 2007), pp. 759–766
236. S. Rebecchi, H. Paugam-Moisy, M. Sebag, Learning sparse features with an auto-associator. in ed. by Kowaliw et al. [160], pp. 144–165
237. T. Reil, Dynamics of gene expression in an artificial genome—implications for biological and artificial ontogeny. in *Proceedings of the 5th European Conference on Artificial Life (ECAL99), Number 1674 in Lecture Notes in Artificial Intelligence* (1999), pp. 457–466
238. J. Reisinger, R. Miikkulainen, Acquiring evolvability through adaptive representations. in *8th Conference on Genetic and Evolutionary Computation (GECCO)* (2007), pp. 1045–1052
239. J. Reisinger, R. Miikkulainen, Selecting for evolvable representations. in *7th Conference on Genetic and Evolutionary Computation (GECCO)* (2006), pp. 1297–1304
240. J. Rieffel, D. Knox, S. Smith, B. Trimmer, Growing and evolving soft robots. Artif. Life 1–20 (2012)
241. J. Rieffel, J. Pollack, The emergence of ontogenic scaffolding in a stochastic development environment. in ed. by K. Deb *Conference on Genetic and Evolutionary Computation (GECCO) of Lecture Notes in Computer Science*, vol. 3102 (Springer, 2004), pp. 804–815
242. B. Roeschies, C. Igel, Structure optimization of reservoir networks. Logic J. IGPL **18**(5), 635–669 (2010)
243. D. Roggen, D. Federici, Multi-cellular development: is there scalability and robustness to gain? in *Parallel Problem Solving from Nature (PPSN)* (2004), pp. 391–400
244. D. Roggen, D. Federici, D. Floreano, Evolutionary morphogenesis for multi-cellular systems. Genet. Program. Evol. Mach. **8**(1), 61–96 (2006)
245. D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
246. R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering. in *Proceedings of the 24th International Conference on Machine Learning, ICML '07* (ACM, New York, 2007), pp. 791–798

247. K. Sano, H. Sayama, Wriggraph: a kinetic graph model that uniformly describes ontogeny and motility of artificial creatures. in *Artificial life X: proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, vol. 10 (MIT Press, 2006), p. 77
248. L. Schramm, Y. Jin, B. Sendhoff, Redundancy creates opportunity in developmental representations. in *IEEE Symposium on Artificial Life (IEEE-ALIFE)*(2011)
249. L. Schramm, B. Sendhoff, An animat's cell doctrine. in *European Conference on Artificial Life (ECAL)* (MIT Press, 2011), pp. 739–746
250. B. Schrauwen, M. Wardermann, D. Verstraeten, J.J. Steil, D. Stroobandt, Improving reservoirs using intrinsic plasticity. Neurocomputing **71**(7–9), 1159–1171 (2008)
251. E.K. Scott, L.L. Luo, How do dendrites take their shape? Nat. Neurosci. **4**(4), 359–365 (2001)
252. S.I. Sen, A.M. Day, Modelling trees and their interaction with the environment: A survey. Comput. Graph. **29**(5), 805–817 (2005)
253. B. Sendhoff, E. Körner, O. Sporns, Creating brain-like intelligence. in ed. by Sendhoff et al. [255], pp. 1–14
254. B. Sendhoff, E. Körner, O. Sporns, H. Ritter, K. Doya (eds.), *Creating Brain-Like Intelligence* vol. 5436 (Springer, 2009)
255. S.H. Seung, Neuroscience: towards functional connectomics. Nature **471**(7337), 170–172 (2011)
256. C.W. Seys, R.D. Beers, Genotype reuse more important than genotype size in evolvability of embodied neural networks. in *9th European Conference on Advances in Artificial Life (ECAL)* (2007), pp. 915–924
257. S.K. Sharma, P. Chandra, An adaptive slope sigmoidal function cascading neural networks algorithm. in *2010 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)* (2010), pp. 531–536
258. A.A. Siddiqi, S.M. Lucas, Comparison of matrix rewriting versus direct encoding for evolving neural networks. in *IEEE International Conference on Evolutionary Computation, ICEC'98* (1998), pp. 392–397
259. M.S.M. Siddiqui, B. Bhaumik, Reaction-diffusion based model to develop binocular simple cells in visual cortex along with cortical maps. in *International Joint Conference on Neural Networks (IJCNN)* (2010), pp. 1–8
260. J. Šíma, P. Orponen, General-purpose computation with neural networks: a survey of complexity theoretic results. Neural Comput. **15**(12), 2727–2778 (2003)
261. K. Sims, Evolving virtual creatures. in *Proceedings of SIGGRAPH* (1994), pp. 15–22
262. A. Soltoggio, P. Durr, C. Mattiussi, D. Floreano, Evolving neuromodulatory topologies for reinforcement learning-like problems. *IEEE Congress on Evolutionary Computation (CEC)* (2007), pp. 2471–2478
263. O. Sporns, From complex networks to intelligent systems. in ed. by Sendhoff et al. [255], pp. 15–30
264. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
265. K.O. Stanley, R. Miikkulainen, A taxonomy for artificial embryogeny. Artif. Life **9**(2), 93–130 (2003)
266. K.O. Stanley, D.B. D'Ambrosio, J. Gauci, A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
267. T. Steiner, Y. Jin, B. Sendhoff, Vector field embryogeny. PLoS ONE **4**(12), e8177 (2009)
268. G.F. Striedter, *Principles of Brain Evolution* (Sinauer Associates, Sunderland, 2005)
269. J.L. Subirats, L. Franco, J.M. Jerez, C-mantec: a novel constructive neural network algorithm incorporating competition between neurons. Neural Netw. **26**, 130–140 (2012)
270. M. Suchorzewski, J. Clune, A novel generative encoding for evolving modular, regular and scalable networks. in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation - GECCO '11* (2011), pp. 1523–2531
271. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998)

272. H. Tanaka, L.T. Landmesser, Cell death of lumbosacral motoneurons in chick, quail, and chick-quail chimera embryos: a test of the quantitative matching hypothesis of neuronal cell death. J. Neurosci. **6**(10), 2889–2899 (1986)
273. M.E. Taylor, S. Whiteson, P. Stone, Temporal difference and policy search methods for reinforcement learning: an empirical comparison. in *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)* (2007)
274. G. Tesauro, Practical issues in temporal difference learning. Mach. Learn. **8**(3), 257–277 (1992)
275. R. Thenius, M. Dauschanand, T. Schmickl, K. Crailsheim, Regenerative abilities in modular robots using virtual embryogenesis. in *International Conference on Adaptive and Intelligent Systems (ICAIS)* (2011), pp. 227–237
276. P. Tonelli, J.-B. Mouret, On the relationships between synaptic plasticity and generative systems. in *Conference on Genetic and Evolutionary Computation (GECCO)* (2011)
277. P. Tonelli, J.B. Mouret, On the relationshipd between generative encodings, regularity, and learning abilities when encoding plastic artificial neural networks. PLoS One **8**(11), e79138 (2013)
278. T. Trappenberg, *Fundamentals of Computational Neuroscience* 2nd edn. (Oxford University Press, Oxford, 2009)
279. T. Trappenberg. A brief introduction to probabilistic machine learning and its relation to neuroscience. in ed. by Kowaliw et al. [160], pp. 62–110
280. G. Tufte, P.C. Haddow, Extending artificial development: exploiting environmental information for the achievement of phenotypic plasticity. in *Conference on Evolvable Systems: from Biology to Hardware (ICES)* (Springer, 2007), pp. 297–308
281. A. Turing, The chemical basis of morphogenesis. Philosop. Trans. Roy. Soc. B **237**, 37–72 (1952)
282. M. Ulieru, R. Doursat, Emergent engineering: a radical paradigm shift. Int. J. Auton. Adap. Commun. Syst. **4**(1), 39–60 (2011)
283. V. Valsalam, J.A. Bednar, R. Miikkulainen, Developing complex systems using evolved pattern generators. IEEE Trans. Evol. Comput. **11**(2), 181–198 (2007)
284. P. Verbancsics, K.O. Stanley, Constraining connectivity to encourage modularity in HyperNEAT. in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)* (ACM Press, New York, 2011), pp. 1483–1490
285. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. (2010)
286. C. von der Malsburg, Synaptic plasticity as basis of brain organization. in ed. by J.P. Changeux, M. Konishi *The Neural and Molecular Bases of Learning* (Wiley, 1987), pp. 411–432
287. C. von der Malsburg, The correlation theory of brain function. in *Models of Neural Networks II: Temporal Aspects of Coding and Information Processing in Biological Systems* (Springer, 1981), pp. 95–119
288. C. von der Malsburg, E. Bienenstock, Statistical coding and short-term synaptic plasticity. in *Disordered Systems and Biological Organization* (Springer, 1986), pp. 247–272
289. G.P. Wagner, M. Pavlicev, J.M. Cheverud, The road to modularity. Nat. Rev. Genet. **8**(12), 921–931 (2007)
290. V.J. Wedeen, D.L. Rosene, R. Wang, G. Dai, F. Mortazavi, P. Hagmann, J.H. Kaas, W.-Y.I. Tseng, The geometric structure of the brain fiber pathways. Science **335**(6076), 1628–1634 (2012)
291. J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, E. Thelen, Autonomous mental development by robots and animals. Science **291**(5504), 599–600 (2001)
292. J. Weng, A computational introduction to the biological brain-mind. Nat. Intell. INNS Mag. **1**(3), 5–16 (2012)
293. D.J. Willshaw, C. von der Malsburg, How patterned neural connections can be set up by self-organization. Proc. Roy. Soc. London Ser. B Biol. Sci. **194**(1117), 431–445 (1976)
294. L. Wolpert, *Developmental Biology* (Oxford University Press, Oxford, 2011)

295. L. Wolpert, Positional information and the spatial pattern of cellular differentiation. J. Theor. Biol. **1**, 1–47 (1969)
296. B. Wróbel, A. Abdelmotaleb, M. Joachimczak, Evolving spiking neural networks in the GReaNs (gene regulatory evolving artificial networks) plaftorm. in *EvoNet2012: Evolving Networks, from Systems/Synthetic Biology to Computational Neuroscience Workshop at Artificial Life XIII* (2012), pp. 19–22
297. B. Wróbel, M. Joachimczak, Using the GReaNs (genetic regulatory evolving artificial networks) platform for signal processing, animat control, and artificial multicellular development. in ed. by Kowaliw et al. [160], pp. 198–214
298. H. Yamada, T. Nakagaki, R.E. Baker, P.K. Maini, Dispersion relation in oscillatory reaction-diffusion systems with self-consistent flow in true slime mold. J. Math. Biol. **54**(6), 745–760 (2007)
299. S.-H. Yang, Y.-P. Chen, An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications. Neurocomputing **86**, 140–149 (2012)
300. Yann LeCun, J.S. Denker, S. Solla, R.E. Howard, L.D. Jackel, Optimal brain damage. in ed. by D. Touretzky *NIPS'89* (Morgan Kaufman, 1990)
301. X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks. IEEE Trans. Neural Netw. **8**, 694–713 (1997)
302. X. Yao, Evolving neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)
303. I.B. Yildiz, H. Jaeger, S.J. Kiebel, Re-visiting the echo state property. Neural Netw. **35**, 1–9 (2012)
304. J. Yin, Y. Meng, Y. Jin, A developmental approach to structural self-organization in reservoir computing. IEEE Trans. Auton. Ment. Dev. **4**(4), 273–289 (2012)
305. T. Yu, J. Miller, Neutrality and the evolvability of boolean function landscape. in ed. by J. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A. Tettamanz, W.B. Langdon *Genetic Programming* (Springer, 2001), pp. 204–217
306. C. Yu, M.T. Manry, J. Li, An efficient hidden layer training method for multilayer perceptron. Neurocomputing **70**(1–3), 525–535 (2006)
307. B. Zhang, D.J. Miller, Y. Wang, Nonlinear system modelling with random matrices: echo state networks revisited. IEEE Trans. Neural Netw. Learn. Syst. **23**(1), 175–182 (2012)
308. R. Zhang, Y. Lan, G.-B. Huang, Z.-B. Xu, Universal approximation of extreme learning machine with adaptive growth of hidden nodes. IEEE Trans. Neural Netw. Learn. Syst. **23**(2), 365–371 (2012)
309. P. Zheng, C. Dimitrakakis, J. Triesch, Network self-organization explains the distribution of synaptic efficacies in neocortex. in ed. by Kowaliw et al. [159], pp. 8–9
310. N.E. Ziv, C.C. Garner, Principles of glutamatergic synapse formation: seeing the forest for the trees. Current Opin. Neurobiol. **11**(5), 536–543 (2001)
311. F. Zubler, A. Hauri, S. Pfister, A.M. Whatley, M. Cook, R. Douglas, An instruction language for self-construction in the context of neural networks. Front. Comput. Neurosci. **5**(57), 1–15 (2001)

Chapter 2
A Brief Introduction to Probabilistic Machine Learning and Its Relation to Neuroscience

Thomas P. Trappenberg

Abstract My aim in this chapter is to give a concise summary of what I consider the most important ideas in modern machine learning, and relate to one another different approaches, such as support vector machines and Bayesian networks, or reinforcement learning and temporal supervised learning. I begin with general comments on organizational mechanisms, then focus on unsupervised, supervised and reinforcement learning. I point out the links between these concepts and brain processes such as synaptic plasticity and models of the basal ganglia. Examples for each of the three main learning paradigms are also included to allow experimenting with these concepts.

1 Evolution, Development and Learning

Development and learning are two crucial ingredients for the success of natural organisms, and applying those concepts to artificial systems might hold the key to new breakthroughs in science and technology. This chapter is an introduction to machine learning that illustrates its links with neuroscientific findings. There has been much progress in this area, in particular by realizing the importance of representing uncertainties and the corresponding usefulness of a probabilistic framework.

1.1 Organizational Mechanisms

Before focusing on the main learning paradigms that dominate much of our recent thinking in machine learning, I would like to briefly outline some of my views on

Available at http://projects.cs.dal.ca/hallab/MLreview2013.

T. P. Trappenberg (✉)
Dalhousie University, Halifax, Canada
e-mail: tt@cs.dal.ca

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_2,

the close relationships that exist among the organizational mechanisms discussed in this volume. It seems to me that at least three levels of these mechanisms contribute to the success of living organisms: evolutionary mechanisms, developmental mechanisms and learning mechanisms. *Evolutionary mechanisms* focus on the long-term search for suitable architectures. This search takes time, usually many generations, to establish small modifications that are beneficial for the survival of a species, and even longer to branch off new species that can exploit niches in the environment. Evolution is by essence adaptive, as it depends on the environment, the physical space, and other organisms. A good basic organization and good choices by an organism ultimately determine the survival of the individuals, hence the species in general.

While evolution works on the general architectural level of the population, a precise architecture has to be realized in individuals, too. This is where *development* comes into play. The genetic code is used to grow specific organisms from a master plan (the genome) and environmental conditions. Thus, this mechanism is also adaptive since the environment can influence the specific decoding of the master plan. For example, the shape and metabolism of the sockeye salmon can change drastically when environmental conditions allow migration from a freshwater environment to the ocean—whereas this fish remains small and adapted to fresh water if prevented from migrating, or if food sources are sufficient in the river. The ability to grow specific architectures in response to the environment gives organisms a considerable advantage, and these external stimuli seem to continually influence genetic expression.

Having grown a specific architecture, the resulting organisms can continue to respond to environmental conditions by *learning* about specific situations and how to take appropriate actions. Learning is another type of adaptation of a specific architecture that can take several forms. For example, it can be supervised by other individuals, such as parents teaching their offspring behavioural patterns that they find advantageous, or the organisms can learn from more general environmental feedback by receiving reinforcement signals such as food reward or the accuracy of anticipated outcomes. This chapter will focus for the most part on such learning mechanisms.

The three different adaptive frameworks outlined above are somewhat abstract at this level and it is important to be more precise about their meaning by showing specific implementations. However, this is also when distinctions between these mechanisms become somewhat blurred. For example, the emergence of receptive fields (e.g. in the visual cortex) during the critical postnatal period is definitely an important event at the developmental level, yet we will discuss such mechanisms as a special form of "learning" in this chapter. For the sake of this volume it might be useful to think about the learning processes described here as *fine-tuning* a system to specific environmental conditions, as they can be experienced by an individual during its lifetime. Other mechanisms discussed in this volume are aimed at developing better learning systems in the long term, or growing specific individuals in response to the environment.

While I will try to draw lines between development and learning, mainly to discuss approaches from different scientific camps, it is debatable that such distinctions could

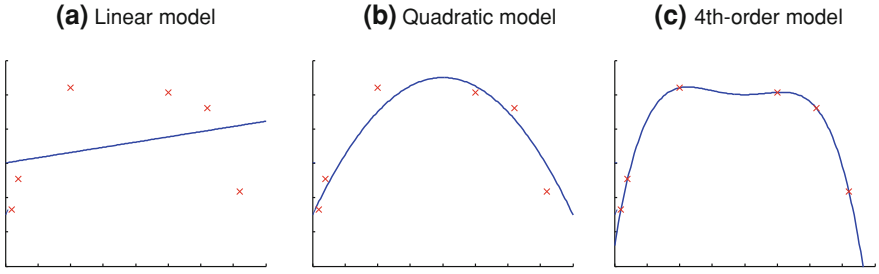

Fig. 1 Examples of underfitting (**a**) and overfitting (**c**)

even be made in the first place since, ultimately, all model implementations have to be reflected by some *morphological changes* in the system. Thus it is quite appropriate to bring together the modeling of different biological views into this volume.

1.2 Generalization

The general goal of the learning systems described here is to predict *associations*, or "labels", for future unseen data. The examples given during the learning phase are used to choose the parameters of a model that represents certain hypotheses so that a specific realization of this model can later make good predictions. The quality of generalization from training data depends crucially on the complexity of the model that is hypothesized to describe the data, as well as the number of training samples.

This is illustrated by Fig. 1. Let us think about describing the six data points shown there with a linear model: the corresponding regression curve is shown in the left-hand graph, while the other two graphs show the regression of a quadratic model and a fourth-order polynomial. Certainly, the linear model seems too low-dimensional since the data points deviate systematically, with the points in the middle trending above the curve and the points at both ends laying below the curve. Such a systematic *bias* is a clear indication that the model complexity is too low. In contrast, the curve on the right fits the data perfectly. Indeed, we can always achieve a perfect fit for a finite number of training points if the number of free parameters (one for each order of the polynomial, in this example) approaches the number of training points. But this could be *overfitting* the data in the light of possible noise. To evaluate whether we are overfitting, we need additional validation examples. An indication of overfitting is when the *variance* of this validation error grows with an increasing model complexity.

What we just discussed, called the *bias-variance tradeoff* when choosing between different potential hypotheses, is summarized in the left-hand graph of Fig. 2. Many advances in machine learning have been made by addressing ways to choose good models. While the bias-variance tradeoff has been well appreciated in the machine learning community for some time now [1], many methods are still based on general learning machines that have a large number of parameters. For such machines it is

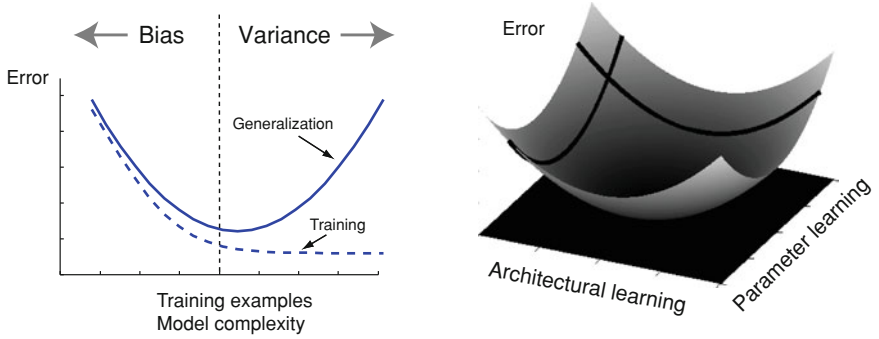

Fig. 2 Bias-variance tradeoff and explorative learning. While a training error can always decrease with increasing model complexity, minimizing the generalization error is what we are seeking. To find the smallest possible generalization error we need to search in hypothesis space and optimize in parameter space

now common to use *meta-learning* methods to address the bias-variance tradeoff, such as cross-validation where some of the training data is used to evaluate the generalization ability of the model.

We also need to consider if the model takes into account all the necessary factors that influence the outcome. How about including new features not previously considered such as a temporal domain? I believe that genetic and developmental mechanisms can address these issues by exploring a hypothesis space by ways of different model architectures. Of course, the exploration of a hypothesis space (developmental learning) must be accompanied by parameter optimization (behavioural learning) to find the best possible generalization performance. Several of the contributions in this volume represent good examples of this approach.

In summary, for the discussion in this volume it is useful to draw a distinction between two main processes:

- **Architectural exploration**: This process explores the hypothesis space in terms of *global structures*, such as what kind of features are relevant to build appropriate models and what model structures (parameterized functions) can be used.
- **Parameter optimization**: This process is about finding solutions (appropriate values of the parameters) within a specific architecture (a parameterized function).

Naturally, these processes are ultimately entwined and can be covered by common mechanisms. It remains that both aspects need to be included to find good predicitve systems, as illustrated in the right-hand graph of Fig. 2.

1.3 Learning with Uncertainties

Machine learning has recently revolutionized computer applications such as autonomous car driving or information searching. Two major ingredients have contributed to this recent success. The first was building into the system the ability to *adapt*

to unforeseen events. In other words, we must build "machines that learn", since the traditional method of encoding appropriate responses for all future situations is impossible. Like humans, machines should not be static entities that can only blindly follow orders, which might be outdated by the time real situations are encountered. Although learning machines have been studied for at least half a century, often inspired by human capabilities, the field has matured considerably in recent years through more rigorous formulations of the systems and the realization of the importance of predicting previously unseen events rather than only memorizing former events. Machine learning is now a well established discipline within artificial intelligence.

The second ingredient for the recent breakthroughs was the acknowledgment that there were *uncertainties* in the world. Thus, rather than only following the most likely explanation for a given situation, keeping an open mind and considering other possible explanations has proven to be essential in systems that have to work in a real-world environment, in contrast to a controlled lab environment. The language of describing uncertainty, that of probability theory, has proven to be elegant and tremendously simplify arguing in such worlds. This chapter is dedicated to an introduction to the *probabilistic formulation* of machine learning.

In the following sections I outline a contemporary view of learning theories that includes unsupervised, supervised and reinforcement learning. I begin with unsupervised learning since it is likely less known and relates more closely to certain developmental aspects of an organism. Then, I briefly review supervised learning in a probabilistic framework. Finally, I present reinforcement learning as an important generalization of supervised learning. In addition, I discuss some relations of these learning theories with biological analogies. This includes the relations of unsupervised learning with the development of filters in early sensory cortical areas, synaptic plasticity as the physical basis of learning, and research that relates the basal ganglia to reinforcement learning theories.

I thought important to include supervised, unsupervised and reinforcement learning in a form that would correspond to an advanced treatment of these topics in a course on machine learning. While there are now many good publications that focus on specific approaches in machine learning (such as kernel methods or Bayesian models), my aim is to link together and contrast several popular learning approaches. Most discussions of machine learning start with supervised learning, but I opted here for an initial discussion on unsupervised learning instead, as it logically precedes supervised learning and is generally less known.

1.4 Predictive Learning

Since my main research focus is neuroscience, I would like to first clarify how machine learning relates to this field. Machine learning can actually help neuroscience in many ways, one of which certainly concerns *data analysis*, as learning methods constitute the foundations of most advanced data mining techniques. Another application area, and the one examined here, is to understand the main

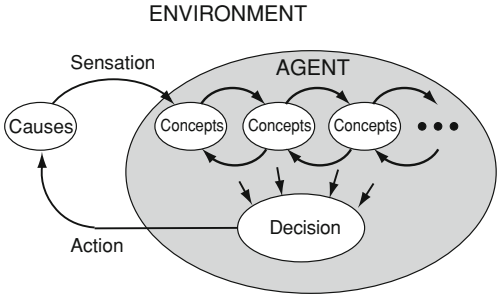

Fig. 3 The "anticipating brain" contains a hierarchical generative model of concepts and a decision system that guides behavior with the help of an anticipatory world model

problems and solutions in machine learning that can guide our understanding of biological learning systems. That is, we can ask what essential methods for solving learning problems are available, in a way somewhat reminiscent of Marr and Poggio's view of a computational, and possibly algorithmic, level of neuroscience. Similarly, many models discussed here can be construed as models of the brain on a more abstract level. Within computational neuroscience, there are also models that represent more mechanistic levels with specific representations and physical implementations. The implementation of learning via synaptic plasticity, and a more system-level model of the basal ganglia are a few of the examples mentioned later in this chapter.

If pressed to summarize what the brain does, I would say that it is an organ that represents a sophisticated decision system based on an adaptive world model. The goal of learning as it is described here is anticipation, or *prediction*. A predictive model can be used by an organism to make appropriate decisions to reach some goals. I believe that increasingly complex nervous systems evolved to make increasingly sophisticated predictions that could give them survival and evolutionary advantages.

A possible architecture of a predictive learning system resembling my high-level view of the brain is outlined in Fig. 3. An agent must interact with the environment from which it learns and receives a reward. This interaction has two sides: sensation and action. The state of the environment is conveyed by sensations that are caused by specific situations in the environment. A comprehension of these sensations requires hierarchical processing in deep-learning systems. The hierarchical processes are bidirectional so that the same structure can be used to generate expectations that should ultimately yield appropriate actions. These actions have to be guided by a decision system that itself needs to learn from the environment. This chapter reviews the principal components of such a learning system.

2 Unsupervised Learning

2.1 Representations

An important requirement for a natural or artificial agent is to decide on an appropriate course of action given specific circumstances, mainly the encountered environment.

We can treat the environmental circumstances as cues given to the agent. These cues are communicated by sensors that specify the values of certain features. Let us represent these *feature values* as a vector $\mathbf{x}$. The goal of the agent is then to calculate an appropriate response

$$y = f(\mathbf{x}). \tag{1}$$

In this review we use a probabilistic framework so that we can address uncertainties, or different possible responses. The corresponding statement of the deterministic function approximation of Eq. (1) is then to find a probability density function

$$p(y|\mathbf{x}). \tag{2}$$

A common example is object recognition where the feature values might be RGB values of pixels in a digital image and the desired response might be the identity of a person in this image. A learning machine for such a task is a model that is presented with specific examples of feature vectors $\mathbf{x}$ and their corresponding desired *labels* y. Learning under these circumstances mainly consists of adjusting the model's parameters based on the given examples. A trained machine should be able to *generalize* by predicting the appropriate labels of previously unseen feature vectors, where the "appropriateness" usually depends on the task. Since this type of learning is based on specific training examples with known labels, it is called *supervised*. We discuss specific algorithms of supervised learning and corresponding models in the next section. We start here with unsupervised learning since it is a more fundamental task that precedes supervised learning.

As stated above, the aim of learning is to find a mapping function $y = f(\mathbf{x})$ or probability density function $p(y|\mathbf{x})$. An important insight that we explore in this section is that finding such relations is much easier if the representation of the feature vector is chosen carefully [1]. For example, it is very challenging to use raw pixel values to infer the content of a digital photo such as the recognition of a face. In contrast, if we possess useful descriptions of faces, such as the distance between the eyes or other landmarks, the hair colour, nose length, and so on, it becomes much easier to classify photographs into specific target faces. Finding a useful representation of a problem is key to a successful application. When we use learning techniques for this task we talk about *representational learning*. Representational learning mostly exploits statistical characteristics of the environment without the need for labeled training examples. This is therefore an important area of *unsupervised learning*.

Representational learning itself can be viewed as a mapping problem, for example the mapping from raw pixel values to more direct features of a face. This is illustrated in Fig. 4: the raw input feature vector $\mathbf{x}$ is represented by a layer of nodes at the bottom, which we will call the *input layer*, while the feature vector $\mathbf{h}$ supporting higher order representations is represented by nodes in the upper layer of this network, which we will call the *representational layer* or *hidden layer*. The connections between the nodes represent the desired transformation between input layer and hidden layer. In line with our probabilistic framework, each node represents a random

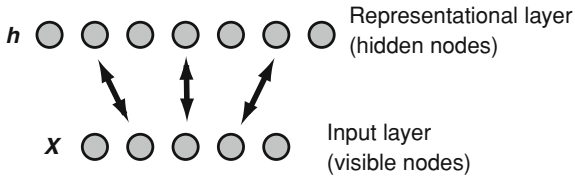

Fig. 4 A restricted Boltzmann machine is a probabilistic two-layer network with bidirectional symmetric connections between the input layer and the representational (hidden) layer

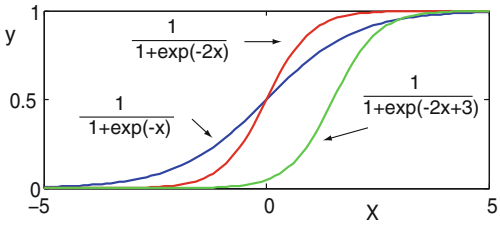

Fig. 5 Logistic function with different slopes and offsets

variable. The main idea behind the principle that we will employ to find "useful" representations is that these representations should be useful inasmuch as they can help reconstructing the input.

Before we discuss different variants of hidden representations, let us make the functions of the model more concrete. Specifically, we consider binary random variables for illustration purposes. Given the values of the input nodes (indexed by j), we choose to calculate the value of the hidden nodes (indexed by i), or more precisely their probability of having a certain value, via the logistic function shown in Fig. 5:

$$p(h_i = 1|\mathbf{x}) = \frac{1}{1 + e^{-\frac{1}{T}(\mathbf{w}_i \mathbf{x} + b_i^{\mathrm{h}})}}, \tag{3}$$

where T is a "temperature" parameter controlling the steepness of the curve, $\mathbf{w}$ are the weight values of the connections between the input and hidden layers, and b_i^{h} is the offset of the logistic function, also called the *bias* of the hidden node. In this model, which is called a "restricted Boltzmann machine" (RBM) [2], there are no connections among the hidden nodes, so these nodes represent random variables that are conditionally independent when the inputs are observed. In other words, the joint density function with fixed inputs factorizes as follows:

$$p(\mathbf{h}|\mathbf{x}) = \prod_i p(h_i|\mathbf{x}). \tag{4}$$

The connections here are bidirectional and symmetric, meaning that $w_{ij} = w_{ji}$, therefore this kind of model also represents an "undirected Bayesian network", which is a special case of the Bayesian networks that will be discussed later. Thus the state of the input nodes can be generated by hidden activities according to

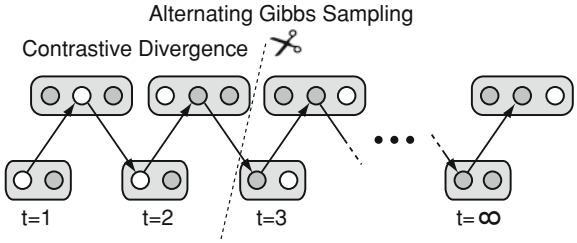

Fig. 6 Alternating Gibbs sampling and the approximation of contrastive divergence

$$p(x_j = 1|\mathbf{h}) = \frac{1}{1 + e^{-\frac{1}{T}\sum_i w_{ij} h_i + b_j^{\mathrm{v}}}}$$
$$p(\mathbf{x}|\mathbf{h}) = \prod_j \frac{1}{1 + e^{-\frac{1}{T}\sum_i w_{ij} h_i + b_j^{\mathrm{v}}}} \tag{5}$$

where b_j^{v} are the biases for each visible (input) node.

The remaining question is: how can we choose the parameters, specifically the weights and biases of the model? Since our aim is to reconstruct the observed world, we can formulate the answer in a probabilistic framework by minimizing the distance between the world's distribution (the density function of the visible nodes when set to unlabeled examples from the environment) and the generated model of the world when sampled from hidden activities. The difference between distributions is often measured by the Kullbach-Leibler divergence, denoted by D_{KL}, and minimizing this objective function with a gradient method leads to a Hebbian-type learning rule:

$$\Delta w_{ij} = \eta \frac{\partial D_{\mathrm{KL}}}{\partial w_{ij}} = \eta \frac{1}{2T} \left(\langle h_i v_j \rangle_{\mathrm{clamped}} - \langle h_i v_j \rangle_{\mathrm{free}} \right). \tag{6}$$

The angular brackets $\langle . \rangle$ denote sample averages, either in the clamped mode where the inputs are fixed or in the free running mode where the input nodes' activities are determined by the hidden nodes. Unfortunately, in practice this learning rule suffers from the long time it takes to produce an unbiased average from sequentially sampled time series. However, it turns out that learning still works for a few steps in the Gibbs sampling as illustrated in Fig. 6. This learning rule, which has finally made Boltzmann machines applicable, is called *contrastive divergence* [3] (see also [4]).

An example of a basic restricted Boltzmann machine is given in Table 1. This RBM has $n_{\mathrm{h}} = 100$ hidden nodes and is trained for $n_{\mathrm{epochs}} = 150$ epochs, where one epoch consists of presenting all images once. The network is trained with contrastive divergence in the next block of code. The training curve, which shows the average error of recall of patterns, is shown on the left in Fig. 7. After training, 20 % of the bits of the training patterns are flipped and presented as input to the network, then the program plots the patterns after repeated reconstructions as displayed on the right side of Fig. 7. Only the first 5 letters are shown here, but this number can be increased to inspect more letters.

Table 1 Basic restricted Boltzmann machine for learning letter patterns

```
clear; nh = 100; nepochs = 150; lrate = 0.01;
%load data from text file and rearrange into matrix
load pattern1.txt;
letters = permute(reshape(pattern1, [12 26 13]), [1 3 2]);

%train rbm for nepochs presentations of the 26 letters
input = reshape(letters, [12*13 26])
vb = zeros(12*13, 1); hb = zeros(nh, 1); w = .1*randn(nh, 12*13);

figure; hold on;
xlabel 'epoch'; ylabel 'error'; xlim([0 nepochs]);
for epoch = 1:nepochs;
  err = 0;
  for i = 1:26
    %sample hidden units given input, then reconstruct
    v = input(:, i);
    h = 1./(1 + exp(-(w *v + hb))); %sigmoidal activation
    hs = h > rand(nh, 1); %probabilistic sampling
    vr = 1./(1 + exp(-(w'*hs + vb))); %input reconstruction
    hr = 1./(1 + exp(-(w *vr + hb))); %hidden reconstruction

    %contrastive divergence rule: dw = h*v - hr*vr
    dw = lrate*(h*v'-hr*vr'); w = w + dw;
    dvb = lrate*(v - vr); vb = vb + dvb;
    dhb = lrate*(h - hr); hb = hb + dhb;
    err = err + sum((v-vr).^2); %reconstruction error
  end
  plot(epoch, err/(12*13*26), '.'); drawnow; %figure output
end

%plot reconstructions of noisy letters
r = randomFlipMatrix(round(.2*12*13)); %(20% of bits flipped)
noisy_letters = abs(letters - reshape(r, [12 13 26]));
recon = reshape(noisy_letters, 12*13, 26); %put data in matrix
recon = recon(:, 1:5); %plot only first 10
figure; set(gcf, 'Position', get(0, 'screensize'));

for i = 0:3
  for j = 1:5
    subplot(3 + 1, 5, i*5 + j);
    imagesc(reshape(recon(:, j), [12 13])); %plot
    colormap gray; axis off; axis image;

    h = 1./(1 + exp(-(w *recon(:, j) + hb))); %compute hidden
    hs = h > rand(nh, 1); %sample hidden
    recon(:, j) = 1./(1 + exp(-(w'*hs + vb))); %compute visible
    recon(:, j) = recon(:, j) > rand(12*13, 1); %sample visible
  end
end

function r = randomFlipMatrix(n);
%return matrix with components 1 at n random positions
r = zeros(156, 26);
for i = 1:26
  x = randperm(156);
  r(x(1:n), i) = 1;
end
```

This network is used to learn digitized letters of the alphabet that are provided in the file `pattern1.txt` at http://www.cs.dal.ca/~/repository/MLintro2012 together with the other programs of this chapter

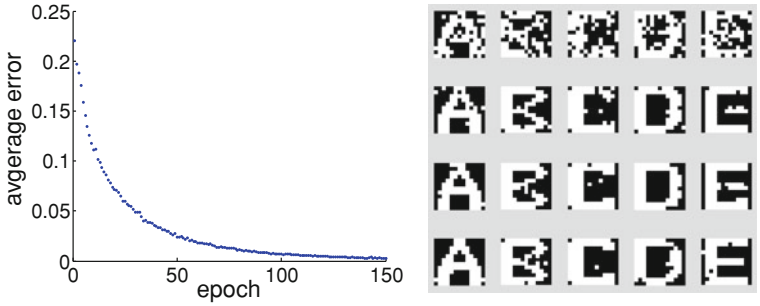

Fig. 7 Output of the example program for a restricted Boltzmann machine. *Left* learning curve showing the evolution of the average reconstruction error. *Right* reconstructions of noisy patterns after training

2.2 Sparse and Topographic Representations

In the previous section we reviewed a basic probabilistic network that implements representational learning based on the reconstruction of inputs. There are many other unsupervised algorithms that can achieve representational learning, such as non-probabilistic recurrent networks (for example, see Rebecchi et al. in this volume). Also, many other representational learning algorithms originate from signal processing, such as Fourier transform, wavelet analysis, or independent component analysis (ICA). Indeed, most advanced signal processing methods include steps to re-represent or decompose a signal into basis functions. For example, the Fourier transform decomposes a signal into sine waves with different amplitudes and phases. The original signal can then be reconstructed from the sum of individual sine waves weighted by their amplitude parameters. An example is shown in Fig. 8. The signal in the upper left is made out of three sine waves as revealed by the power spectrum on the right, which plots the square of the corresponding coefficients.

The Fourier transform has been very useful in describing periodic signals, but one problem with this representation is that an infinite number of basis functions are needed to represent a signal that is localized in time. An example of a square signal localized in time is shown in the lower left panel of Fig. 8 together with its power spectrum on the right. In the case of the time-localized signal, the power spectrum shows that a continuous interval of frequencies is necessary to accurately represent the original signal. Thus, a better choice for applications with localized features would be basis functions that are localized in time. Examples are wavelet transforms [5] or the Huang-Hilbert transform [6]. The usefulness of a specific transformation depends of course on the nature of the signals. Periodic signals with few frequency components, such as the rhythm of the heart or yearly fluctuations of natural events, are well represented by Fourier transforms, while signals with localized features, such as objects in a visual scene, are often well represented with wavelets. The main reason for calling a representation "useful" is that the original signal can be represented with

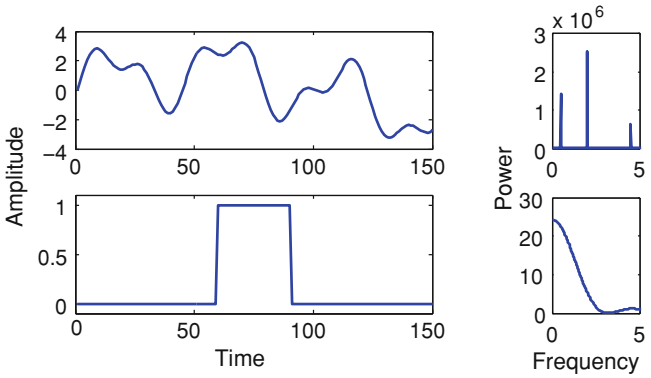

Fig. 8 Decomposition of signals into sine waves. The example signals are shown on the *left side*, and the corresponding description of the power spectrum on the *right*. The power spectrum shows the square of the amplitude for each contributing sine wave with specified frequency

only a small number of basis functions—in other words, when only a small number of coefficients have significantly large values. Therefore, even if the dictionary is large, each example of a signal from the specific environment can be represented with a small number of components. Such representations are called *sparse*.

The importance of sparse representations in the visual system has long been pointed out by Horace Barlow [7], and one of the best and probably first examples that demonstrate such mechanisms was give by his student Peter Földiák [8] (see also [9]). Another very influential article by Olshausen and Field [10] demonstrated that sparseness constraints are essential in learning basis functions that resemble receptive fields in the primary visual cortex, and similar concepts should also hold for higher-order representations in deep-belief networks [11]. It is now argued that such unsupervised mechanisms resemble receptive fields of simple cells.

The major question is then how to find good (sparse) representations for specific environments. One solution is to learn representations by unsupervised training as demonstrated above with the example of a Boltzmann machine. To learn sparse representations we now add additional constraints that force the learning of specific basis functions. In order to do this we can keep track of the mean activation of the hidden nodes by setting

$$q_i(t) = (1 - \lambda)q_i(t - 1) + \lambda h_i(t), \tag{7}$$

where parameter λ determines the averaging window. We then add to the learning rule the constraint of minimizing the difference between the *desired sparseness* ρ and the *actual sparseness* q, expressed by

$$\Delta w_{ij} \propto v_j(h_i + \rho - q_i) - v_j^r h_i^r. \tag{8}$$

This works well in practice and has the extra advantage of preventing *dead nodes* [4].

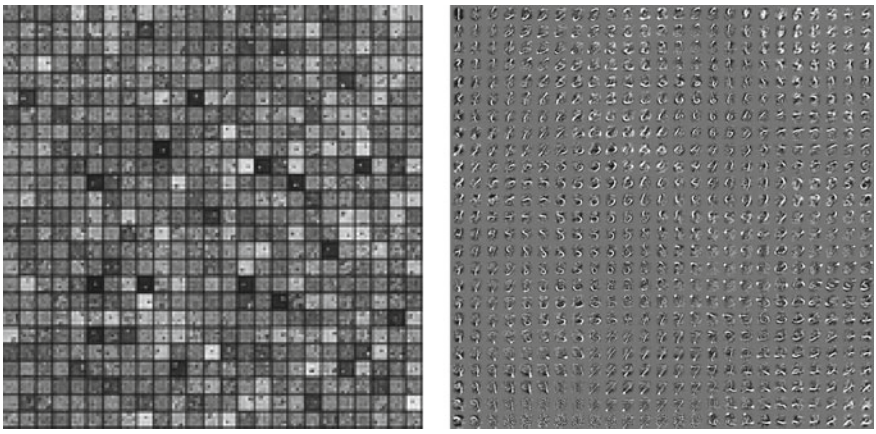

Fig. 9 Examples of learned receptive fields of a RBM without (*left*) and with (*right*) sparse and topographic constraints

In addition to the typical form of receptive fields, many brain areas show some *topographic organization* in that neurons with adjacent features of receptive fields are located in adjacent tissues. An example of unsupervised topographic representations are "self-organizing projections" [12, 13] or "self-organizing maps" (SOMs) [14]. Topographic self-organization can be triggered by lateral interactions with local facilitation and distant competition, as can be implemented with pairwise local excitation and distant inhibition between neurons. Such interactions also promote sparse representations. Along these lines, my student Paul Hollensen together with my collaborator Pitoyo Hartono and myself proposed to include lateral interactions within the hidden layer [15] as follows:

$$p(\hat{h}_i|\mathbf{v}) = \sum_j \mathcal{N}_{ij}\ p(h_j|\mathbf{v}), \tag{9}$$

where i, j both represent hidden units here, and $\mathcal{N}_{ij}$ is a kernel such as a shifted Gaussian or a Mexican-hat function centered on hidden node i. For binary hidden units the natural measure of the difference in distributions is the cross entropy, for which the derivative with respect to the weights is simply $(\hat{h}_i - h_i) \cdot \mathbf{v}$. Combining this with the contrastive divergence update yields

$$\Delta w_{ij} \ \propto\ v_j h_i - v_j^r h_i^r + v_j(\hat{h}_i - h_i) \ =\ v_j \hat{h}_i - v_j^r h_i^r. \tag{10}$$

Figure 9 presents examples of receptive fields learned with (right) and without (left) sparse topographic learning.

While purely bottom-up driven SOMs have dominated the thinking in this field, it is also important to consider models with top-down guidance of self-organized feature representations. An excellent example is the Adaptive Resonance Theory

(ART) of Stephen Grossberg [13, 16], which is most relevant in a biological context and even addresses the stability-plasticity dilemma. Further aspects of top-down control in SOMs are discussed in [17].

2.3 Hierarchical Representations and Deep Learning

Before leaving our discussion about representational learning, I would like to mention at least briefly the importance of hierarchical representations. So far we have only considered one layer of internal representations that we called the hidden layer. However, it is widely believed that representations that allow abstractions at different levels are essential to enable the cognitive abilities displayed by humans.

An obvious example consists of stacking Boltzman machines so that the hidden layer of one Boltzman machine becomes the input layer to the next Boltzman machine. This already has the advantage that more complex filters can be built from filters learned in previous levels. For example, if a first layer represents edges in a visual scene, a higher level could represent corners or more elaborate combinations of edges.

However, just obtaining more elaborate filters might not be the only advantage derived from hierarchical representations. In order to enable more advanced cognitive abilities, such as exploiting more general concepts or making higher-level plans, we need to enable more abstract representations of concepts. Such *deep learning* algorithms are the subject of much recent research in machine learning, and the chapter by Joshua Bengio is an excellent discussion of some of the challenges in this area.

Deep learning structures also resemble better the situation of the brain as a learning machine. We have mentioned above that filters in the early sensory areas and higher levels of the cortex, such as neurons in the inferotemporal cortex [18], are known to respond to more complex patterns. But it is also known that the prefrontal cortex contributes to high-level cognition functions such as planning and other executive functions that are often based on abstract concepts.

3 Supervised Learning

3.1 Regression

Representational learning is about learning a mapping function that transforms a signal (input vector) into a new signal (hidden vector):

$$f_h{:}\mathbf{x} \rightarrow \mathbf{h} \quad \text{(given unlabeled examples and constraints).} \tag{11}$$

Weight (in pounds)	Time of one-mile run (in seconds)
217	481
141	292
152	338
153	357
180	396
.	
.	
.	
245	469
141	252
177	338

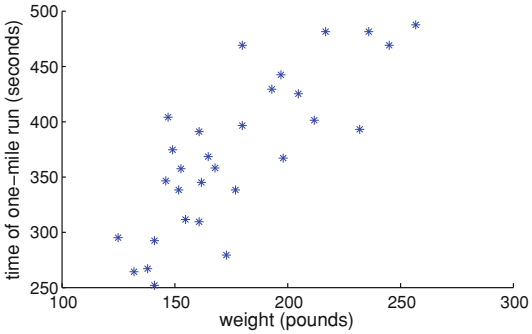

Fig. 10 Health data

The unsupervised learning of this mapping function typically exploits statistical regularities in the signals, and therefore depends on the nature of the input signals. This learning process is also guided by principles such as good reconstruction abilities, sparseness and topography. Supervised learning, on the other hand, is about learning an unknown mapping function from labeled examples:

$$f_y{:}\mathbf{h} \rightarrow \mathbf{y} \quad \text{(given labeled examples)}. \tag{12}$$

We have indicated in the formula above that supervised learning takes the hidden representation of examples, $\mathbf{h}$ and maps them to a desired output vector $\mathbf{y}$. This assumes that representational learning is somewhat completed during a developmental learning phase, which is then followed by supervised learning with a teacher that supplies desired labels (output values) for given examples. It may be argued that in natural learning systems these learning phases are not as strictly separated as discussed here, but for the purpose of this tutorial it is useful to make a distinction between these two major learning components.

In our discussion of strictly supervised learning for this section, let us follow the common nomenclature in denoting input values by $\mathbf{x}$ and output values by $\mathbf{y}$. In supervised learning we consider training data that consists of example inputs and corresponding labels, that is, pairs of values $(\mathbf{x}^{(e)}, \mathbf{y}^{(e)})$, where $e = 1, ..., m$ indexes the m training examples. For instance, Fig. 10 presents a partial list and plot of the running records of 30 employees who were regular members of a company's health club [19]. Specifically, the data shows the relationship between the weight of these persons and their time in a one-mile run.

Looking at the plot seems to reveal a systematic relation between the weights and running times, with a trend for heavier individuals to be slower at running, although this is not true for everyone. Moreover, the trend appears linear. This hypothesis can be quantified as a parameterized function

$$h(x; \boldsymbol{\theta}) = \theta_0 + \theta_1 x. \tag{13}$$

This notation means that hypothesis h is a family of functions of the quantity x that includes all possible straight lines, where each line can have a different offset θ_0 (intercept with the y-axis) and slope θ_1. We typically collect parameters in a *parameter vector* denoted by $\boldsymbol{\theta}$. We only considered a single input feature x above, but we can easily generalize this to higher-dimensional problems where more input *attributes* are given. For example, there might be the amount of exercising each week that might impact the results of running times. If we make the hypothesis that this additional variable has also a linear influence on the running time, independently from the other attribute that adds or reduces the time, we can express this new hypothesis with

$$h(\mathbf{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2. \tag{14}$$

A useful trick to enable a compact notation in higher dimension with n attributes is to introduce $x_0 = 1$. We can then write the linear equations as

$$h(\mathbf{x};\boldsymbol{\theta}) = \theta_0 x_0 + ... + \theta_n x_n = \sum_j \theta_j x_j = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}. \tag{15}$$

where vector $\boldsymbol{\theta}^{\mathrm{T}}$ is the transpose of vector $\boldsymbol{\theta}$.

At this point it would be common to fit the unknown parameters $\boldsymbol{\theta}$ with methods such as a least mean squares (LMS) regression. However, I would like to frame this problem right away in a more modern probabilistic framework. The data already shows that the relations between the weight and the running time is not strictly linear, thus the main question is how we should interpret the differences. We could introduce a more complicated nonlinear hypothesis to obtain a better fit. However, this could lead to conclusions such as: increasing your weight from 180 to 200 pounds will make you run faster. While we might wish this conclusion were true, it is most certainly unwarranted. Thus, instead of making the hypothesis function more complex, we should consider other possible sources that influence this data. One is certainly that the ability to run does not only depend on the weight of a person but also on other physiological factors. However, this data does not include information about such other factors, and the best we can do (other than collecting more information) is to treat these deviations as *uncertainties*.

There are many possible sources of uncertainties such as *irreducible indeterminacy* or *epistemological limitations*. Irreducible indeterminacy might be called "true noise", as it comes from system limitations such as time constraints on measurements, other sensors' limitations, or simply laziness for collecting more information. For us, it is actually not important where these uncertainties originate; rather, we must only acknowledge the uncertain nature of the data. In this type of thinking, we treat sampled data from the outset as fundamentally stochastic, that is, sensory data can be different even in situations that we deem identical.

To model the uncertainties in this data, we look at the deviations from the mean. Figure 11 shows a histogram of the differences between the actual data and the hypothesized regression line. This histogram looks a bit like one sampled from

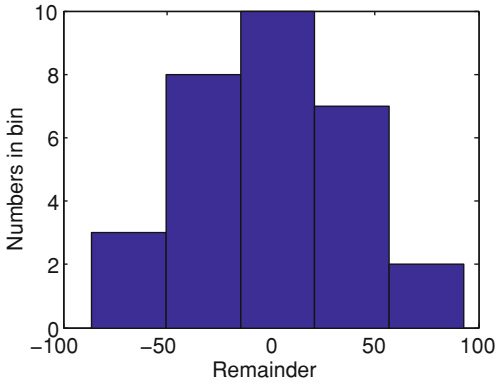

Fig. 11 Histogram of the differences between the data points and the fitted hypothesis, $(y - \theta_0 - \theta_1 x)$

Gaussian data, which is a frequent finding in many situations though not necessarily the only one. In any case, let us just make this additional assumption that there is noise in the data. With this conjecture, we should revise our hypothesis in a probabilistic framework. More precisely, we acknowledge that we can only give a probability of finding certain values. Specifically, we assume here that the data follows a certain trend $h(\mathbf{x}; \boldsymbol{\theta})$ with an *additive noise* denoted η,

$$p(y|\mathbf{x}; \boldsymbol{\theta}) = h(\mathbf{x}; \boldsymbol{\theta}) + \eta, \tag{16}$$

where the random variable η comes from a Gaussian (normal) distribution $\mathcal{N}$ in the above example, i.e.,

$$p(\eta) = \mathcal{N}(\mu, \sigma). \tag{17}$$

We can then also write the probabilistic hypothesis in the above example as a Gaussian model with a mean that depends on the variable $\mathbf{x}$:

$$\begin{aligned} p(y|\mathbf{x}; \boldsymbol{\theta}) &= \mathcal{N}(\mu = h(\mathbf{x}; \boldsymbol{\theta}), \sigma) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})^2}{2\sigma^2}\right). \end{aligned} \tag{18}$$

This function defines the probability of an y value, given an input $\mathbf{x}$ and parameters $\boldsymbol{\theta}$. We have here treated the variance σ^2 as given, although it, too, could be part of the model parameters that need to be estimated. Specifying a model with a density function is an important step in modern modeling and machine learning.

We have thus far made a parameterized hypothesis underlying the nature of the data. We now need to estimate values for the parameters to make real predictions. Therefore, let us consider again the examples of input-output pairs, i.e. our training set $\{(x^{(e)}, y^{(e)}); e = 1, ..., m\}$ (in 1D). The important principle that we will follow now is to choose the parameter θ so that the examples we have are most likely covered by

the model. This is called *maximum likelihood estimation*. To formalize this principle, we need to think about how to combine probabilities for several observations. If the observations are independent, then the joint probability of several observations is the product of the individual probabilities:

$$p(Y_1, Y_2, ..., Y_m|X_1, X_2, ..., X_m; \theta) = \prod_{e=1}^{m} p(Y_e|X_e; \theta). \tag{19}$$

Note that the Y_i's are still random variables in the above formula. We now use our training examples as specific observations (point estimates) for each of these random variables, and introduce the *likelihood function*

$$L(\theta) = \prod_{e=1}^{m} \hat{p}(\theta;\ y^{(e)},\ x^{(e)}). \tag{20}$$

Here, on the right-hand side, $\hat{p}$ is not a density function but a regular function of parameter θ (with the same functional form as our parameterized hypothesis p) for the given values $y^{(e)}$ and $x^{(e)}$. Instead of evaluating this large product, however, it is common to use the logarithm of the likelihood function, so that we can use the sum over the training examples:

$$l(\theta) = \log L(\theta) = \sum_{e=1}^{m} \log(\hat{p}(\theta;\ y^{(e)},\ x^{(e)})). \tag{21}$$

Since the log function is strictly monotonically increasing, the maximum of L is also the maximum of l. The maximum (log-)likelihood estimate (MLE) of the parameter can thus be calculated from the examples by

$$\theta^{\text{MLE}} = \arg\max_{\theta} l(\theta). \tag{22}$$

In some cases, we can calculate this analytically or we can use a search algorithm to find an approximation.

Let us now apply this strategy to the regression of a linear function with Gaussian noise as discussed above. The log-likelihood function for this example is given by

$$\begin{aligned} \hat{p}(\theta;\ y^{(e)},\ x^{(e)}) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y^{(e)} - \theta x^{(e)})^2}{2\sigma^2}\right) \\ \Rightarrow\quad l(\theta) &= -\frac{m}{2}\log 2\pi\sigma - \sum_{e=1}^{m} \frac{(y^{(e)} - \theta x^{(e)})^2}{2\sigma^2}. \end{aligned} \tag{23}$$

Since the first term on the right-hand side of Eq. (23) is independent of θ, and since we considered here a model with a given variance σ^2 for the data, maximizing the log-likelihood function is equivalent to minimizing a quadratic error term

$$E = \frac{1}{2}(y - h(\mathbf{x}; \boldsymbol{\theta}))^2 \quad \Leftrightarrow \quad p(y|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y - h(\mathbf{x}; \boldsymbol{\theta}))^2}{2}\right) \tag{24}$$

(switching the notation back to the multidimensional case). Thus, the MLE of a Gaussian dataset corresponds to minimizing a quadratic cost function, as it was commonly used in LMS regression. LMS regression is well motivated for Gaussian data, but our derivation also shows that data with non-Gaussian noise should be fitted with different cost functions. For example, a *polynomial error function* corresponds more generally to a density model of the form

$$E = \frac{1}{p}||y - h(\mathbf{x}; \boldsymbol{\theta})||^p \quad \Leftrightarrow \quad p(y|\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{2\Gamma(1/p)} \exp(-||y - h(\mathbf{x}; \boldsymbol{\theta})||^p). \tag{25}$$

Later we will mention the ε-*insensitive error function*, where errors less than a constant ε do not contribute to the error measure:

$$E = ||y - h(\mathbf{x}; \boldsymbol{\theta})||_\varepsilon \quad \Leftrightarrow \quad p(y|\mathbf{x}; \boldsymbol{\theta}) = \frac{p}{2(1-\varepsilon)} \exp(-||y - h(\mathbf{x}; \boldsymbol{\theta})||_\varepsilon). \tag{26}$$

Since we already acknowledged that we expected noisy data, it is logical not to count some amount of deviation from the expectation as error. It also turns out that this last error function is often more robust than other error functions, especially for datasets that contain outliers.

3.2 Classification as a Logistic Regression

We have grounded supervised learning in probabilistic function regression and maximum likelihood estimation. An important special instance of supervised learning is *classification*, and the simplest case is binary classification which corresponds to data that has only two possible labels, such as $y \in \{0, 1\}$.

More formally, let us consider a random number that takes value 1 with probability ϕ and value 0 with probability $1 - \phi$. Such a random variable is called a *Bernoulli distribution*. Tossing a coin is a good example of a process that generates a Bernoulli random variable, and we can use maximum likelihood estimation to estimate the parameter ϕ from such trials. For example, if we consider m tosses of a coin, the log-likelihood of finding h heads ($y = 1$) and $m - h$ tails ($y = 0$) is

$$\begin{aligned} l(\phi) &= \log(\phi^h (1 - \phi)^{m-h}) \\ &= h \log(\phi) + (m - h) \log(1 - \phi). \end{aligned} \tag{27}$$

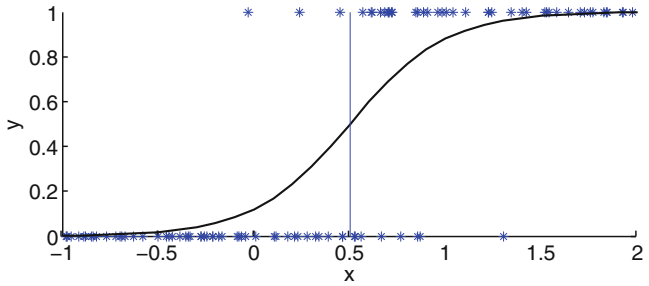

Fig. 12 Binary random numbers (*stars*) drawn from the density $p(y=1) = 1/(1+\exp(-\theta_0-\theta_1 x))$ (*solid line*) with offset $\theta_0 = -2$ and slope $\theta_1 = 4$

To find the maximum of l with respect to ϕ, we set the derivative of l to zero:

$$\begin{aligned} \frac{dl}{d\phi} &= \frac{h}{\phi} - \frac{m-h}{1-\phi} = 0 \\ \Rightarrow \phi &= \frac{h}{m}. \end{aligned} \tag{28}$$

As you might have expected, the MLE of parameter ϕ is the fraction of heads in m trials.

Let us now discuss the case when the probability of observing a head or tail, the parameter ϕ, depends on some attribute x, as usual in a stochastic way. An example is illustrated in Fig. 12 with 100 examples plotted as star symbols. The data suggests that it is far more likely that the class is $y = 0$ for smaller (possibly negative) values of x, and $y = 1$ for larger values of x. They also show that the transition between the low and high probability region is smooth. We can qualify this hypothesis by a parameterized density function p known as a *logistic* (sigmoidal) function:

$$p(y=1) = \frac{1}{1+\exp(-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})}. \tag{29}$$

As before, we can then treat this density as a function of the parameters $\boldsymbol{\theta}$ for the given data values (likelihood function), and apply MLE to estimate the values of the parameters for which the data is most likely.

How can we use the knowledge (estimate) of the density function to perform classification? The obvious choice is to predict the class with the highest probability, given the input attribute. This *Bayesian decision point*, denoted by $\mathbf{x}_d$, is characterized by

$$\begin{aligned} p(y=1|\mathbf{x}_d) = p(y=0|\mathbf{x}_d) &= 0.5 \\ \Leftrightarrow \quad \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}_d &= 0, \end{aligned} \tag{30}$$

where the last expression is called the *dividing hyperplane*.

We looked here at binary classification with linear decision boundaries as a logistic regression, but we could also generalize this method to problems where hypotheses have different functional forms, creating nonlinear decision boundaries. However, coming up with specific functions for boundaries is often difficult in practice, and we will discuss more practical methods for binary classification later in this chapter.

3.3 Multivariate Generative Models and Probabilistic Reasoning

We have so far only considered very simple hypotheses appropriate for the low dimensional data given in the above examples. An important issue that has to be considered in machine learning is generalizing to more complex nonlinear data in high-dimension, that is, when many factors interact in a complicated way. This topic is probably one of the most important when applying machine learning to real world data. This section discusses a useful way of formulating more complicated stochastic models with causal relations and how to use such models to argue, i.e. do inference.

Let us consider high-dimensional data and the corresponding supervised learning problem which is simply a generalization of our discussions above. In the probabilistic framework, this means making a hypothesis of joint density function for the problem:

$$p(y, \mathbf{x}) = p(y, x_1, x_2, ...|\boldsymbol{\theta}), \tag{31}$$

where $y, x_1, ...$ are random variables and $\boldsymbol{\theta}$ represents the parameters of the model. With this joint density function we could argue about every possible situation in the environment. For example, we could request classification or object recognition by calculating the conditional density function

$$p(y|\mathbf{x}) = p(y|x_1, x_2, ...; \boldsymbol{\theta}). \tag{32}$$

Of course, the general joint density function and even this conditional density function for high-dimensional problems typically have many free parameters that we need to calculate with MLE. Thus it is useful to make more careful assumptions of causal relations that would restrict the density functions.

The object recognition formulation above is sometimes called a *discriminative approach* to object recognition because it tries to discriminate labels given the feature values. Another approach is to consider modeling the inverse conditional density

$$p(\mathbf{x}|y) = p(x_1, x_2, ...|y; \boldsymbol{\theta}). \tag{33}$$

This is called a *generative model* as it can generate examples from a class, given its label. To use generative models in classification or object recognition we can apply Bayes' rule and calculate a discriminative model. It means relying on *class priors* (the relative frequencies of the classes) to calculate the probability that an item with features $\mathbf{x}$ belongs to a class y:

$$p(y|\mathbf{x};\boldsymbol{\theta}) = \frac{p(\mathbf{x}|y;\boldsymbol{\theta})p(y)}{p(\mathbf{x})}. \tag{34}$$

While using generative models for classification seems to be much more elaborate, there are several reasons that make generative models attractive for machine learning. For example, in many cases, features might be conditionally independent given a label, i.e. they verify

$$p(x_1, x_2, ...|y) = p(x_1|y)p(x_2|y)... . \tag{35}$$

where the indication of the parameter vector was dropped to make the formula less cluttered. Even if the independence does not strictly hold, this *naive Bayes assumption* is often useful and drastically reduces the number of parameters that must be estimated. This can be seen by factorizing the full joint density function with the chain rule

$$\begin{aligned} p(x_1, x_2, ..., x_n|y) &= p(x_n|y, x_1, ..., x_{n-1})p(x_1, ..., x_{n-1}|y) \\ &= p(x_n|y, x_1, ..., x_{n-1})...p(x_2|y, x_1)p(x_1|y) \\ &= \prod_{j=1}^{n} p(x_j|y, x_{j-1}, ..., x_1). \end{aligned} \tag{36}$$

But what if the naive Bayes assumption is not appropriate? Then we need to build more elaborate models, or *causal models*. This particular challenge has been greatly simplified with *graphical methods* that specify the conditional dependencies between random variables using graphs [20]. A well known example from one of the inventors of graphical models, Judea Pearl, is shown in Fig. 13. In graphical models, the nodes represent random variables, and the links between them represent causal relations with conditional probabilities. In the case shown here, there are arrows on the links and the graph contains no loops, which makes it an example of *directed acyclic graph* (DAG). In contrast, the RBM discussed previously was an example of undirected Bayesian network.

In Fig. 13, each of the five nodes stands for a random binary variable: Burglary $B = \{\text{yes}, \text{ no}\}$, Earthquake $E = \{\text{yes}, \text{ no}\}$, Alarm $A = \{\text{yes}, \text{ no}\}$, JohnCalls $J = \{\text{yes}, \text{ no}\}$, MaryCalls $M = \{\text{yes}, \text{ no}\}$. In general, a joint distribution of several variables can be factorized in various ways following the chain rule mentioned before Eq. (36), for example:

$$p(B, E, A, J, M) = p(B|E, A, J, M)p(E|A, J, M)p(A|J, M)p(J|M)p(M). \tag{37}$$

In this case, with binary random variables we need $2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 31$ parameters to specify the full joint density function. However, the model of Fig. 13 restricts causal relations between the random variables to represent only a subset of the factorization of the joint probability function, namely

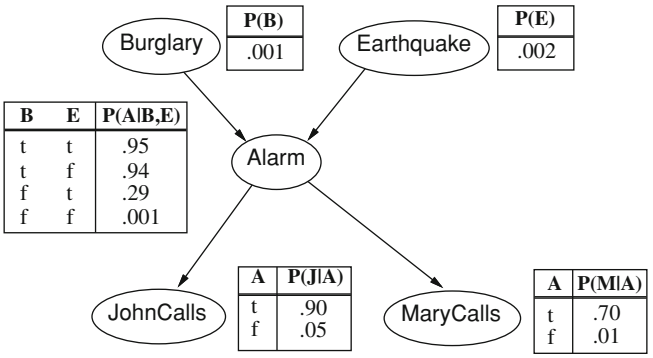

Fig. 13 Example of causal model with a two-dimensional probability density function (pdf) and a few other marginal pdf's

$$p(B, E, A, J, M) = p(B)p(E)p(A|B, E)p(J|A)p(M|A). \tag{38}$$

Therefore, we only need $1 + 1 + 2^2 + 2 + 2 = 10$ parameters to specify all the knowledge in the system. Example parameters for a specific case are displayed in the *conditional probability tables* (CPTs), which define the conditional probabilities represented by the links between the nodes. The graphical representation makes is very convenient to represent the particular hypotheses about causal relations.

The graph structure of the model also makes it easier to do inference (draw conclusions) on specific questions. For example, say we want to know the probability that there was no earthquake or burglary when the alarm rings and both John and Mary call. This is expressed by

$$\begin{aligned}
&p(B = f, E = f, A = t, J = t, M = t) \\
&= p(B = f)p(E = f)p(A = t|B = f, E = f)p(J = t|A = t)p(M = t|A = t) \\
&= 0.999 * 0.998 * 0.001 * 0.9 * 0.7 \\
&= 0.00063
\end{aligned}$$

where f stands for false and t for true. Although we have a causal model where parent variables influence the outcome of child variables, we can also use evidence from child variables to infer possible values for the parent variables. For example, let us calculate the probability that the alarm rings given that John calls, $p(A = t|J = t)$. For this we should first calculate the probability that the alarm rings as we will need this later. It is given by

$$\begin{aligned}
p(A = t) = &\; p(A = t|B = t, E = t)p(B = t)p(E = t) \\
&+ p(A = t|B = t, E = f)p(B = t)p(E = f) \\
&+ p(A = t|B = f, E = t)p(B = f)p(E = t) \\
&+ p(A = t|B = f, E = f)p(B = f)p(E = f)
\end{aligned}$$

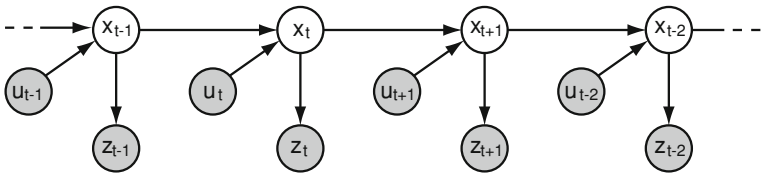

Fig. 14 A temporal Bayesian network called a Hidden Markov Model (HMM), with hidden states x_t, observations z_t, and external influences u_t

$$\begin{aligned} &= 0.95 * 0.001 * 0.002 + 0.94 * 0.001 * 0.998 \\ &\quad +0.29 * 0.999 * 0.002 + 0.001 * 0.999 * 0.998 \\ &= 0.0025. \end{aligned}$$

We can then use Bayes' rule to calculate the required probability:

$$\begin{aligned} p(A = t|J = t) &= \frac{p(J = t|A = t)p(A = t)}{p(J = t|A = t)p(A = t) + p(J = t|A = f)p(A = f)} \\ &= \frac{0.9 * 0.0025}{0.9 * 0.0025 + 0.05 * 0.9975} \\ &= 0.043 \end{aligned}$$

We can similarly apply the rules of probability theory to calculate other quantities, but these calculations can get cumbersome with larger graphs. It is therefore better to resort to numerical tools for the inference, for example a Matlab toolbox for Bayesian networks.[1]

I already mentioned the importance of learning in temporal sequences (anticipatory systems), and Bayesian networks are easily extended to this domain, where they are called *dynamic Bayesian networks* (DBN). An important example of DBN is a *hidden Markov model* (HMM), as shown in Fig. 14. In this model, a state variable x_t is not directly observed and is called a *hidden* or *latent* random variable. The "Markov condition" in this model means that each state only depends on the previous state (or states), which can include external influences denoted here by u_t. A typical example is robot localization, where a robot is driven with some motor command u_t and the goal is to estimate the new state of the robot. We can use some knowledge about the influence of the motor command on the system to calculate a new expected location, and can also combine this in a Bayesian optimal way with sensor measurements denoted by z_t. Such Bayesian models are essential in many robotics applications.

[1] Available at http://code.google.com/p/bnt/, and used to implement Fig. 13; file at www.cs.dal.ca/~tt/repository/MLintro2012/PearlBurglary.m.

3.4 Nonlinear Regression and the Bias-Variance Tradeoff

While graphical models are great to argue about situations (doing inference), the role of supervised learning is to determine the parameters of the model. We have only considered binary models where each Bernoulli variable is characterized by a single parameter ϕ. However, the density function can be much more complicated than that and introduce many more parameters. Therefore, a major problem in practice is to have enough labeled training examples to restrict useful learning appropriately. This is one important reason for unsupervised learning, as we usually have a lot of unlabeled data that can be used to learn how to represent the problem appropriately in order to simplify the task. But we still need to understand the relations between free parameters and the amount of training data.

We already discussed the bias-variance tradeoff in the first section. Finding the right function that describes nonlinear data is one of the most difficult tasks in modeling, and there is no single algorithm that can give us the answer. This is why more general learning machines, which we will discuss in the next section, are popular. To evaluate the generalization performance of a specific model, it is helpful to split the training data into a *training set*, which is used to estimate the parameters of the model, and a *validation set*, which is used to study the *generalization performance* on data that has not been included during the training of the model.

A important question then becomes how much data we should keep for validation vs. training. If we use too much data for validation, then we might end up with too little data for accurate learning in the first place. On the other hand, if we have too little data for validation, then it might not be very representative. In practice, we often use some *cross-validation* technique to minimize the tradeoff, i.e. we use most of the data for training but repeat the selection of the validation data several times to make sure that the validation was not just a result of outliers. The repeated division of the data into a training set and a validation set can be done in different ways. For example, in *random subsampling* we merely use random subsamples for each set and repeat the procedure with other random samples. More common is *k-fold cross-validation*: in this technique, we divide the data set into k subsamples and use $k-1$ subsamples for training and 1 subsample for validation. In the next round, we use another subsample to validate the training. A common choice for the number of subsamples is $k = 10$. By combining the results for the different runs we can often reduce the variance of our prediction while utilizing most data for learning.

We can sometimes help the learning process further. In many learning examples it turns out that some data is easy to learn while other data is much harder. In particular techniques called *boosting*, data that is hard to learn is oversampled in the learning set so that the machine has more opportunities to learn these examples. A popular implementation of such an algorithm is *AdaBoost* (adaptive Boosting).

Before proceeding to general nonlinear learning machines, I would like to outline a point that was eloquently made by Doug Tweet in a course module that we shared in the summer of 2012, part of a computational neuroscience program in Kingston, Canada. As discussed above, supervised learning is best phrased in terms

of regression and many applications are nonlinear in nature. It is common to make a nonlinear hypothesis under the form $y = h(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})$, where $\boldsymbol{\theta}$ is a parameter vector and h is a nonlinear function. A common example of such a model is an artificial perceptron with a sigmoidal transfer function in 1D such as $h(x;\theta) = \tanh(\theta x)$. However, as stressed by Doug, there is no reason to make the functions nonlinear *in the parameters*, which would result in a nonlinear optimization problem. Support Vector Machines (SVM; reviewed next) are a good example where the optimization error is simply quadratic in the parameters. The corresponding convex optimization has none of the local minima that plague multilayer perceptrons.

In summary, these different strategies can be expressed through the following optimization functions:

$$\text{Linear Perceptron: } E \propto \left(y - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}\right)^2 \tag{39}$$

$$\text{Nonlinear Perceptron: } E \propto \left(y - h(\mathbf{x};\boldsymbol{\theta})\right)^2 \tag{40}$$

$$\text{Linear in Parameters (LIP): } E \propto \left(y - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right)^2 \tag{41}$$

$$\text{Linear SVM: } E \propto \alpha_i\,\alpha_j\,y_i\,y_j\,\mathbf{x}^{\mathrm{T}}\mathbf{x} + \text{constraints} \tag{42}$$

$$\text{Nonlinear SVM: } E \propto \alpha_i\,\alpha_j\,y_i\,y_j\,\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + \text{constraints} \tag{43}$$

The LIP model is more general than a linear model in that it considers functions of the form $y = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$ involving some mapping function $\boldsymbol{\phi}(\mathbf{x})$. In light of this review, the transformation $\boldsymbol{\phi}(\mathbf{x})$ can thus be seen as re-coding a sensory signal into a more appropriate form using unsupervised learning methods as discussed above.

3.5 General Learning Machines

Before we leave this discussion of basic supervised learning, I would like to mention some methods that are very popular and often used in machine learning applications. In the previous section we discussed the formulation of specific hypothesis functions. However, finding an appropriate hypothesis function requires considerable domain knowledge. To some extend, this is the "hard problem" in machine learning.

Finding general learning machines has long been on the minds of researchers, and this area has been especially inspired by human abilities and the brain itself as a learning machine. A good example are artificial neural networks, in particular multilayer perceptrons, which became popular in the 1980's although they had been introduced much earlier. Boltzmann machines (discussed above) and support vector machines, which I briefly describe in this section, are also examples of this category. The overall concept behind these learning machines is to provide a very general function with many parameters that are adjusted through learning. Of course, the real problem then becomes to avoid "overfitting" the data with the model. This can

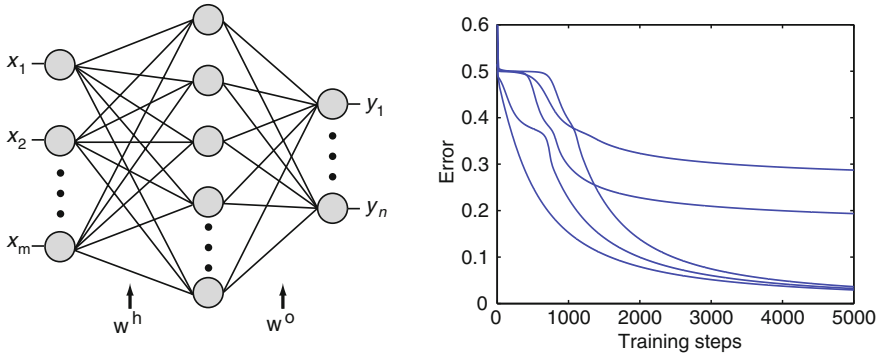

Fig. 15 Multilayer perceptron with one hidden layer. The parameters are called weights **w**. The graph on the *right* shows example training curves when trained on XOR data

be done by applying appropriate restrictions and making the learning efficient enough so that it can be used for a larger problem size.

Scientists who design different models specially tailored to the different cognitive functions and their applications point out that a general learning machine is always at a disadvantage. There is "no free lunch", they argue, meaning that we need to create specific models for specific problems. While this is true in principle, general learning machines can still be successful by providing answers where other methods are not known. In fact, these methods are currently experiencing something like a renaissance as they are now applied to massive data sets, whose size also help alleviate overfitting issues (see [21] for a recent example).

Let us begin with a multilayer perceptron (MLP) as shown in Fig. 15. Each node represents a simple calculation. The input layer relays the inputs, while the hidden (resp. output) layer multiplies each input channel x_j (resp. h_i) by an associated weight w^{h}_{ij} (resp. w^{o}_{ki}), sums these net inputs, then passes them through a transfer function, generally nonlinear, often the sigmoid curve of the logistic function. This reads:

$$y_k = g\left(\sum_i w^{\mathrm{o}}_{ki}\, g\left(\sum_j w^{\mathrm{h}}_{ij}\, x_j\right)\right). \tag{44}$$

A network of this type is a graphical representation of nested nonlinear functions with parameters **w**. Applying a particular input results in a particular output **y**, which can be compared to a desired output $\mathbf{y}'$ in supervised learning. The parameters can then be adjusted as usual in LMS regression, by minimizing the least square error $E = ||\mathbf{y} - \mathbf{y}'||^2$, typically via a gradient descent:

$$w \leftarrow w + \alpha \frac{\partial E}{\partial w}, \tag{45}$$

Table 2 A multilayer perceptron with backpropagation for solving the XOR problem

```
clear;
N_i = 2; N_h = 2; N_o = 1;
w_h = randn(N_h, N_i); w_o = randn(N_o, N_h);

%training vectors (XOR)
r_i = [0 1 0 1 ; 0 0 1 1];
r_d = [0 1 1 0];

%Updating and training network with sigmoid activation function
for trial = 1:5000;
  r_h = 1./(1 + exp(-w_h*r_i));
  r_o = 1./(1 + exp(-w_o*r_h));

  %error over all pattern
  d(trial) = 0.5*sum((r_o-r_d).^2);

  %training
  d_o = (r_o.*(1-r_o)).*(r_d-r_o);
  d_h = (r_h.*(1-r_h)).*(w_o'*d_o);
  w_o = w_o + 0.7*(r_h*d_o')';
  w_h = w_h + 0.7*(r_i*d_h')';
end
plot(d)
```

where α is a learning rate. Since $\mathbf{y}$ is a nested function of the parameters, this requires the application of the chain rule. The resulting equations appear to be "propagating back" an error term $\mathbf{y} - \mathbf{y}'$ from the output to the previous layers, and for this reason this algorithm has been termed *error-backpropagation* [22]. An example program of an MLP that learns to represent the Boolean logic XOR function is shown in Table 2, and training curves in Fig. 15.

It is easy to see that such networks are universal approximators [23], i.e. the error of the training examples can be made as small as desired by increasing the number of parameters. This can be achieved by adding hidden nodes. However, the aim of supervised learning is to make predictions, that is to minimize the generalization error and not the training error. Thus, choosing a smaller number of hidden nodes might be more appropriate for this goal. The bias-variance dilemma [1] reappears here in this specific graphical model, and years of research have been investigated in solving this puzzle. Good practical methods and research directions have been proposed to counter overfitting, such as early stopping [24], weight decay [25] or Bayesian regularization [26]. Also, transfer learning [27, 28] can be seen as biasing models beyond the current data set.

A more recent general learning machine for classification are support vector machines, which were introduced by Vapnik, Guyon and Boser in 1992 [29, 30]. These machines are fundamentally based on minimizing the estimated generalization (called the "empirical error" in this community). The main idea behind SVMs for binary classification is that the best linear classifier for a separable binary classification problem is the one that maximizes the margin between a separating classification

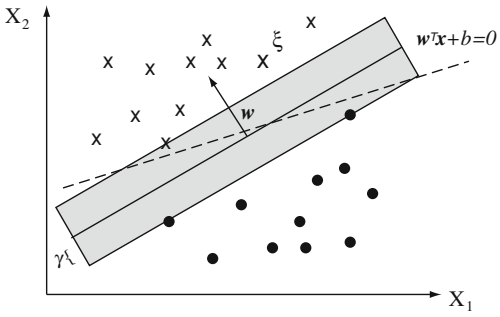

Fig. 16 Illustration of linear support vector classification

line (separating hyperplane in higher dimensions) and the nearest data points [31]. Since there are many lines that can separate the data, as shown in Fig. 16, the most robust line is expected to be positioned as far from any data point as possible, since we also expect new data to be more likely to fall near the clusters of the training data—if the training data is indeed representative of the general distribution. In the end, the separating line is determined only by a few close points that are the ones called *support vectors*.

Vapnik's important contributions did not stop there. He also formulated the margin maximization problem in a form such that the formulas are quadratic in the parameters and only contain dot products of training vectors, $\mathbf{x}^{\mathrm{T}}\mathbf{x}$ by solving the dual problem cast in a Lagrangian formalism [30]. This has several important benefits. The problem becomes a convex optimization challenge, which avoids the local minima that have crippled MLPs. Furthermore, since only dot products between example vectors appear in these formulations, it is possible to apply a so-called "kernel trick" to efficiently generalize these approaches to nonlinear functions.

Let me illustrate the idea behind using kernel functions for dot products. To do this, it is important to distinguish attributes from features as follows. Attributes are the raw measurements, whereas features can be made up by combining attributes. For example, the attributes x_1 and x_2 could be combined into a tentative feature vector $(x_1, x_2, x_1x_2, x_1^2, x_2^2)^{\mathrm{T}}$. This is a bit like trying to guess a better representation of the problem, one that should be "useful" as discussed above in the part about structural learning. Let us now denote this transformation by a function $\phi(\mathbf{x})$. The interesting part in Vapnik's formulation is that we actually do not even have to calculate this transformation explicitly, but we can replace the corresponding dot products by a *kernel function*

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{z}), \tag{46}$$

which is often much easier to calculate. For example, a Gaussian kernel function formally corresponds to an infinite-dimensional feature transformation ϕ. There are some arguments from structural learning [30, 32] why SVMs are less prone to overfitting, and extensions have also been made to problems with overlapping data in the form of soft margin classification [31]. These ideas have also been generalized to regression problems [33], but their performance is often not as satisfactory. We will

Table 3 Using LibSVM for classification

```
clear; close all; figure; hold on; axis square

%training data and training SVM
r1 = 2 + rand(300, 1); a1 = 2*pi*rand(300, 1); polar(a1, r1, 'bo');
r2 = randn(300, 1); a2 = .5*pi*rand(300, 1); polar(a2, r2, 'rx');

x = [r1.*cos(a1), r1.*sin(a1); r2.*cos(a2), r2.*sin(a2)];
y = [zeros(300, 1); ones(300, 1)];
model = svmtrain(y, x);

%test data and SVM predicition
r1 = 2 + rand(300, 1); a1 = 2*pi*rand(300, 1);
r2 = randn(300, 1); a2 = .5*pi*rand(300, 1);
x = [r1.*cos(a1), r1.*sin(a1); r2.*cos(a2), r2.*sin(a2)];
yp = svmpredict(y, x, model);

figure; hold on; axis square
[tmp, I] = sort(yp);
plot(x(1:600-sum(yp), 1), x(1:600-sum(yp), 2), 'bo');
plot(x(600-sum(yp) + 1:600, 1), x(600-sum(yp) + 1:600, 2), 'rx');
```

not dive more into the theory of Support Vector Machine but show instead an example using the popular LibSVM [34] implementation. This implementation includes interfaces to many programming languages, such as MATLAB and Python. SVMs are probably currently the most successful general learning machines for classification.

Table 3 gives an example of applying the LibSVM library to the data displayed in Fig. 17. The plot on the left is the training data, which is produced from sampling two distributions: the points in the first class (blue circles) are chosen within a ring of radius 2.0–3.0, while the points in second class (red crosses) are distributed across two quadrants. The examples are provided with their corresponding labels to the training function `svmtrain`. Similarly, the plot on the right of Fig. 17 is test data. The corresponding class labels are given to the function `svmpredict` only for the purpose of calculating the cross-validation error. For true predictions, this vector can be set to arbitrary values. The performance of this classification is around 97 % with the standard parameters of the LibSVM package. However, it is advisable to tune these parameters, for example with search methods [35].

While SVMs have had a large impact in application-oriented machine learning, more recent research works are combining ideas from SVMs (in particular kernel methods), Bayesian networks, and good old-fashioned neural networks. Such hybrid methods are now taking off, too, and starting to have another great impact—not only in research but also in many industrial domains.

4 Reinforcement Learning

As discussed above, a basic form of supervised learning is function approximation, relating input vectors to output vectors, or more generally finding density functions

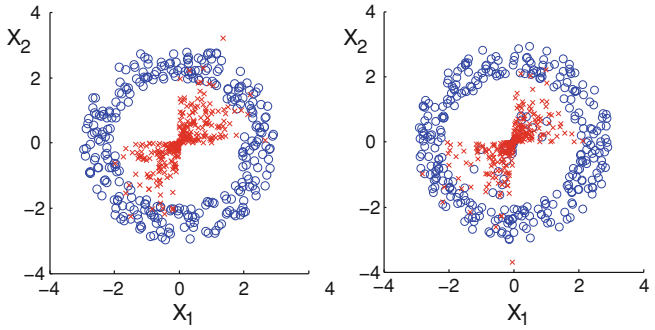

Fig. 17 Example of using training data on the *left* to predict the labels of the test data on the *right*

$p(\mathbf{y}, \mathbf{x})$ from examples $(\mathbf{x}^{(e)}, \mathbf{y}^{(e)})$. However, in many applications we do not have a teacher to tell us exactly at any time the appropriate response to a specific input. Rather, feedback from a teacher is often delayed and given only in the form of general feedback such as 'good' or 'bad', instead of a detailed explanation about what the learner should have done.

We are now turning to these more general learning problems. Specifically, we are interested in learning a sequence of appropriate actions to maximize an expected payoff. More formally, let us learn a temporal density function

$$p\big(\mathbf{y}(t+1)|\mathbf{x}(t), \mathbf{x}(t-1), ..., \mathbf{x}(1)\big). \tag{47}$$

We have already encountered such models in the form of temporal Bayesian networks. We will now discuss this issue further within the realm of *reinforcement learning* or *learning from reward*. While we mainly consider here the prediction of a scalar utility function, most of this discussion can be applied directly to a more graded environmental feedback.

4.1 Markov Decision Processes

Reinforcement learning is best illustrated in a Markovian world.[2] As discussed before, such a world is characterized by transition probabilities between states, $T(s'|s, a)$, that only depend on the current state $s \in S$ and the action $a \in A$ taken in this state. We now consider feedback from the environment in the form of a reward $r(s)$ and ask what actions should be taken in each state to maximize future reward. More formally, we define the *value function* or *utility function*

$$Q^{\pi}(s, a) = E\big[r(s) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + ...\big]_{\pi}, \tag{48}$$

[2] Markov models are often a simplification or abstraction of a real world. In this section, however, we discuss a "toy world" in which state transitions were designed to fulfill the Markov condition.

as the expected future payoff (cumulative reward) for being in state s, then s_1, s_2, etc. We introduce here the "discount factor" $0 \leq \gamma < 1$ to express that we value immediate reward over later reward. This is a common treatment to keep the expected value finite. An alternative scheme would be to consider only finite action sequences. The policy $\pi(a|s)$ describes what action can be taken in each state. In accordance with our overall probabilistic world view, we consider the most general case of probabilistic policies, i.e., we want to know with what probability a given action should be chosen. If the policy was deterministic, then taking a specific action would be determined by applying the policy to the current state, and the value function is often denoted by $V^\pi(s)$.[3] Our goal is to find the optimal policy π^*, i.e. the one that maximizes the expected future payoff Q^π:

$$\pi^*(a|s) = \arg\max_\pi Q^\pi(s,a). \tag{49}$$

This search is called a *Markov Decision Process* (MDP).

MDPs have been studied since the mid 1950s, and Richard Bellman noted that it was possible to calculate the value function for each state, and a given policy π, using a self-consistent equation now named the *Bellman equation*. He also called the corresponding algorithm *dynamic programming*. Specifically, we can separate the expected value of the immediate reward from the expected value of the reward from visiting subsequent states as follows:

$$Q^\pi(s,a) = E\big[r(s)\big]_\pi + \gamma\, E\big[r(s_1) + \gamma\, r(s_2) + \gamma^2 r(s_3) + ...\big]_\pi. \tag{50}$$

The first expected value on the right-hand side is simply the immediate reward received when reaching state s at this particular point in time. The second expected value is the function of state s_1. State s_1 is related to state s, since s_1 can be reached from s when taking action a_1 with a certain probability according to policy π (for example by setting $s_1 = s + a_1$, or more generally $s_n = s_{n-1} + a_n$). The state actually reached can also depend on stochastic environmental factors encapsulated in the matrix $T(s'|s,a)$. Incorporating these factors into the equation yields

$$Q^\pi(s,a) = r(s) + \gamma \sum_{s'} \Big(T(s'|s,a) \sum_{a'} \Big(\pi(a'|s')\; E\big[r(s') + \gamma\, r(s'_1) + \gamma^2 r(s'_2) + ...\big]_\pi \Big) \Big), \tag{51}$$

where s'_1 is the next state after state s', etc. Thus the expression on the right is the state-value-function of state s'. If we substitute it with the corresponding expression on the left of Eq. (48), we get the *Bellman equation for a specific policy*, namely

$$\textbf{Bellman-stoch-}\pi\text{:}\quad Q^\pi(s,a) = r(s) + \gamma \sum_{s'} \Big(T(s'|s,a) \sum_{a'} \Big(\pi(a'|s') Q^\pi(s',a') \Big) \Big). \tag{52}$$

[3] $V^\pi(s)$ is usually called the *state value function* and $Q^\pi(s,a)$ the *state-action value function*. Note, however, that the value depends in both cases on the states *and* the actions taken.

The variant of this equation for deterministic policies is a bit simpler. Since in this case an action a is uniquely specified by the policy, the value function $Q^\pi(s,a)$ reduces to $V^\pi(s)$ and the equation becomes[4]

$$\textbf{Bellman-det-}\pi\text{:}\quad V^\pi(s) = r(s) + \gamma \sum_{s'} \Big(T(s'|s,a) V^\pi(s') \Big). \tag{53}$$

The Bellman equation is a set of N linear equations in an environment with N states, one equation for each unknown value function of each state. The environment being given, i.e. having functions r and T, we can use well-known methods from linear algebra to solve for $V^\pi(s)$. This can be formulated compactly by matrix notation, in which s and s' are the indices:

$$\mathbf{r} = (\mathbf{I} - \gamma\, \mathbf{T})\mathbf{V}^\pi, \tag{54}$$

where $\mathbf{r}$ is the reward vector, $\mathbf{I}$ is the identity matrix, and $\mathbf{T}$ is the transition matrix. To solve this equation we have to invert a matrix and multiply this with the reward values,

$$\mathbf{V}^\pi = (\mathbf{I} - \gamma\, \mathbf{T})^{-1}\mathbf{r}^{\mathrm{T}}, \tag{55}$$

where $\mathbf{r}^{\mathrm{T}}$ is the transpose of $\mathbf{r}$. We can also use the Bellman equation directly to calculate a state-value-function iteratively. We can start with a guess $\mathbf{V}$ for the value of each state, then calculate from this a better estimate:

$$\mathbf{V} \leftarrow \mathbf{r} + \gamma\, \mathbf{T}\mathbf{V} \tag{56}$$

and so on, until this process converges. Either way, we get a value function for a specific policy. To find the best policy, the one that maximizes the expected payoff, we have to loop through different policies and find the maximal value function. This can be done in different ways, most commonly by using the *policy iteration*, which starts with a guess policy, iterates a few times the value function for this policy, and then chooses a new policy that maximizes this approximate value function. This process is repeated until convergence.

Table 4 provides an example program for a simple 1D state space consisting of a chain of 10 states, as shown on the left of Fig. 18. The 10th state is rewarded with $r = 1$, while the first state receives a large negative reward, $r = -1$. The intermediate

[4] This formulation of the Bellman equation for an MDP [36–38] is slightly different from the formulation of Sutton and Barto in [39], as these authors define the value function to be the cumulative reward starting from the next state, not the current state. In their case, the Bellman equation reads $V^\pi(s) = \sum_{s'} T(s'|s,a)(r(s') + \gamma\, V^\pi(s'))$. This is only a matter of convention about when we consider the prediction: just before getting the current reward of after taking the next step.

Table 4 Program for the chain example using the policy iteration process

```
%chain example: policy iteration

%parameters
clear; N = 10; P = 0.8; gamma = 0.9;

%reward function
r = zeros(1, N) - 0.1; r(1) = -1; r(N) = 1;

%initiality random start policy and value function
policy = ceil(2*rand(1, N)); policy(1) = 2; policy(N) = 1;
Vpi = rand(1, N); Vpi(1) = r(1); Vpi(N) = r(N);

for iter = 1:3
  %estimate V for this policy
  for i = 1:10
    for s = 2:N-1
      snext = s-1 + 2*(policy(s)-1);
      sother = s + 1-2*(policy(s)-1);
      Vpi(s) = r(s) + gamma*(P*Vpi(snext) + (1-P)*Vpi(sother));
    end
  end
  %updating policy
  for s = 2:N-1
    [tmp, policy(s)] = max([Vpi(s-1), Vpi(s + 1)]);
  end
end
plot(Vpi);
```

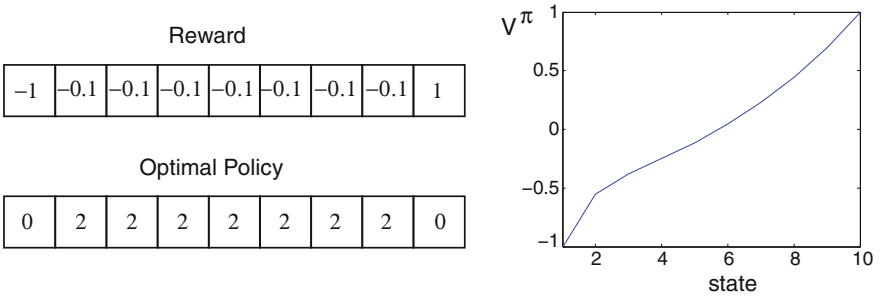

Fig. 18 Example of using policy iteration on a chain of rewarded states. *Left* reward values and optimal policy for each state, where a policy value 1 means "go left" (not present) and a value 2 means "go right". No further action is taken in the end states

states receive a small negative reward to account for movement costs. After three iterations, the policy reaches the optimal one (bottom left of figure). Actually, the optimal policy is often found within just one or two iterations, so the extra iteration was added to ensure that the value function was properly calculated for this policy.

It is also possible to derive a version of the Bellman equation *for the optimal value function* itself:

$$\textbf{Bellman-det-*} \qquad V^*(s) = r(s) + \max_a \gamma \sum_{s'} T(s'|s,a)V^*(s'). \tag{57}$$

The max function is a little more difficult to implement in the analytic solution, but we can again easily use an iterative method to solve for this optimal value function. This algorithm is called *value iteration*. The *optimal policy* can always be calculated from the optimal value function with

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s'|s,a)V^*(s'). \tag{58}$$

A policy tells an agents what action should be chosen, hence the optimal policy is related to optimal control as long as the reward reflects the desired performance.

The previously discussed policy iteration has some advantages over value iteration. In value iteration we have to try out all possible actions when evaluating the value function, which can be time consuming when there are many possible actions. In policy iteration, we choose only one specific policy, although we then have to iterate over consecutive policies. In practice, it turns out that policy iteration often converges fairly rapidly.

4.2 Temporal Difference Learning

In dynamic programming, we iterate repeatedly over every possible state of the system. This only works if we have complete knowledge of the system. In that scenario, the agent does not even have to 'perform' the actions physically, which would be very time consuming. Instead, the agent can just 'sit' and calculate the solution during a "planning phase". However, in many cases we do not know the rewards given in different states, and we usually have to estimate transition probabilities, too, etc. One approach would be to estimate these quantities by interacting with the environment before using dynamic programming. In contrast, the following methods are more direct estimations of the state value function that determines the optimal actions. These online methods assume that we still know exactly in which state the agent is, and they can be generalized to partially observable situations by considering probability maps over the state space.

A general strategy for estimating the value of states is to act in the environment and thereby sample reward. This sampling should be done with some degree of stochasticity to ensure sufficient exploration of the states. These methods are generally called *Monte Carlo* methods. Monte Carlo methods can be combined with the bootstrapping ideas of dynamic programming, and the resulting algorithms are called *temporal difference* (TD) learning, since they rely on the difference between expected reward and actual reward.

We start again by estimating the value function for a specific policy before moving to schemes for estimating the optimal policy. The Bellman equations require the estimation of future reward:

$$\sum_{s'} T(s'|s,a)V^{\pi}(s') \;\approx\; V^{\pi}(s'). \tag{59}$$

In this equation we introduced an approximation of the sum by the value of the state that is reached in one Monte Carlo step. In other words, we replace the total sum that we could build knowing the environment with a single sampling step. While this approach is only an estimation, the idea is that it will still result in an improvement of the estimation of the value function, and that other trials have the possibility to evaluate other states that have not been reached in this trial. The value function should then be updated carefully, by considering the new estimate only incrementally:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha[r(s) + \gamma\, V^{\pi}(s') - V^{\pi}(s)]. \tag{60}$$

This is called *temporal difference* or *TD learning*. The constant α is a learning rate and should be fairly small. This policy evaluation can then be combined with policy iteration as already discussed in the section on dynamic programming.

We should now think a little more about what policy to follow. An obvious choice is to take the action that leads to the largest expected payoff, also called *greedy policy*. Applying this policy should be optimal when the value function is exact. However, one problem with purely sticking to this strategy is that we might not be sufficiently "exploring" the state space—as opposed to "exploiting" known returns. We address this *exploration-exploitation dilemma* here by opting for stochastic policies. Thus we need to go back to the notation of the state-action value function (although we will drop the '*' superscript for the optimal value function for convenience). To include randomness in the policy we can, for example, follow the greedy policy most of the time, and only choose another possible action with a small probability denoted by ε. This probabilistic policy is called the *ε-greedy policy* and can be formulated as

$$\pi(a = \arg\max_{a} Q(s,a)) = 1 - \varepsilon. \tag{61}$$

A more graded approach employs the *softmax policy*, which chooses each action proportionally to a Boltzmann distribution:

$$\pi(a|s) = \frac{e^{\frac{1}{T}Q(s,a)}}{\sum_{a'} e^{\frac{1}{T}Q(s,a')}}. \tag{62}$$

This policy chooses most often the action with the highest expected reward, followed by the second highest, etc., where the temperature parameter T sets the relative probability of these choices.

We can now use these policies to explore the state space and estimate the optimal value function with temporal difference learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r(s) + \gamma\, Q(s',a') - Q(s,a)], \tag{63}$$

where the actions a' is the action chosen according to the policy. This *on-policy TD algorithm* is called *Sarsa* for state-action-reward-state-action [39]. A variant of this approach uses the stochastic action above only when choosing the next state, but estimates the value function by considering the other possible actions, too:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r(s) + \max_{a'} \gamma\, Q(s',a') - Q(s,a)]. \tag{64}$$

This is is called an *off-policy TD algorithm*, or Q-leaning [40]. These algorithms have been instrumental in the success of reinforcement learning in many engineering applications.

4.3 Function Approximation and TD(λ)

The large number of states in real-world applications makes these algorithms unpractical. This was already noted by Richard Bellman himself, who coined the phrase *"curse of dimensionality"*. We have only considered discrete state spaces, while many applications involve a continuous state space. While discretizing a continuous state space is a common approach, increasing the resolution of the discretization has the consequence of increasing the number of states exponentially. Another major problem in practice is that the environment is not fully, or reliably, observable. Thus we might not even know exactly in which state the agent finds itself when considering the value update. A common approach to a "partially observable Markov decision process" (POMDP) is the introduction of a *probability map*. In the update of the Bellman equation, we need then to consider all possible states that can be reached from the current state, something which will typically increase the number of calculations even further. We will not follow this approach here but rather consider the use of *function approximators* to overcome these problems. A more general discussion of reinforcement learning in continuous state and action spaces is given in [41].

The idea behind the following method is to make a hypothesis of the relation between sensor data and expected values in the form of a parameterized function as in supervised learning[5]:

$$V_t = V(\mathbf{x}_t) \approx V(\mathbf{x}_t; \boldsymbol{\theta}), \tag{65}$$

[5] The same function name is used on both sides of this equation, but these are distinguished by the inclusion of parameters. The value functions all refer to the parametric model, which should be clear from the context.

and to estimate the parameters by maximum likelihood as before. We use here a time index to distinguish state sequences. In principle, one could build very specific temporal Bayesian models for specific problems as discussed above, but in this circumstance I will outline the use of general learning machines. In particular, let us adjust the weights of a neural network using gradient-descent methods on a mean square error (MSE) function:

$$\Delta\theta_j = \alpha \sum_{t=1}^{m} (r - V_t) \frac{\partial V_t}{\partial \theta_j}. \tag{66}$$

We consider here the total change of the weights for a whole episode of m time steps by summing the errors for each time step. One specific difference between this situation and the supervised learning examples before is that the reward is only received after several time steps in the future, at the end of an episode. One possible approach to manage this situation is to keep a history of our predictions and make the changes for the whole episode only after the reward is received at the end. This is what we have done in Eq. (66) by providing the reward r as supervision signal in each timestep. Another approach is to make incremental (online) updates by following the TD learning philosophy, and replacing the supervision signal for a particular time step by the prediction of the value of the next time step. Specifically, we can write the difference between the received reward $V_{m+1} = r$ at the end of the sequence and the prediction V_t at time t as

$$r - V_t = \sum_{u=t}^{m} (V_{u+1} - V_u) \tag{67}$$

since the intermediate terms chancel out. Replacing this in Eq. (66) yields

$$\Delta\theta_j = \alpha \sum_{t=1}^{m} \sum_{u=t}^{m} (V_{u+1} - V_u) \frac{\partial V_t}{\partial \theta_j} \tag{68}$$

$$= \alpha \sum_{t=1}^{m} (V_{t+1} - V_t) \sum_{u=1}^{t} \frac{\partial V_u}{\partial \theta_j}, \tag{69}$$

which can be verified by developing the sums and reordering the terms. Of course, this is only rewriting the original equation, Eq. (66). We still have to keep a memory of all the gradients from the previous time steps, or at least a running sum of these gradients.

While the rules portrayed in Eqs. (66) and (69) are equivalent, Richard Sutton [42] suggested a modified version that multiplied recent gradients by stronger weights than gradients in the more remote past. For this, he introduced a decay factor $0 \leq \lambda \leq 1$. The rule above corresponds to $\lambda = 1$ and is called the *TD(1) rule*, while the more general *TD(λ) rule* is given by

$$\Delta_t \theta_j = \alpha(V_{t+1} - V_t) \sum_{u=1}^{t} \lambda^{t-u} \frac{\partial V_u}{\partial \theta_j}. \tag{70}$$

It is also interesting to look at the other extreme, when $\lambda = 0$. The *TD(0) rule* is given by

$$\Delta_t \theta_j = \alpha(V_{t+1} - V_t) \frac{\partial V_t}{\partial \theta_j}. \tag{71}$$

While this last rule gives in principle different results fom the original supervised learning problem described by TD(1), it has the advantage that it is local in time, does not require any memory, and often still works very well. The TD(λ) algorithm can be implemented in a multilayer perceptron where the error term is back-propagated to hidden layers. A generalization to stochastic networks has also been made within the framework of free-energy formalism [43].

5 Some Biological Analogies

The brain seems to be a very successful learning machine, and it is therefore not surprising that human capabilities have motivated much research in artificial intelligence. Conversely, insights from learning theory are important, too, for our understanding of brain processes. In this last section, I want to mention some interesting relations that neuroscience has with learning theory. I already remarked on the close links between unsupervised learning and receptive fields in the early sensory areas of the cortex, which I believe is a wonderful example of underlying mechanisms behind physiological findings. In the following, I would like to add comments on two other subjects related to supervised learning and reinforcement learning. The first is about *synaptic plasticity*, which appears to be an important mechanism for the physical implementation of learning rules. The second is about the close relation of reinforcement learning with classical conditioning and the *basal ganglia*. Classical conditioning has been a major area in animal learning, and recent recordings in the basal ganglia have helped relating these areas on a behavioural, physiological and learning-theoretical level.

5.1 Synaptic Plasticity

As speculated by the Canadian scientist Donald Hebb [44], the leading theory of the physical implementation of learning is that of synaptic changes, whereby the synaptic efficacy varies in response to causally related pre- and postsynaptic firings. Such correlation rules have first been made concrete by Eduardo Caianiello [45], and have recently been refined in terms of "spike timing-dependent plasticity" (STDP; see for example [46]). The main idea is that when a driving neuron participates in

firing a subsequent neuron, then the connection strength between these neurons will increase—whereas it will decrease in the absence of correlated firing. Many of the learning rules of neural networks have followed this main association rule through increment terms that are proportional to pre- and postsynaptic activity, such as

$$\Delta w_{ij} \propto x_i x_j . \tag{72}$$

Synaptic plasticity is not only a fascinating area in neuroscience but also constitutes an important medical issue, since neurodegenerative disorders, such as Alzheimer's disease and dementia, have synaptic effects and a great number of psychiatric medications exert their action on the synaptic receptors.

There are many mysteries left that need to be understood if we want to make progress in helping with neurological conditions and maybe even make progress in machine learning. One basic fact that seems puzzling is that synapses are not long-lasting compared to the time scale of human memories.[6] Synapses consist of proteins that have to be actively maintained by protein synthesis. Thus, one may wonder how this maintenance can survive for years and support long-term memory, such as returning to our place of birth after many years of absence, or meeting friends whom we had not seen in ages. These are fundamental questions that, to my knowledge, have not been sufficiently addressed.

While the Hebbian perspective on synaptic plasticity and learning is well established, I would like to outline an aspect of synaptic plasticity that might be less well-known. In particular, I would like to point out the findings of my friend Alan Fine and his colleagues [47], which fit nicely with the probabilistic theme that I have emphasized in this chapter. Fine and colleagues have performed classical plasticity experiments that use high- or low-frequency stimulations of hippocampal slices of rodents to induce measurable changes in synapses. Some of their results are summarized in Fig. 19. To test the strength of the synapses, they stimulated them with two pulses, as paired pulses facilitate synaptic responses (the second pulse makes it easier to elicit a postsynaptic spike). The slices are then activated with high-frequency stimulations inbetween these tests. As shown in Fig. 19a, the electric response of the postsynaptic neuron as measured by the excitatory post-synaptic potential (EPSP) is higher after the high-frequency stimulation. This corresponds to the classical findings by Bliss and Lømo [48] and is called long-term potentiation (LTP), since this enhanced response to a presynaptic stimulus lasts relatively long compared to the usual scale of neuronal dynamics. Of course, the EPSP is a measure that can depend on multiple synapses. But Fine and colleagues also imaged the calcium-related optical luminance signal from individual synapses. This is shown in Fig. 19b. Surprisingly, they observed that this luminance did not change despite the fact the calcium-dependent mechanisms are generally associated with synaptic activity and plasticity. Instead, they found that the probability of eliciting a postsynaptic spike varied nicely. Specifically, the probability of transmitter release increases with high-frequency simulations that are usually associated with LTP. They could also

[6] Julian Miller made this point nicely at the aforementioned workshop.

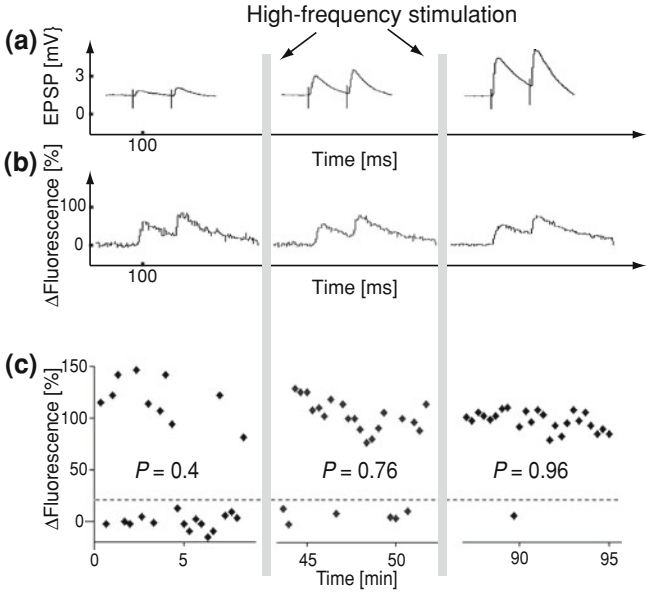

Fig. 19 Plasticity experiment in hippocampal slices in which not only EPSPs were measured, but additionally postsynaptic calcium-dependent fluorescence signals at single synapses were imaged (data courtesy of Alan Fine and Ryosuke Enoki, after [47])

lower the probability of transmitter release with low-frequency stimulus that usually elicits a decrease in EPSPs, called long-term depression (LTD; not shown in the figure).

A manipulation of the probability of transmitter release could explain the increased EPSP in such experiments. If there is a population of synapses that drive that neuron, than a population of synapses with higher likelihood of transmitter release would result in a larger EPSP than a population with smaller likelihood of transmitter release. In this sense, the findings are still consistent with some of the consequences of synaptic plasticity. But these findings also point to additional possibilities also consistent with the view that brain processing might be based on probabilistic computation rather than dealing with point estimates. Thus, the common view of a noisy nervous system with noisy synapses might be misleading. If this is noise in the sense of the "limitations" of a biological implementation, then why could the probability of synaptic responses be modulated reliably?

From a theoretical perspective it is rather difficult for noise to survive thresholding processes. For example, consider a biased random walk to a threshold as shown on the left-hand side in Fig. 20. In this example we add 1 plus a Gaussian noise (mean μ, standard deviation σ) to the signal at each time step, then the signal is reset when crossing the threshold. The noise in the process leads to different times of threshold crossings, and the variation of these times is related to the variations in the signal as shown on the right-hand side of Fig. 20 where the coefficient of variation $C_v = \sigma/\mu$

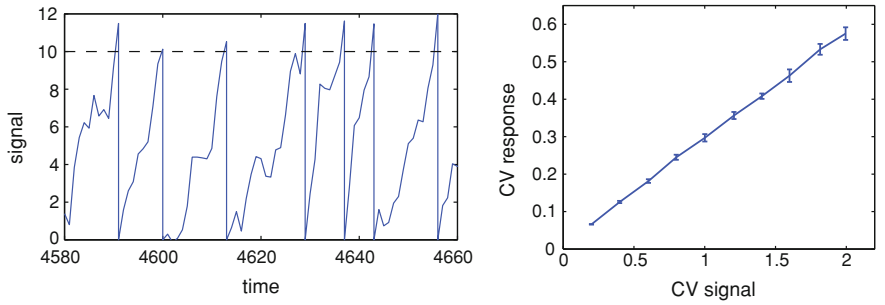

Fig. 20 Demonstration of the relation between variability in signal versus variability in spike timing response. The graph on the left side shows a noisy accumulation toward a threshold. The graph on the right shows how the coefficient of variation (C_v) varies with noise

is plotted. While there is a positive slope between them (higher noise leads to higher variations in firing times), the proportionality factor is only around $1/\sqrt{4\pi}$. Hence, if noise is an issue, then one could use thresholding mechanisms to reduce it and through repeated stages, as in the brain, the noise should become smaller. Or, in other words, if noise is the problem then one should filter it out early in the process and higher processes should be less noisy. In sum, it could be that signal variations in the brain are not all undesirable noise but could play an important information processing role such as representing the likelihood of sensory signals or the confidence in possible actions. This conjecture is consistent with the probabilistic approaches to machine learning.

5.2 Classical Conditioning and the Basal Ganglia

One of the important roles of computational neuroscience is to bridge the gap between behavioural and physiological findings [49]. The following discussion is a good example. Classical conditioning has been intensively studied in the psychological discipline of animal learning at least since the studies by Pavlov. One of the most basic findings of Pavlov is that it is possible to learn the fact that a stimulus is predicting a reward, and that this prediction elicits the same behaviour as the primary reward signal, such as salivation following a tone when the tone predicts food reward. Many similar predictions have been summarized very successfully by the Rescorla-Wagner theory [50]. In terms of the learning paradigms discussed above, this theory relates the change in the value of a state ΔV_i to the reward prediction error $\lambda - V_i$ by the formula

$$\Delta V_i = \alpha_i \beta (\lambda - V_i), \tag{73}$$

where factors α_i and β describe the saliencies of the conditioned and unconditioned stimulus, respectively, and λ represents the reward. This model is equivalent to

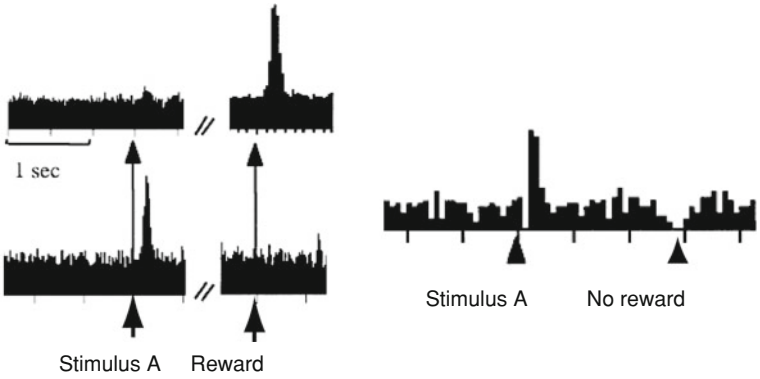

Fig. 21 Recordings by Schultz et al. [51] in a classical conditioning experiment, where a stimulus was presented followed by a reward. Early in the trials the SN neurons responded after the animal received a reward (*top left*), while the neurons responded to the predictor of the reward in later trials (*bottom left*). The neurons even seem to indicate the absence of an expected reward after learning (*right*)

temporal difference learning in a one-step prediction task where the reward follows immediately the stimulus.

The Rescola-Wagner theory with its essential reliance on the reward prediction error is very successful in explaining behaviour, and it was very exciting when Wolfram Schultz [51] and colleagues discovered neural signatures of reward prediction errors. Schultz found these signals in the *substantia nigra*, which is part of a complex of different nuclei in the midbrain called the basal ganglia. Its name means "black substance", and the dark aspect of this area is apparently due to a chemical compound related to dopamine, which these neurons transmit to the input area of the basal ganglia and to the cortex, and has been implicated in modulating learning. Some examples of the response of these neurons are shown in Fig. 21.

We can integrate the Rescorla-Wagner theory with these physiological findings in a neural network model, as shown in Fig. 22. The reward prediction error $\hat{r}$ is conveyed by the nigra neurons to the striatum, an input area of the basal ganglia, in order to mediate the plasticity of cortical-striatal synapses. The synapses are thereby assumed to contain an eligibility trace, since the learning rule requires the association with the previous state. Many psychological experiments can be modeled by a one-step prediction task where the actual reward follows a specific condition. The learning rule can then be simplified to a temporal learning rule in which the term in γ can be neglected, corresponding to the model in Fig. 22a. The implementation of the full TD rule would require a fast side-loop as shown in Fig. 22b, which has been speculated to be associated with the subthalamus [52].

Of course, the anatomy of the basal ganglia is more elaborate than this. My student Patrick Connor and I have suggested a model with lateral interactions in the striatum [53] that has some physiological grounding [54] and can explain a variety of behavioral findings not covered by the Rescorla Wagner model [55]. Moreover, there

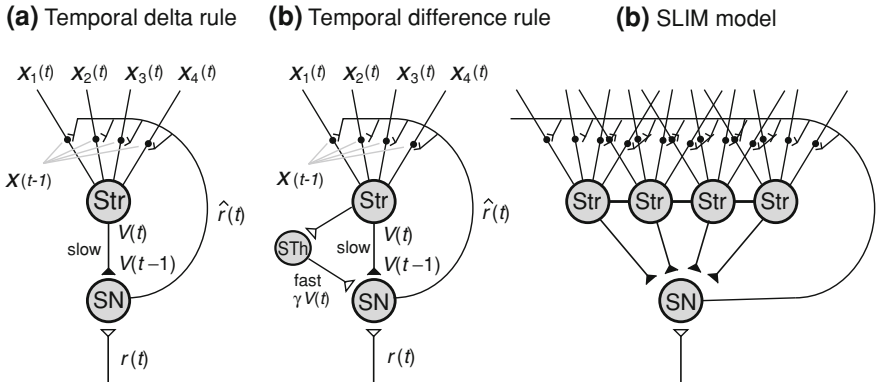

Fig. 22 Implementation of reinforcement learning models through analogies with the basal ganglia. **a** Single state of one-step reinforcement learning model (temporal delta rule) with cortical input, a striatal neuron (Str), and a neuron in the substantia nigra (SN) that conveys the reward prediction error to striatal spines. **b** Implementation of the temporal difference (TD) learning with a fast subthalamic side-loop. **c** Basic version of the striatal-with-lateral-inhibition (SLIM) model

are two main pathways through the basal ganglia, a direct pathway and an indirect one, with intermediate stages in distinct subregions of the basal ganglia (not shown in Fig. 22). The direct pathway has a inhibitory effect on the output neurons of the basal ganglia, while the indirect one has a facilitatory effect. Since the effect of the output of the basal ganglia is itself to inhibit motor areas, it has been speculated that the direct pathway could learn to inhibit non-rewarding actions, whereas the indirect pathway could learn to facilite rewarding actions. Different alterations of specific pathways have been suggested to relate to different neurological conditions that are known to involve the basal ganglia, such as Parkinson decease, Tourette syndrome, ADHD, schizophrenia and others [56]. Thus, modeling and understanding this learning system has the potential to guide refined intervention strategies.

6 Outlook

Learning is an exciting field that has made considerable progress in the last few years, specifically through statistical learning theory and its probabilistic embedding. These theories have at least clarified what could be expected from ideal learning systems, such as their ability to generalize. Much progress has also been made in unsupervised learning and starting to tackle temporal learning problems. Most excitingly, the advances in this area have enabled machine learning to find its way out of the research labs and into commercial products that have recently revolutionized technologies, such as advanced gaming platforms and smarter recommendation systems. Statistical learning theory has clarified general learning principles, such as optimal generalizability and optimal (Bayesian) decision making in the face of uncertainties.

What are the outstanding questions, then? While machine learning has enabled interesting applications, many of these applications are very focused in scope. The complexity of the environments that humans face still appears far beyond the reach of our models. Scaling up methods even farther is important to enable more applications. Many believe that, to this goal, we require truly hierarchical systems [57], and more specifically systems that process temporal data [58]. While there is exciting progress in this field, learning to map simple features, such as pixels from an image, to high-level abstract concepts, such as objects in a scene, is still challenging.

While Bayesian inference has been instrumental in the maturation of machine learning, there are also severe limitations to such methods. Specifically, truly Bayesian methods have an unbounded requirement for knowledge as we typically have to sum over all possible outcomes with their likelihood of each event in order to faithfully calculate posteriors. This seems not only excessive in its required knowledge and processing demands, but also faces practical limitations in many applications. An alternative approach is *bounded rationality*, which could be underlying a lot of human decision making [59]. Critical for the success of such methods are fast and frugal heuristics that depend on the environment. Thus there is a major role for learning in this domain on many different scales, including developmental and genetic domains. Understanding learning and development is therefore crucial for scientific reasons as well as technological advancements.

In this chapter, I tried to summarize and relate learning systems that sometimes seem to form different camps. While the application of probability theory made a strong impact on our understanding of learning systems in all camps, there has been some divide between Bayesian modelers, on the one hand, and people in "general" learning machine, on the other hand. The first point out that there is no such thing as a "free lunch", i.e. general learning machines can never become really good compared to specific models for a particular problem. Yet, finding these specific models can also be a major challenge that must be solved by domain experts. What kind of learner does the brain represent? Many aspects of the brain seem to resemble general learning machines such as the astonishing universality of neocortical architecture. On the other hand, the ability of high-level inference seems at this point out of the reach of such learning machines.

I believe that the brain might be somewhat inbetween, as it represents a *biased learning machine* that already encapsulates specific strategies (learned through evolution and development) in the specific environments typically encountered by the organisms. Such restricted learning machines should be able to support the emergence of Bayesian causal models that could be used by humans to argue about the world. Such models would not only enable smarter applications but would also help us in understanding more deeply the nature of cognition and the mind.

Acknowledgments I would like to express my thanks to René Doursat for careful edits, Christian Albers, Igor Farkas, and Stephen Grossberg for useful comments of an earlier draft circulation, and all the colleagues that have provided me with encouraging comments.

References

1. S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992)
2. P. Smolensky, Information Processing in Dynamical Systems: Foundations of Harmony Theory, in *Parallel Distributed Processing: Volume 1: Foundations*, ed. by D.E. Rumelhart, J.L. McClelland (MIT Press, Cambridge, MA, 1986), pp. 194–281
3. G. Hinton, Training products of experts by minimizing contrastive divergence. Neural Comput. **14**, 1711–1800 (2002)
4. G. Hinton, A Practical Guide to Training Restricted Boltzmann Machines. University of Toronto Technical Report UTML TR 2010–003, 2010
5. A. Graps, *An Introduction to Wavelets*. http://www.amara.com/IEEEwave/IEEEwavelet.html
6. N. Huang et al., The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. Proc. R. Soc. Lond. A **454**, 903–995 (1998)
7. H. Barlow (1961) Possible principles underlying the transformation of sensory messages. Sens. Commun. 217–234, (1961)
8. P. Földiák, Forming sparse representations by local anti-Hebbian learning. Biol. Cybern. **64**, 165–170 (1990)
9. P. Földiák, D. Endres, Sparse coding. Scholarpedia **3**, 2984 (2008)
10. B. Olshausen, D. Field, Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature **381**, 607–609 (1996)
11. H. Lee, E. Chaitanya and A. Ng, Sparse deep belief net model for visual area V2, NIPS*2007
12. C. von der Malsburg, Self-organization of orientation sensitive cells in the striate cortex. Kybernetik **14**, 85–100 (1973)
13. S. Grossberg, Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. Biol. Cybern. **23**, 121–134 (1976)
14. T. Kohonen, *Self-Organizing Maps* (Springer, Berlin, 1994)
15. P. Hollensen, P. Hartono, T. Trappenberg (2011) Topographic RBM as Robot Controller, JNNS 2011
16. S. Grossberg, Adaptive resonance theory: how a brain learns to consciously attend, learn, and recognize a changing world. Neural Netw. **37**, 1–47 (2012)
17. T. Trappenberg, P. Hartono, D. Rasmusson, in Top-Down Control of Learning in Biological Self-Organizing Maps, ed. by J. Principe, R. Miikkulainen. Lecture Notes in Computer Science 5629, WSOM 2009 (Springer, 2009), pp. 316–324
18. K. Tanaka, H. Saito, Y, Fukada, M. Moriya, Coding visual images of objects in the inferotemporal cortex of the macaque monkey. J. Neurophysiol. **66**, 170–189 (1991)
19. S. Chatterjee, A. Hadi, *Sensitivity Analysis in Linear Regression* (John Wiley & Sons, New York, 1988)
20. Judea Pearl, *Causality: Models, Reasoning and Inference* (Cambridge University Press, Cambridge, 2009)
21. D. Cireşan, U. Meier, J. Masci, J. Schmidhuber, Multi-column deep neural network for traffic sign classification. Neural Netw. **32**, 333–338 (2012)
22. D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
23. K. Hornik, Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
24. A. Weigend, D. Rumelhart (1991) Generalization through minimal networks with application to forecasting, ed. by E.M. Keramidas. in *Computing Science and Statistics (23rd Symposium INTERFACE'91, Seattle, WA)*, pp. 362–370
25. R. Caruana, S. Lawrence, C.L. Giles, Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping, in *Proceedings of Neural Information Processing Systems Conference*, 2000. pp. 402–408
26. D.J.C. MacKay, A practical Bayesian framework for backpropagation networks. Neural Comput. **4**(3), 448–472 (1992)

27. D. Silver, K. Bennett, Guest editor's introduction: special issue on inductive transfer learning. Mach. Learn. **73**(3), 215–220 (2008)
28. S. Pan, Q. Yang, A survey on transfer learning. IEEE Trans. Knowl. Data Eng. (IEEE TKDE) **22**(10), 1345–1359 (2010)
29. B.E. Boser, I.M. Guyon, V. Vapnik, A training algorithm for optimal margin classifiers, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, (ACM, 1992), pp. 144–152
30. V. Vapnik, *The Nature of Statistical Learning Theory* (Springer, Berlin, 1995)
31. C. Cortes, V. Vapnik, Support-vector networks. Mach. Learn. **20**, 273–297 (1995)
32. C. Burges, A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Disc. **2**(2), 121–167 (1998)
33. A. Smola, B. Schölkopf, A tutorial on support vector regression. Stat. Comput. **14**(3) (2004)
34. C.-C. Chang, C.-J. Lin, LibSVM: a library for support vector machines (2001), http://www.csie.ntu.edu.tw/cjlin/libsvm
35. M. Boardman, T. Trappenberg, A heuristic for free parameter optimization with support vector machines, WCCI 2006, pp. 1337–1344, (2006). http://www.cs.dal.ca/boardman/wcci
36. E. Alpaydim, *Introduction to Machine Learning, 2e* (MIT Press, Cambridge, 2010)
37. S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics* (MIT Press, Cambridge, 2005)
38. S. Russel, P. Norvigm, *Artificial Intelligence: A Modern Approach*, 3rd edn. (Prentice Hall, New York, 2010)
39. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, 1998)
40. C.J.C.H. Watkins, Learning from Delayed Rewards. Ph.D. thesis, Cambridge University, Cambridge, England, 1989
41. H. van Hasselt, Reinforcement learning in continuous state and action spaces. Reinforcement Learn.: Adapt. Learn. Optim. **12**, 207–251 (2012).
42. R. Sutton, Learning to predict by the methods of temporal differences. Mach. Learn. **3**, 9–44 (erratum p. 377) (1988)
43. B. Sallans, G. Hinton, Reinforcement learning with factored states and actions. J. Mach. Learn. Res. **5**, 1063–1088 (2004)
44. D.O. Hebb, *The Organization of Behaviour* (John Wiley & Sons, New York, 1949)
45. E.R. Caianiello, Outline of a theory of thought-processes and thinking machines. J. Theor. Biol. **1**, 204–235 (1961)
46. T. Trappenberg, *Fundamentals of Computational Neuroscience*, 2nd edn. (Oxford University Press, Oxford, 2010)
47. R. Enoki, Y.L. Hu, D. Hamilton, A. Fine, Expression of long-term plasticity at individual synapses in hippocampus is graded, bidirectional, and mainly presynaptic: optical quantal analysis. Neuron **62**(2), 242–253 (2009)
48. T. Bliss, T. Lømo, Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. J. Physiol. **232**(2), 331–56 (1973)
49. D. Heinke, E. Mavritsaki (eds.), *Computational Modelling in Behavioural Neuroscience: Closing the gap between neurophysiology and behaviour* (Psychology Press, London, 2008)
50. R. Rescorla, A. Wagner, in *A Theory of Pavlovian Conditioning: Variations, in the Effectiveness of Reinforcement and Nonreinforcement*, ed. by W.F. Prokasy, A.H. Black, Classical Conditioning, II: Current Research and Theory, (Appleton Century Crofts, New York, 1972), pp. 64–99
51. W. Schultz, Predictive reward signal of dopamine neurons. J. Neurophysiol. **80**(1), 1–27 (1998)
52. J. Houk, J. Adams, A. Barto in *A Model of How the Basal Ganglia Generate and Use Neural Signals that Predict Reinforcement*, ed. by J.C. Hauk, J.L. Davis, D.G. Breiser. Models of Information Processing in the Basal Ganglia (MIT Press, Cambridge, 1995)
53. P. Connor, T. Trappenberg, in *Characterizing a Brain-Based Value-Function Approximator*, ed. by E. Stroulia, S. Matwin, Advances in Artificial Intelligence LNAI 2056, (Springer, Berlin, 2011), pp. 92–103

54. J. Reynolds, J. Wickens, Dopamine-dependent plasticity of corticostriatal synapses. Neural Netw. **15**(4–6), 507–521 (2002)
55. P. Connor, V. LoLordo, T. Trappenberg (2012) An elemental model of retrospective revaluation without within-compound associations. Anim. Learn. **42**(1), 22–38
56. T. Maia, M. Frank, From reinforcement learning models to psychiatric and neurological disorders. Nat. Neurosci. **14**, 154–162 (2011)
57. Y. Bengio, Learning deep architectures for AI. Found. Trends Mach. Learn. **2**, 1–127 (2009)
58. J. Hawkins, *On Intelligence* (Times Books, New York, 2004)
59. G. Gigerenzer, P. Todd and the ABC Research Group, *Simple Heuristics that Make Us Smart* (Oxford University Press, Oxford, 1999)

Chapter 3
Evolving Culture Versus Local Minima

Yoshua Bengio

Abstract We propose a theory that relates difficulty of learning in deep architectures to culture and language. It is articulated around the following hypotheses: (1) learning in an individual human brain is hampered by the presence of effective local minima; (2) this optimization difficulty is particularly important when it comes to learning higher-level abstractions, i.e., concepts that cover a vast and highly-nonlinear span of sensory configurations; (3) such high-level abstractions are best represented in brains by the composition of many levels of representation, i.e., by deep architectures; (4) a human brain can learn such high-level abstractions if guided by the signals produced by other humans, which act as hints or indirect supervision for these high-level abstractions; and (5), language and the recombination and optimization of mental concepts provide an efficient evolutionary recombination operator, and this gives rise to rapid search in the space of communicable ideas that help humans build up better high-level internal representations of their world. These hypotheses put together imply that human culture and the evolution of ideas have been crucial to counter an optimization difficulty: this optimization difficulty would otherwise make it very difficult for human brains to capture high-level knowledge of the world. The theory is grounded in experimental observations of the difficulties of training deep artificial neural networks. Plausible consequences of this theory for the efficiency of cultural evolution are sketched.

Y. Bengio (✉)
CIFAR Fellow, Department of computer science and operations research, University of Montréal, Montreal, Canada
e-mail: yoshua.bengio@umontreal.ca

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_3,

1 Introduction

Interesting connections can sometimes be made at the interface between artificial intelligence research and the sciences that aim to understand human brains, cognition, language, or society. The aim of this chapter is to propose and elaborate a theory at this interface, inspired by observations rooted in machine learning research, on so-called Deep Learning.[1] Deep Learning techniques aim at training models with many levels of representation, a hierarchy of features and concepts, such as can be implemented with artificial neural networks with many layers. A *deep architecture* typically has more than 2 or 3 trained levels of representation, and in fact a *deep learning algorithm* can *discover* the appropriate number of levels of representation based on the training data. The visual cortex is believed to have between 5 and 10 such levels. Theoretical arguments have also been made to suggest that deep architectures are necessary to efficiently represent the kind of high-level concepts required for artificial intelligence [7]. This chapter starts from experimental observations of the difficulties in training deep architectures [15], and builds a theory of the role of cultural evolution to reduce the difficulty of learning high-level abstractions. The gist of this theory is that training deep architectures such as those found in the brain is difficult because of an optimization difficulty (apparent local minima), but that the cultural evolution of ideas can serve as a way for a whole population of humans, over many generations, to efficiently discover better solutions to this optimization problem.

2 Neural Networks and Local Minima

2.1 Neural Networks

Artificial neural networks are computational architectures and learning algorithms that are inspired by the computations believed to take place in the biological neural networks of the brain [2]. The dominant and most successful approaches to training artificial neural networks are all based on the idea that *learning can proceed by gradually optimizing a criterion* [37]. A neural network typically has free parameters, such as the synaptic strengths associated with connections between neurons. Learning algorithms formalize the computational mechanism for changing these parameters so as to take into account the evidence provided by observed (training) examples. Different learning algorithms for neural networks differ in the specifics of the criterion and how they optimize it, often approximately because no analytic and exact solution is possible. On-line learning, which is most plausible for biological organisms, involves changes in the parameters either after each example has been

[1] See [3] for a review of Deep Learning research, which had a breakthrough in 2006 [6, 22, 36].

seen or after a small batch of examples has been seen (maybe corresponding to a day's worth of experience).

2.2 Training Criterion

In the case of biological organisms, one could imagine that the ultimate criterion involves the sum of expected future rewards (survival, reproduction, and other innately defined reward signals such as hunger, thirst, and the need to sleep).[2] However, intermediate criteria typically involve modeling the observations from the senses, i.e., improving the prediction that could be made of any part of the observed sensory input given any other part, and improving the prediction of future observations given the past observations. Mathematically, this can often be captured by the statistical criterion of maximizing likelihood, i.e., of maximizing the probability that the model implicitly or explicitly assigns to new observations.

2.3 Learning

Learners can exploit observations (e.g., from their sensors of the real world) in order to construct functions that capture some of the statistical relationships between the observed variables. For example, learners can build predictors of future events given past observations, or associate what is observed through different modalities and sensors. This may be used by the learner to predict any unobserved variable given the observed ones. The learning problem can be formalized as follows. Let θ be a vector of parameters that are free to change while learning (such as the synaptic strengths of neurons in the brain). Let z represent an example, i.e., a measurement of the variables in the environment which are relevant to the learning agent. The agent has seen a past history $z_1, z_2, \ldots, z_t$, which in realistic cases also depends on the actions of the agent. Let $E(\theta, z)$ be a measurement of an error or loss to be minimized, whose future expected value is the criterion to be minimized. In the simple case[3] where we ignore the effect of current actions on future rewards but only consider the value of a particular solution to the learning problem over the long term, the objective of the learner is to minimize the criterion

$$C(\theta) = \int P(z)E(\theta, z)dz = \mathbb{E}[E(\theta, Z)] \tag{1}$$

[2] Note that the rewards received by an agent depend on the tasks that it faces, which may be different depending on the biological and social niche that it occupies.

[3] Stationary i.i.d case where examples independently come from the same stationary distribution P.

which is the expected future error, with $P(z)$ the unknown probability distribution from which the world generates examples for the learner. In the more realistic setting of reinforcement learning [43], the objective of the learner is often formalized as the maximization of the expected value of the weighted sum of future rewards, with weights that decay as we go further into the future (food now is valued more than food tomorrow, in general). Note that the training criterion we define here is called *generalization error* because it is the expected error on new examples, not the error measured on past training examples (sometimes called training error). Under the stationary i.i.d. hypothesis, the expected future reward can be estimated by the ongoing online error, which is the average of rewards obtained by an agent. In any case, although the training criterion cannot be computed exactly (because $P(\cdot)$ is unknown to the learner), the criterion $C(\cdot)$ can be approximately minimized[4] by stochastic gradient descent (as well as other gradient-based optimization techniques): the learner just needs to estimate the gradient $\frac{\partial E(\theta,z)}{\partial \theta}$ of the example-wise error E with respect to the parameters, i.e., estimate the effect of a change of the parameters on the immediate error. Let g be such an estimator (e.g., if it is unbiased then $\mathbb{E}[g] = \mathbb{E}[\frac{\partial E(\theta,z)}{\partial \theta}]$). For example, g could be based on a single example or a day's worth of examples.

Stochastic gradient descent proceeds by small steps of the form

$$\theta \leftarrow \theta - \eta g \tag{2}$$

where η is a small constant called learning rate or gain. Note that if new examples z are continuously sampled from the unknown distribution $P(z)$, the instantaneous online gradient g is an unbiased estimator of the generalization error gradient (which is the integral of g over P), i.e., an online learner is directly optimizing generalization error.

Applying these ideas to the context of biological learners gives the hypothesis that follows.

2.4 What Do Brains Optimize?

Optimization Hypothesis. When the brain of a single biological agent learns, it performs an approximate optimization with respect to some endogenous objective.

Here note that we refer to a *single* learning agent because we exclude the effect of interactions between learning agents, like those that occur because of communication

[4] In many machine learning algorithms, one minimizes the training error plus a regularization penalty which prevents the learner from simply learning the training examples by heart without good generalization on new examples.

between humans in a human society. Later we will advocate that in fact when one takes into account the learning going on throughout a society, the optimization is not just a local descent but involves a global parallel search similar to that performed by evolution and sexual reproduction.

Note that the criterion we have in mind here is not specialized to a single task, as is often the case in applications of machine learning. Instead, a biological learning agent must make good predictions in all the contexts that it encounters, and especially those that are more relevant to its survival. Each type of context in which the agent must take a decision corresponds to a "task". The agent needs to "solve" many tasks, i.e. perform *multi-task learning*, *transfer learning* or *self-taught learning* [11, 35]. All the tasks faced by the learner share the same underlying "world" that surrounds the agent, and brains probably take advantage of these commonalities. This may explain how brains can sometime learn a new task from a handful or even just one example, something that seems almost impossible with standard single-task learning algorithms.

Note also that biological agents probably need to address multiple objectives together. However, in practice, since the same brain must take the decisions that can affect all of these criteria, these cannot be decoupled but they can be lumped into a single criterion with appropriate weightings (which may be innate and chosen by evolution). For example, it is very likely that biological learners must cater both to a "predictive" type of criterion (similar to the data-likelihood used in statistical models or in *unsupervised learning* algorithms) and a "reward" type of criterion (similar to the rewards used in reinforcement learning algorithms). The former explains curiosity and our ability to make sense of observations and learn from them even when we derive no immediate or foreseeable benefit or loss. The latter is clearly crucial for survival, as biological brains need to focus their modeling efforts on what matters most to survival. Unsupervised learning is a way for a learning agent to prepare itself for any possible task in the future, by extracting as much information as possible from what it observes, i.e., figuring out the unknown explanations for what it observes.

One issue with an objective defined in terms of an animal survival and reproduction (presumably the kinds of objectives that make sense in evolution) is that it is not well defined: it depends on the behaviors of other animals and the whole ecology and niche occupied by the animal of interest. As these change due to the "improvements" made by other animals or species through evolution or learning, the individual animal or species's objective of survival also changes. This feedback loop means there isn't really a static objective, but a complicated dynamical system, and the discussions regarding the complications this brings are beyond the scope of this chapter. However, it is interesting to note that there is one component of an animal's objective (and certainly of many humans' objective, especially scientists) that is much more stable: it is the "unsupervised learning" objective of "understanding how the world ticks".

2.5 Local Minima

Stochastic gradient descent is one of many optimization techniques that perform a *local descent*: starting from a particular configuration of the parameters (e.g. a configuration of the brain's synapses), one makes small gradual adjustments which on average tend to improve the expected error, our training criterion. The theory proposed here relies on the following hypothesis:

> **Local Descent Hypothesis.** When the brain of a single biological agent learns, it relies on approximate local descent in order to gradually improve itself.

The main argument in favor of this hypothesis would be based on the assumption that although our state of mind (firing pattern of neurons) changes quickly, synaptic strengths and neuronal connectivity only change gradually.

If the learning algorithm is a form of stochastic gradient descent (as Eq. 2 above), where g approximates the gradient (it may even have a bias), and if η is chosen small enough (compared to the largest second derivatives of C), then C will gradually decrease with high probability, and if η is gradually decreased at an appropriate rate (such as $1/t$), then the learner will converge towards a *local minimum* of C. The proofs are usually for the unbiased case [9], but a small bias is not necessarily very hurtful, as shown for Contrastive Divergence [10, 48], especially if the magnitude of the bias also decreases as the gradient decreases (stochastic approximation convergence theorem [48]).

Note that in this chapter we are talking about *local minima of generalization error*, i.e., with respect to expected future rewards. In machine learning, the terms "optimization" and "local minimum" are usually employed with respect to a training criterion formed by the error on training examples (training error), which are those seen in the past by the learner, and on which it could possibly *overfit* (i.e. perform apparently well even though generalization error is poor).

2.6 Effective Local Minima

As illustrated in Fig. 1, *a local minimum is a configuration of the parameters such that no small change can yield an improvement of the training criterion.* A consequence of the **Local Descent Hypothesis**, if it is true, is therefore that biological brains would be likely to stop improving after some point, after they have sufficiently approached a local minimum. In practice, if the learner relies on a stochastic gradient estimator (which is the only plausible hypothesis we can see, because no biological learner has access to the full knowledge of the world required to directly estimate C), it will continue to change due to the stochastic nature of the gradient estimator (the training signal), hovering stochastically around a minimum. It is also quite possible that biological learners do not have enough of a lifetime to get close to an actual

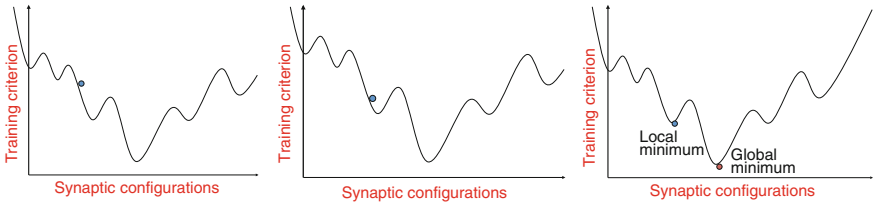

Fig. 1 Illustration of learning that proceeds by local descent, and can get stuck near a local minimum (going from *left* figure to *right* figure). The *horizontal axis* represents the space of synaptic configurations (parameters of the learner), while the *vertical axis* represents the training criterion (expected future error). The *ball* represents the learner's current state, which tends to go *downwards* (improving the expected error). Note that the space of synaptic configurations is huge (number of synaptic connections on the order of 100 trillion in humans) but represented here schematically with a single dimension, the *horizontal axis*

local minimum, but what is plausible is that they get to a point where progress is slow (so slow as to be indistinguishable from random hovering near a minimum). In practice, when one trains an artificial neural network with a learning algorithm based on stochastic gradient descent, one often observes that training *saturates*, i.e., no more observable progress is seen in spite of the additional examples being shown continuously. The learner appears stuck near a local minimum. Because it is difficult to verify that a learner is really near a local minimum, we call these *effective local minima*. We call it effective because it is due to the limitations of the optimization procedure (e.g., stochastic gradient descent) and not just to the shape of the training criterion as a function of the parameters. The learner equipped with its optimization procedure which is stuck in an effective local minimum will appear as if it is stuck in an actual local minimum (it might also be an actual local minimum). It may happen that the training criterion is a complicated function of the parameters, such that stochastic gradient descent is sometimes practically stuck in a place in which it is not possible to improve in *most directions*, but from where other more powerful descent methods could escape [32].

2.7 Inference

Many learning algorithms involve *latent variables*, which can be understood as associated with particulars factors that can contribute to "explain" each other and "explain" the current and recent observations. These latent variables are encoded in the activation of *hidden units* (neurons that are neither inputs nor outputs). One can think of a particular configuration of these latent or hidden variables as corresponding to a *state of mind*. In a Boltzmann machine [1, 18, 38], when an input is presented, many configurations of these latent variables are possible and an *inference mechanism* normally takes place in order to explore possible configurations of the latent variables which "fit well" with the observed input. This inference is often iterative,

although it can be approximated or initialized in a single bottom-up pass [40] from perception to state-of-mind. Inference can be stochastic (neurons randomly choose their state with a probability that depends on the state of the others, and such that more probable configurations of neuron activations are sampled accordingly more often) or deterministic (through an iterative process that can sometimes correspond to an optimization, gradually changing the configuration of neurons towards one that agrees more with the observed input percept). Whereas learning in brains (besides the simple memorization of facts and observed events) occurs on a scale of minutes, hours or days, inference (changes in the state of mind) occurs on the scale of a fraction of a second or few seconds. Whereas learning is probably gradual, stochastic inference can quickly jump from one thought pattern to another in less than a second. In models such as the Boltzmann machine, learning requires inference as an inner loop: patterns of latent variables (hidden units, high-level concepts) that fit well with the observed data are reinforced by the changes in synaptic weights that follow. One should not confuse local minima in synaptic weights with local minima (or the appearance of being stuck) in inference. Randomness or association with new stimuli can change the state of our inference for past inputs and give us the impression that we are not stuck anymore, that we have escaped a local minimum, but that regards the inference process, not necessarily the learning process (although it can certainly help it).

3 High-Level Abstractions and Deep Architectures

Deep architectures [3] are parametrized families of functions which can be used to model data using multiple levels of representation. In deep neural networks, each level is associated with a group of neurons (which in the brain could correspond to an area, such as areas V1, V2 or IT of the visual cortex). During sensory perception in animal brains, information travels quickly from lower (sensory) levels to higher (more abstract) levels, but there are also many feedback connections (going from higher to lower levels) as well as lateral connections (between neurons at the same level). Each neuron or group of neurons can be thought of as capturing a concept or feature or aspect, and being activated when that concept or feature or aspect is present in the sensory input, or when the model is generating an internal configuration (a "thought" or "mental image") that includes that concept or feature or aspect. Note that very few of these features actually come to our consciousness, because most of the inner workings of our brains are not directly accessible (or rarely so) to our consciousness. Note also that a particular linguistic concept may be represented by many neurons or groups of neurons, activating in a particular pattern, and over different levels (in fact so many neurons are activated that we can see whole regions being activated with brain imaging, even when a single linguistic concept is presented as stimulus). These ideas were introduced as central to connectionist approaches [19, 20, 37] to cognitive science and artificial neural networks, with the concept of *distributed representation*: what would in most symbolic systems be represented by a single "on/off" bit

(e.g., the symbol for 'table' is activated) is associated in the brain with a large number of neurons and groups of neurons being activated together in a particular pattern. In this way, concepts that are close semantically, i.e., share some attributes (e.g. represented by a group of neurons), can have an overlap in their brain representation, i.e., their corresponding patterns of activation have "on" bits in many of the same places.

3.1 Efficiency of Representation

Deeper architectures can be much more efficient in terms of representation of functions (or distributions) than shallow ones, as shown with theoretical results where for specific families of functions a too shallow architecture can require exponentially more resources than necessary [3–5, 7, 16, 17, 47]. The basic intuition why this can be true is that in a deep architecture there is *re-use* of parameters and *sharing* of sub-functions to build functions. We do not write computer programs with a single main program: instead we write many subroutines (functions) that can call other subroutines, and this nested re-use provides not only flexibility but also great expressive power. However, this greater expressive power may come at the price of making the learning task a more difficult optimization problem. Because the lower-level features can be used in many ways to define higher-level features, the interactions between parameters at all levels makes the optimization landscape much more complicated. At the other extreme, many shallow methods are associated with a convex optimization problem, i.e., with a single minimum of the training criterion.

3.2 High-Level Abstractions

We call *high-level abstraction* the kind of concept or feature that could be computed efficiently only through a deep structure in the brain (i.e., by the sequential application of several different transformations, each associated with an area of the brain or large group of neurons). An edge detector in an image seen by the eye can be computed by a single layer of neurons from raw pixels, using Gabor-like filters. This is a very low-level abstraction. Combining several such detectors to detect corners, straight line segments, curved line segments, and other very local but simple shapes can be done by one or two more layers of neurons, and these can be combined in such a way as to be locally insensitive to small changes in position or angle. Consider a hierarchy of gradually more complex features, constructing detectors for very abstract concepts which are activated whenever any stimulus within a very large set of possible input stimuli are presented. For a higher-level abstraction, this set of stimuli represents a highly-convoluted set of points, a highly curved manifold. We can picture such a manifold if we restrict ourselves to a very concrete concept, like the image of a specific object (the digit 4, as in Fig. 2) on a uniform background. The only factors that can vary here are due to object constancy; they correspond to changes in imaging geometry (location and orientation of the object with respect to the eye) and lighting,

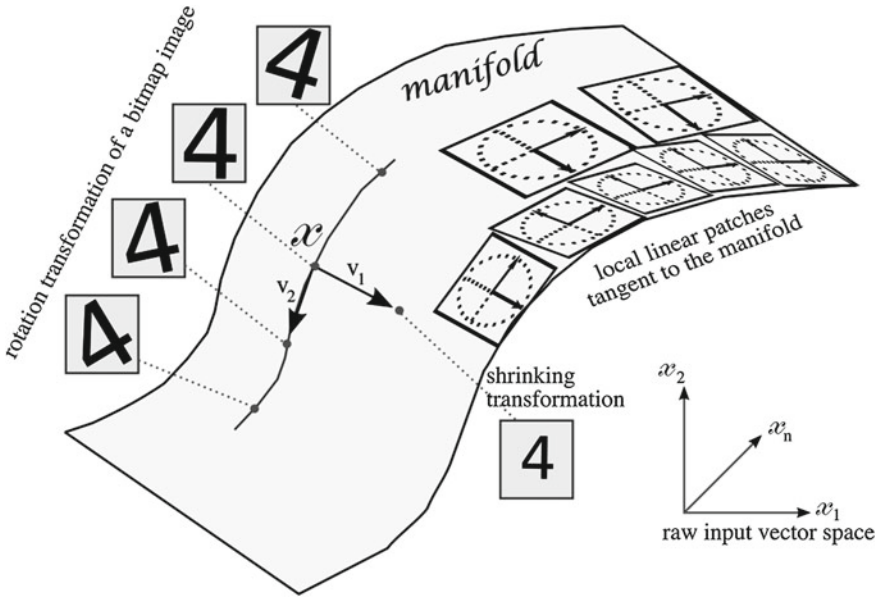

Fig. 2 Example of a simple manifold in the space of images, associated with a rather low-level concrete concept, corresponding to rotations and shrinking of a **specific** instance of the image of a drawn digit *4*. Each point on the manifold corresponds to an image which is obtained by rotating or translating or scaling another image on the manifold. The set of points in the manifold defines a concrete concept associated with the drawing of a *4* of a particular shape irrespective of its position, angle and scale. Even learning such simple manifolds is difficult, but learning the much more convoluted and higher-dimensional manifolds of more abstract concepts is much harder

and we can use mathematics to help us make sense of such manifolds. Now think about all the images which can elicit a thought of a more abstract concept, such as "human", or even more abstract, all the contexts which can elicit a thought of the concept "Riemann integral". These contexts and images associated with the same high-level concept can be very different from each other, and in many complicated ways, for which scientists do not know how to construct the associated manifolds. Some concepts are clearly higher-level than others, and often we find that higher-level concepts can be defined in terms of lower-level ones, hence forming a hierarchy which is reminiscent of the kind of hierarchy that we find current deep learning algorithms to discover [31]. This discussion brings us to the formulation of a hypothesis about high-level abstractions and their representation in brains.

Deep Abstractions Hypothesis. Higher-level abstractions in brains are represented by deeper computations (going through more areas or more computational steps in sequence over the same areas).

4 The Difficulty of Training Deep Architectures

There are a number of results in the machine learning literature that suggest that training a deeper architecture is often more difficult than training a shallow one, in the following sense. When trying to train all the layers together with respect to a joint criterion such as the likelihood of the inputs or the conditional likelihood of target classes given inputs, results can be worse than when training a shallow model, or more generally, one may suspect that current training procedures for deep networks underuse the representation potential and the parameters available, which may correspond to a form of underfitting[5] and inability at learning very high-level abstractions.

4.1 Unsupervised Layer-Wise Pre-training

The first results of that nature appear in [6, 36], where the same architecture gives very different results depending on the initialization of the network weights, either purely randomly, or based on *unsupervised layer-wise pre-training*. The idea of the layer-wise pre-training scheme [6, 22, 23, 36] is to train each layer with an unsupervised training criterion, so that it learns a new representation, taking as input the representation of the previous layer. Each layer is thus trained in sequence one after the other. Although this is probably not biologically plausible as such, what would be plausible is a mechanism for providing an unsupervised signal at each layer (group of neurons) that makes it learn to better capture the statistical dependencies in its inputs. That layer-local signal could still be combined with a global training criterion but might help to train deep networks if there is an optimization difficulty in coordinating the training of all layers simultaneously. Another indication that a layer-local signal can help to train deep networks came from the work of [46], where the unsupervised layer-local signal was combined with a supervised global signal that was propagated through the whole network. This observation of the advantage brought by layer-local signals was also made in the context of purely unsupervised learning of a deep stochastic network, the Deep Boltzmann Machine [38]. By pre-training each layer as a Restricted Boltzmann Machine (RBM)[6] before optimizing a Deep Boltzmann Machine (DBM) that comprises all the levels, the authors are able to train the DBM, whereas directly training it from random initialization was problematic. We summarize several of the above results in the deep learning literature with the following **Observation O1**: training deep architectures is easier if hints are provided about the function that intermediate levels should compute [3, 22, 38, 46]. This is connected to an even more obvious **Observation O2**, from the work on

[5] Although it is always possible to trivially overfit the top two layers of a deep network by memorizing patterns, this may still happen with very poor training of lower levels, corresponding to poor representation learning.

[6] Which ignores the interaction with the other levels, except for receiving input from the level below.

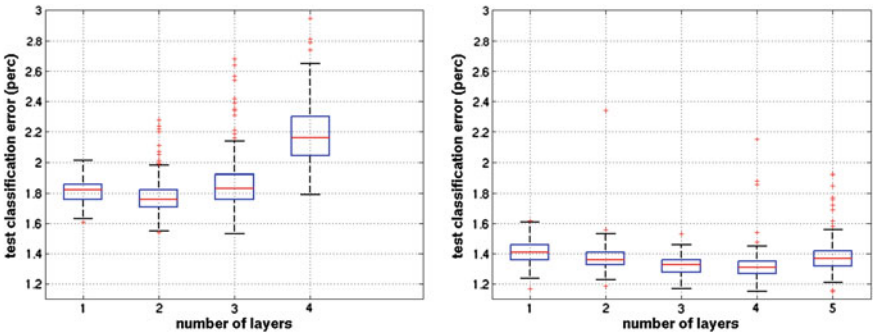

Fig. 3 Effect of depth on generalization error, **without** layer-wise unsupervised pre-training (*left*) and **with** (*right*). The training problem becomes more difficult for deeper nets, and using a layer-local cue to initialize each level helps to push the difficulty a bit farther and improve error rates

artificial neural networks: it is much easier to teach a network with supervised learning (where we provide it examples of when a concept is present and when it is not present in a variety of examples) than to expect unsupervised learning to discover the concept (which may also happen but usually leads to poorer renditions of the concept).

4.2 More Difficult for Deeper Architectures and More Abstract Concepts

Another clue to this training difficulty came in later studies [15, 30] showing that directly training all the layers together would not only make it difficult to exploit all the extra modeling power of a deeper architecture but would actually get worse results[7] *as the number of layers is increased*, as illustrated in Fig. 3. We call this **Observation O3**.

In [15] we went further in an attempt to understand this training difficulty and studied the trajectory of deep neural networks during training, in function space. Such trajectories are illustrated in Fig. 4. Each point in the trajectory corresponds to a particular neural network parameter configuration and is visualized as a two-dimensional point as follows. First, we approximate the function computed by a neural network non-parametrically, i.e., by the outputs of the function over a large test set (of 10,000 examples). We consider that two neural networks behave similarly if they provide similar answers on these test examples. We cannot directly use the network parameters to compare neural networks because the same function can be represented in many different ways (e.g., because permutations of the hidden neuron

[7] Results got worse in terms of generalization error, while training error could be small thanks to capacity in the top few layers.

indices would yield the same network function). We therefore associate each network with a very long vector containing in its elements the concatenation of the network outputs on the test examples. This vector is a point in a very high-dimensional space, and we compute these points for all the networks in the experiment. We then learn a mapping from these points to 2-dimensional approximations, so as to preserve local (and sometimes global) structure as much as possible, using non-linear dimensionality reduction methods such as t-SNE [45] or Isomap [44]. Figure 4 allows us to draw a number of interesting conclusions:

1. **Observation O4**. No two trajectories end up in the same local minimum. This suggests that the number of functional local minima (i.e. corresponding to different functions, each of which possibly corresponding to many instantiations in parameter space) must be huge.
2. **Observation O5**. A training trick (unsupervised pre-training) which changes the initial conditions of the descent procedure allows one to reach much better local minima, and these better local minima do not appear to be reachable by chance alone (note how the regions in function space associated with the two "flowers" have no overlap at all, in fact being at nearly 90° from each other in the high-dimensional function space).

Starting from the **Local Descent Hypothesis**, **Observation O4** and **Observation O5** bring us to the formulation of a new hypothesis regarding not only artificial neural networks but also humans:

> **Local Minima Hypothesis.** Learning in a single human learner is limited by effective local minima.

We again used the phrase "single human learner" because later in this chapter we will hypothesize that a collection of human learners and the associated evolution of their culture can help to get out of what would otherwise be effective local minima.

Combining the above observations with the worse results sometimes observed when training deeper architectures (**Observation O3**, discussed above), we come to the following hypothesis.

> **Deeper Harder Hypothesis.** The detrimental effect of effective local minima tends to be more pronounced when training deeper architectures (by an optimization method based on iteratively descending the training criterion).

Finally, the presumed ability of deeper architectures to represent higher-level abstractions more easily than shallow ones (see [3] and discussion in Sect. 3.1) leads us to a human analogue of the **Deeper Harder Hypothesis**, which refines the **Local Minima Hypothesis**:

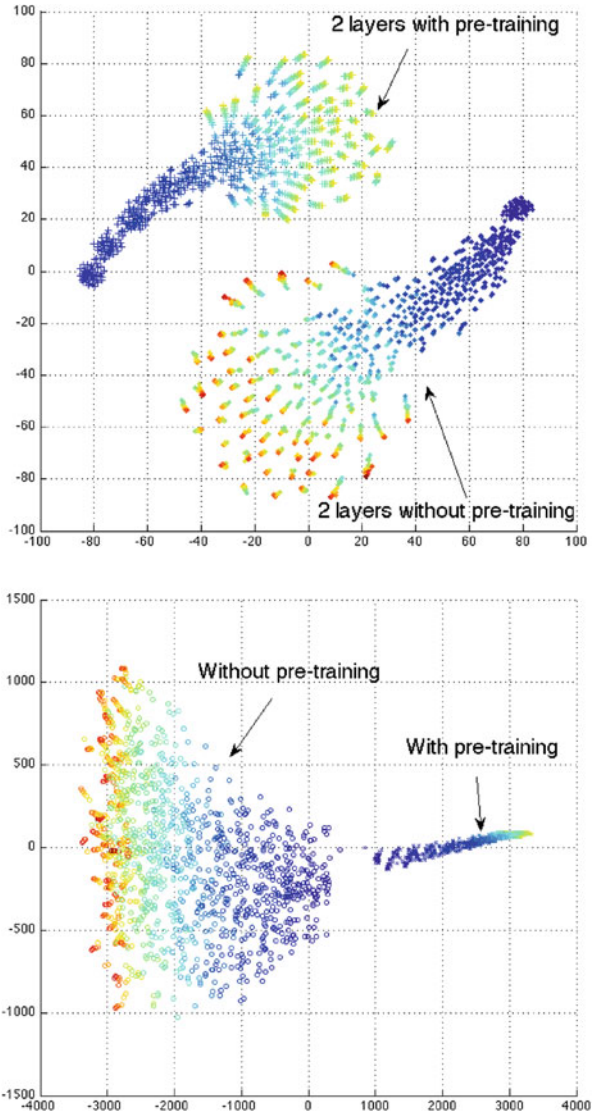

Fig. 4 Two-dimensional non-linear projection of the space of functions visited by artificial neural networks during training. Each *cross* or *diamond* or *circle* represents a neural network at some stage during its training, with *color* indicating its age (number of examples seen), starting from *blue* and moving towards *red*. Networks computing a similar function (with similar response to similar stimuli) are nearby on the graph. *Top figure* uses t-SNE for dimensionality reduction (insists on preserving local geometry) while the *bottom figure* uses Isomap (insists on preserving global geometry and volumes). The *vertical crosses* (*top figure*) and *circles* (*bottom figure*) are networks trained from random initialization, while the *diamonds* (*top figure*) and rotated *crosses* (*bottom figure*) are networks with unsupervised pre-training initialization

Abstractions Harder Hypothesis. A single human learner is unlikely to discover high-level abstractions by chance because these are represented by a deep sub-network in the brain.

Note that this does not prevent some high-level abstractions to be represented in a brain due to innate programming captured in the genes, and again the phrase *single human learner* excludes the effects due to culture and guidance from other humans, which is the subject of the next section.

5 Brain to Brain Transfer of Information to Escape Local Minima

If the above hypotheses are true, one should wonder how humans still manage to learn high-level abstractions. We have seen that much better solutions can be found by a learner if it is initialized in an area from which gradient descent leads to a good solution, and genetic material might provide enough of a good starting point and architectural constraints to help learning of some abstractions. For example, this could be a plausible explanation for some visual abstractions (including simple face detection, which newborns can do) and visual invariances, which could have had the chance to be discovered by evolution (since many of our evolutionary ancestors share a similar visual system). Recent work on learning algorithms for computer vision also suggest that architectural constraints can greatly help performance of a deep neural network [27], to the point where even random parameters in the lower layers (along with appropriate connectivity) suffice to obtain reasonably good performance on simple object recognition tasks.

5.1 Labeled Examples as Hints

However, many of the abstractions that we master today have only recently (with respect to evolutionary scales) appeared in human cultures, so they could not have been genetically evolved: each of them must have been discovered by at least one human at some point in the past and then been propagated or improved as they were passed from generation to generation. We will return later to the greater question of the evolution of ideas and abstractions in cultures, but let us first focus on the mechanics of communicating good synaptic configurations from one brain to another. Because we have a huge number of synapses and their values only make sense in the context of the values of many others, it is difficult to imagine how the recipe for defining individual abstractions could be communicated from one individual to another in a direct way (i.e. by exchanging synaptic values). Furthermore, we need to ask how

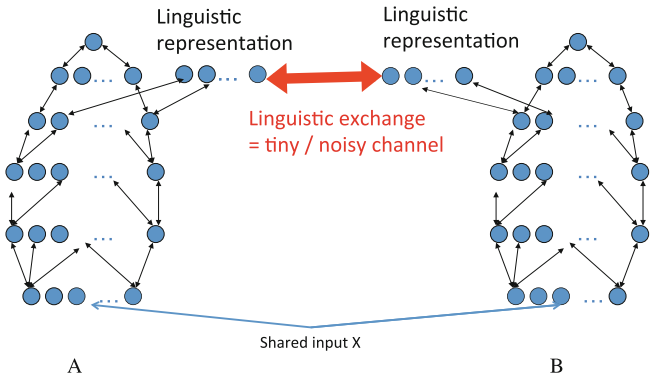

Fig. 5 Illustration of the communication between brains, typically through some language, in a way that can give hints to higher levels of one brain of how the concepts are represented in higher levels of another brain. Both learners see shared input X, and say that A produces an utterance (from its language related areas) that is strongly associated with A's high-level state of mind as a representation of X. B also sees this utterance as an input (that sets B's current linguistic representation units), that it tries to predict from its internal representations of X. Turns may change with B speaking and A listening, so that both get a sense of the explanation of X that the other is forming in its respective state of mind

the hypothesized mechanism could help to escape effective local minima faced by a single learner.

The main insight to answering this question may come from **Observation O1** and **Observation O2**. Training a single hidden layer neural network (supervised or unsupervised) is much easier than training a deeper one, so if one can provide a hint as to the function that deeper layers (corresponding to higher-level abstractions) should capture, then training would be much easier. In the extreme, specifying how particular neurons should respond in specific instances is akin to supervised learning.

Based on these premises, the answer that we propose relies on learning agents exchanging bits of information in the presence of a shared percept. Communicating about the presence of a concept in a sensory percept is something that humans do, and benefit from since their youngest age. The situation is illustrated in Fig. 5.

5.2 Language for Supervised Training

A very simple schema that would help to communicate a concept[8] from one brain to another is one in which there are many encounters between pairs of learners. In each of them, two learners are faced with a similar percept (e.g., they both see the same scene) and they exchange bits of information about it. These bits can for example be

[8] i.e., communicate the concept as a function that associates an indicator of its presence with all compatible sensory configurations.

indicators of the presence of high-level concepts in the scene. These indicators may reflect the neural activation associated with these high-level concepts. In humans, these bits of information could be encoded through a linguistic convention that helps the receiver of the message interpret them in terms of concepts that it already knows about. One of the most primitive cases of such a communication scenario could occur with animal and human non-verbal communication. For example, an adult animal sees a prey that could be dangerous and emits a danger signal (that could be innate) that a young animal could use as a supervised training signal to associate the prey to danger. Imitation is a very common form of learning and teaching, prevalent among primates, and by which the learner associates contexts with corresponding appropriate behavior. A richer form of communication, which would already be useful, would require simply *naming* objects in a scene. Humans have an innate understanding of the *pointing gesture* that can help identify which object in the scene is being named. In this way, the learner could develop a repertoire of object categories which could become handy (as intermediate concepts) to form theories about the world that would help the learner to survive better. Richer linguistic constructs involve the *combination of concepts* and allow the agents to describe relations between objects, actions and events, sequences of events (stories), causal links, etc., which are even more useful to help a learner form a powerful model of the environment.

This brings us to another hypothesis, supported by **Observation O2** and **Observation O1** and following from the **Abstractions Harder Hypothesis**:

> **Guided Learning Hypothesis.** A human brain can much more easily learn high-level abstractions if guided by the signals produced by other humans, which act as hints or indirect supervision for these high-level abstractions.

This hypothesis is related to much previous work in cognitive science, such as for example *cognitive imitation* [42], which has been observed in monkeys, and where the learner imitates not just a vocalization or a behavior but something more abstract that corresponds to a cognitive rule.

5.3 Learning by Predicting the Linguistic Output of Other Agents

How can a human guide another? By encouraging the learner to predict the "labels" that the teacher verbally associates with a given input configuration X. In the schema of Fig. 5, it is not necessary for the emitter (who produces the utterance) to directly provide supervision to the high-level layers of the receiver (who receives the communication and can benefit from it). An effect similar to supervised learning can be achieved *indirectly* by simply making sure that the receiver's brain include in its training criterion the objective of *predicting what it observes*, which includes not just X but also the linguistic output of the emitter in the context of the shared input percept. In fact, with attentional and emotional mechanisms that increase the

importance given to correctly predicting what other humans say (especially those with whom we have an affective connection), one would approach even more the classical supervised learning setting. Since we have already assumed that the training criterion for human brains involves a term for prediction or maximum likelihood, this could happen naturally, or be enhanced by innate reinforcement (e.g. children pay particular attention to the utterances of their parents). Hence the top-level hidden units h of the receiver would receive a training signal that would encourage h to become good features in the sense of being predictive of the probability distribution of utterances that are received (see Fig. 5). This would be naturally achieved in a model such as the Deep Boltzmann Machine so long as the higher-level units h have a strong connection to "language units" associating both speech heard (e.g., Wernicke's area) and speech produced (e.g., Broca's area), a state of affairs that is consistent with the global wiring structure of human brains. The same process could work for verbal or non-verbal communication, but using different groups of neurons to model the associated observations. In terms of existing learning algorithms one could for example imagine the case of a Deep Boltzmann Machine [39]: the linguistic units get 'clamped' by the external linguistic signal received by the learner, at the same time as the lower-level sensory input units get 'clamped' by the external sensory signal X, and that conditions the likelihood gradient received by the hidden units h, encouraging them to model the joint distribution of linguistic units and sensory units.

One could imagine many more sophisticated communication schemes that go beyond the above scenario. For example, there could be a *two-way exchange* of information. It could be that both agents can potentially learn something from the other in the presence of the shared percept. Humans typically possess different views on the world and the two parties in a communication event could benefit from a two-way exchange. In a sense, language provides a way for humans to summarize the knowledge collected by other humans, replacing "real" examples by indirect ones, thus increasing the range of events that a human brain could model. In that context, it would not be appropriate to simply copy or clone neural representations from one brain to another, as the learner must somehow reconcile the indirect examples provided by the teacher with the world knowledge already represented in the learner's brain. It could be that there is no pre-assigned role of teacher (as emitter) and student (as receiver), but that depending on the confidence demonstrated by each agent for each particular percept, one pays more or less attention to the communicated output of the other. It could be that some aspects of the shared percept are well mastered by one agent but not the other, and vice-versa. Humans have the capability to know that some aspect of a situation is surprising (they would not have predicted it with high probability) and then they should rationally welcome "explanations" provided by others. A way to make the diffusion of useful knowledge more efficient is for the communicating agents to keep track of an estimated degree of "authority" or "credibility" of other agents. One would imagine that parents and older individuals in a human group would by default get more credit, and one of the products of human social systems is that different individuals acquire more or less authority and

credibility. For example, scientists strive to maximize their credibility through very rigorous communication practices and a scientific method that insists on verifying hypotheses through experiments designed to test them.

5.4 Language to Evoke Training Examples at Will

Even more interesting scenarios that derive from linguistic abilities involve our ability to *evoke an input scene*. We do not need to be in front of danger to teach about it. We can describe a dangerous situation and mention what is dangerous about it. In this way, the diffusion of knowledge about the world from human brains to other human brains could be made even more efficient.

The fact that verbal and non-verbal communication between animals and humans happens through a noisy and very bandwidth-limited channel is important to keep in mind. Because very few bits of information can be exchanged, only the most useful elements should be communicated. If the objective is only to maximize collective learning, it seems that there is no point in communicating something that the receiver already knows. However, there may be other reasons why we communicate, such as for smoothing social interactions, acquiring status or trust, coordinating collective efforts, etc.

Note that it is not necessary for the semantics of language to have been defined a priori for the process described here to work. Since each learning agent is trying to predict the utterances of others (and thus, producing similar utterances in the same circumstances), the learning dynamics should converge towards one or more languages which become attractors for the learning agents: the most frequent linguistic representation of a given percept X among the population will tend to gradually dominate in the population. If encounters are not uniformly random (e.g., because the learning agents are geographically located and are more likely to encounter spatially near neighbors), then there could be multiple attractors simultaneously present in the population, i.e., corresponding to multiple spatially localized languages.

5.5 Connection with Curriculum Learning

The idea that learning can be improved by guiding it, by properly choosing the sequence of examples seen by the learner, was already explored in the past. It was first proposed as a practical way to train animals through *shaping* [34, 41], as a way to ease simulated learning of more complex tasks [8, 14, 29] by building on top of easier tasks. An interesting hypothesis introduced in [8] is that a proper choice of training examples can be used to approximate a complex training criterion[9] fraught with local minima with a smoother one (where, e.g., only prototypical examples need to

[9] The training criterion is here seen as a function of the learned parameters, as a sum of an error function over a training distribution of examples.

be shown to illustrate the "big picture"). Gradually introducing more subtle examples and building on top of the already understood concepts is typically done in pedagogy. Bengio et al. [8] propose that the learner goes through a sequence of gradually more difficult learning tasks, in a way that corresponds in the optimization literature to a *continuation method* or an *annealing method*, allowing one to approximately discover global minima (or much better local minima), as illustrated in Fig. 6. Interestingly, it was recently observed experimentally that humans use a form of curriculum learning strategy (starting from easier examples and building up) when they are asked to teach a concept to a robot [28]. Khan et al. [28] also propose a statistical explanation why a curriculum learning strategy can be more successful, based on the uncertainty that the learner has about the relevant factors explaining the variations seen in the data. If these theories are correct, an individual learner can be helped (to escape local minima or converge faster to better solutions) not only by showing examples of abstractions not yet mastered by the learner, but also by showing these well-chosen examples in an appropriate sequence. This sequence corresponds to a curriculum that helps the learner build higher-level abstractions on top of lower-level ones, thus again defeating some of the difficulty believed to exist in training a learner to capture higher-level abstractions.

6 Memes, Crossover, and Cultural Evolution

In the previous section we have proposed a general mechanism by which knowledge can be *transmitted* between brains, without having to actually *copy synaptic strengths*, instead taking advantage of the learning abilities of brains to transfer concepts via examples. We hypothesized that such mechanisms could help an *individual learner* escape an effective local minimum and thus construct a better model of reality, when the learner is guided by the hints provided by other agents about relevant abstractions. But the knowledge had to come from another agent. Where did this knowledge arise in the first place? This is what we discuss here.

6.1 Memes and Evolution from Noisy Copies

Let us first step back and ask how "better[10]" brains could arise. The most plausible explanation is that better brains arise due to some form of search or optimization (as stated in the **Optimization Hypothesis**), in the huge space of brain configurations (architecture, function, synaptic strengths). Genetic evolution is a form of parallel search (with each individual's genome representing a candidate solution) that occurs on a rather slow time-scale. Cultural evolution in humans is also a form of search, in

[10] "better" in the sense of the survival value they provide, and how well they allow their owner to understand the world around them. Note how this depends on the context (ecological and social niche) and that there may be many good solutions.

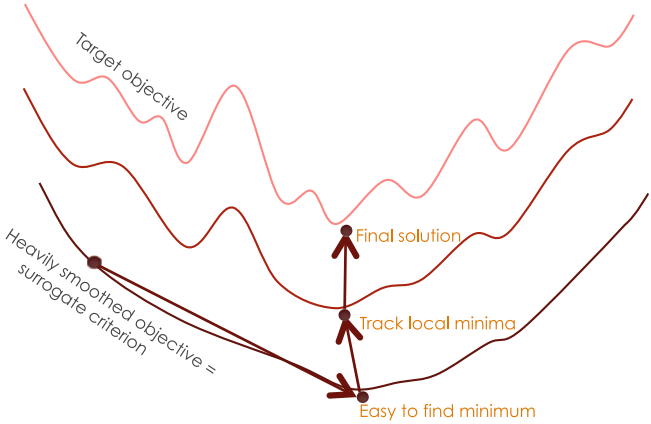

Fig. 6 A general strategy to reduce the impact of local minima is followed in *continuation methods* and *simulated annealing*. The idea is to consider a sequence of optimization problems that start with an easier one for which it is easy to find a global optimum (not corresponding to solving the actual problem of interest, though), with the sequence of problems ending up in the problem of interest, each time starting at the solution previously found with an easier problem and tracking local minima along the way. It was hypothesized [3] and demonstrated with artificial neural networks that following a curriculum could help learners thus find better solutions to the learning problem of interest

the space of ideas or *memes* [12]. A meme is a unit of selection for cultural evolution. It is something that can be copied from one mind to another. Like for genes, the copy can be imperfect. Memes are analogous to genes in the context of cultural evolution [13]. Genes and memes have co-evolved, although it appears that cultural evolution occurs on a much faster scale than genetic evolution. Culture allows brains to modify their basic program and we propose that culture also allows brains to go beyond what a single individual can achieve by simply observing nature. Culture allows brains to take advantage of knowledge acquired by other brains elsewhere and in previous generations.

To put it all together, the knowledge acquired by an individual brain combines four levels of adaptation: **genetic evolution** (over hundreds of thousands of years or more), **cultural evolution** (over dozens, hundreds or thousands of years), **individual learning** and discovery (over minutes, hours and days) and **inference** (fitting the state of mind to the observed perception, over split seconds or seconds). In all four cases, a form of adaptation is at play, which we hypothesize to be associated with a form of approximate optimization, in the same sense as stated in the **Optimization Hypothesis**. One can also consider the union of all four adaptation processes as a global form of evolution and adaptation (see the work of [21] on how learning can guide evolution in the style of Baldwinian evolution). Whereas genetic evolution is a form of parallel search (many individuals carry different combinations and variants of genes which are evaluated in parallel) and we have hypothesized that individual learning is a local search performing an approximate descent (**Local Descent Hypothesis**),

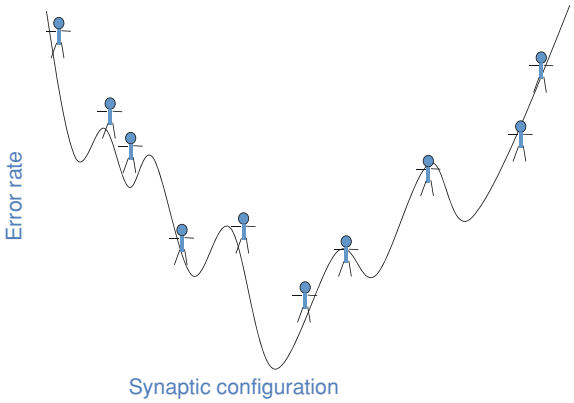

Fig. 7 Illustration of parallel search in the space of synaptic configurations by a population of learners. Some learners start from configurations which happen to lead to a better solution when descending the training criterion

what about cultural evolution? Cultural evolution is based on individual learning, on learners trying to predict the behavior and speech output of individuals, as stated in the **Guided Learning Hypothesis**. Even though individual learning relies on a local descent to gradually improve a single brain, when considering the graph of interactions between humans in an evolving population, one must conclude that cultural evolution, like genetic evolution, is a form of parallel search, as illustrated in Fig. 7.

The most basic working principle of evolution is the *noisy copy* and it is also at work in cultural evolution: a meme can be noisily copied from one brain to another, and the meme can sometimes be slightly modified in the process.[11] A meme exists in a human's brain as an aspect of the dynamics of the brain's neural network, typically allowing the association of words in language (which are encoded in specific areas of the brain) with high-level abstractions learned by the brain (which may be encoded in other cortical areas, depending the semantics of the meme). The meme is activated when neural configurations associated with it arise, and different memes are also connected to each other in the sense of having a high probability of being associated together and echoing each other through thoughts, reasoning, or planning.

Selective pressure then does the work of exponentially increasing the presence of successful memes in the population, by increasing the chances that a successful meme be copied in comparison with a competing less successful meme. This may happen simply because a useful meme allows its bearer to survive longer, commu-

[11] Remember that a meme is copied in a process of teaching by example which is highly stochastic, due to the randomness in encounters (in which particular percepts serve as examples of the meme) and due to the small number of examples of the meme. This creates a highly variable randomly distorted version of the meme in the learner's brain.

nicate with more individuals, or because better ideas are promoted.[12] With genetic evolution, it is necessary to copy a whole genome when the individual bearing it is successful. Instead, cultural evolution in humans has mechanisms to evaluate an *individual meme* and selectively promote it. Good ideas are more likely to be the subject of discussion in public communication, e.g., in the public media, or even better in scientific publications. Science involves powerful mechanisms to separate the worth of a scientist from the worth of his or her ideas (e.g. through independent replication of experimental results or theoretical proofs, or through blind reviewing). That may explain why the pace of evolution of ideas has rapidly increased since the mechanisms for scientific discovery and scientific dissemination of memes have been put in place. The fact that a good idea can stand on its own and be selected for its own value means that the *selective pressure* is much more efficient because it is less hampered by the noisy evaluation that results when fitness is assigned to a whole individual, that integrates many memes and genes.

In this context the premium assigned to novelty in some cultures, in particular in scientific research, makes sense as it favors novel memes that are farther away from existing ones. By increasing the degree of exploration through this mechanism, one might expect that it would yield more diversity in the solutions explored, and thus more efficient search (finding good ideas faster) may be achieved with appropriate amounts of this premium for novelty.

6.2 Fast-Forward with Divide-and-Conquer from Recombination

But if evolution only relied on the noisy copy principle, then it could only speed-up search at best linearly with respect to the number of individuals in a population. Instead of trying N random configurations with N individuals and picking the best by selective pressure, a population with $M > N$ individuals would discover a good selection M/N times faster in average. This is useful but we hypothesize that it would not be enough to make a real dent in the optimization difficulty due to a huge number of poor effective local minima in the space of synaptic configurations. In fact, evolution has discovered an evolutionary mechanism which can yield much larger speed-ups, and is based on *sexual reproduction* in the case of genetic evolution. With sexual reproduction, we have an interaction between two parent individuals (and their associated candidate configurations), and we mix some of the genes of one with some of the genes of the other in order to create new combinations that are not near-neighbors of either parent. This is very different from a simple parallel search because it can explore new configurations beyond local variations around the randomly initialized starting stock. Most importantly, a *recombination operator* can *combine good, previously found, sub-solutions*. Maybe your father had exceptionally good genes for eyes and your mother exceptionally good genes for ears, and with about 25 % probability

[12] Selfish memes [12, 13] may also strive in a population: they do not really help the population but they nonetheless maintain themselves in it by some form of self-promotion or exploiting human weaknesses.

you could get both, and this may confer you with an advantage that no one had had before. This kind of transformation of the population of candidate configurations is called a *crossover operator* in the genetic algorithms literature [24]. Crossover is a recombination operator: it can create new candidate solutions by combining *parts* of previous candidate solutions. Crossover and other operators that *combine existing parts of solutions to form new candidate solutions* have the potential for a much greater speed-up than simple parallelized search based only on individual local descent (noisy copy). This is because such operators can potentially exploit a form of *divide-and-conquer*, which, if well done, could yield exponential speed-up. For the divide-and-conquer aspect of the recombination strategy to work, it is best if *sub-solutions that can contribute as good parts to good solutions receive a high fitness score*. As is well known in computer science, divide-and-conquer strategies have the potential to achieve an *exponential speedup* compared to strategies that require blindly searching through potential candidate solutions (synaptic configurations, here). The exponential speedup would be achieved if the optimization of each of the combined parts (memes) can be done independently of the others. In practice, this is not going to be the case, because memes, like genes, only take value in the context and presence of other memes in the individual and the population.

The success rate of recombination is also important i.e., what fraction of the recombination offsprings are viable? The *encoding* of information into genes has a great influence on this success rate as well as on the fitness assigned to good sub-solutions. We hypothesize that memes are particularly good units of selection in these two respects: they are by definition the units of cultural information that can be meaningfully recombined to form new knowledge. All these ideas are summarized in the following hypothesis.

Memes Divide-and-Conquer Hypothesis. Language, individual learning, and the recombination of memes constitute an efficient evolutionary recombination operator, and this gives rise to rapid search in the space of memes, that helps humans build up better high-level internal representations of their world.

6.3 Where Do New Ideas Come from?

Where do completely new ideas (and memes) emerge? According to the views stated here, they emerge from two intertwined effects. On the one hand, our brain can easily combine into new memes different memes which it inherited from other humans, typically through linguistic communication and imitation. On the other hand, such recombination as well as other creations of new memes must arise from the optimization process taking place in a single learning brain, which tries to reconcile all the sources of evidence that it received into some kind of unifying theory. This search is local in parameter space (synaptic weights) but can involve a stochastic search in the space of neuronal firing patterns (state of mind). For example, in a

Boltzmann machine, neurons fire randomly but with a probability that depends on the activations of other connected neurons, and so as to explore and reach more plausible "interpretations" of the current and past observations (or "planning" for future actions in search for a sequence of decisions that would give rise to most beneficial outcomes), given the current synaptic strenghts. In this stochastic exploration, new configurations of neuronal activation can randomly arise and if these do a better job of explaining the data (the observations made), then synaptic strengths will change slightly to make these configurations more likely in the future. This is already how some artificial neural networks learn and "discover" concepts that explain their input. In this way, we can see "concepts" of edges, parts of face, and faces emerge from a deep Bolzmann machine that "sees" images of faces [31].

What this means is that the recombination operator for memes is doing much more than recombination in the sense of cutting and pasting parts together. It does that but it is also possible for the new combinations to be *optimized* in individual brains (or even better, by groups who create together) so as to better fit the empirical evidence that each learner has access to. This is related to the ideas in [21] where a global search (in their case evolution) is combined with a local search (individual learning). This has the effect of smoothing the fitness function seen by the global search, by allowing half-baked ideas (which would not work by themselves) to be tuned into working ones.

7 Conclusion and Future Work

To summarize, motivated by theoretical and empirical work on Deep Learning, we developed a theory starting from the hypothesis that high-level abstractions are difficult to learn because they need to be represented with highly non-linear computation associated with enough levels of representation, and that this difficulty corresponds to the learner getting stuck around effective local minima. We proposed and argued that other learning agents can provide new examples to the learner that effectively change the learner's training criterion into one where these difficult effective local minima can be avoided. This happens because the communications from other agents can provide a kind of indirect supervision to higher levels of the brain, which makes the task of discovering explanatory factors of variation (i.e., modeling the rest of the observed data) much easier. Furthermore, this brain-to-brain communication mechanism allows brains to recombine nuggets of knowledge called memes. Individual learning corresponds to searching for such recombinations and other variations of memes that are good at explaining the data observed by learners. In this way, new memes are created that can be disseminated in the population if other learning agents value them, creating cultural evolution. Like genetic evolution, cultural evolution efficiently searches (in the space of memes, rather than genes) thanks to parallelism, noisy copying of memes, and creative recombination and optimizations of memes. We hypothesize that this phenomenon provides a divide-and-conquer advantage that

yields much greater speedup in the optimization performed, compared to the linear speedup obtained simply from parallelization of the search across a population.

A lot more needs to be done to connect the above hypotheses with the wealth of data and ideas arising in the biological and social sciences. They can certainly be refined and expanded into more precise statements. Of central importance to future work following up on this chapter is how one could go and *test these hypotheses*. Although many of these hypotheses agree with common sense, it would be worthwhile verifying them empirically, to the extent this is possible. It is also quite plausible that many supporting experimental results from neuroscience, cognitive science, anthropology or primatology already exist that support these hypotheses, and future work should cleanly make the appropriate links.

To test the **Optimization Hypothesis** would seem to require estimating a criterion (not an obvious task) and verifying that learning improves it in average. A proxy for this criterion (or its relative change, which is all we care about, here) might be measurable in the brain itself, for example by measuring the variation in the presence of reward-related molecules or the activity of neurons associated with reward. The effect of learning could be tested with a varying number of training trials with respect to a rewarding task.

If the **Optimization Hypothesis** is considered true, testing the additional assumptions of the **Local Descent Hypothesis** is less obvious because it is difficult to measure the change in synaptic strengths in many places. However, a form of *stability* of synaptic strengths is a sufficient condition to guarantee that the optimization has to proceed by small changes.

There is already evidence for the **Deep Abstraction Hypothesis** in the visual and auditory cortex, in the sense that neurons that belong to areas further away from the sensory neurons seem to perform a higher-level function. Another type of evidence comes from the time required to solve different cognitive tasks, since the hypothesis would predict that tasks requiring computation for the detection of more abstract concepts would require longer paths or more "iterations" in the recurrent neural network of the brain.

The **Local Minima Hypothesis** and the **Abstractions Harder Hypothesis** are ethically difficult to test directly but are almost corollaries of the previous hypotheses. An indirect source of evidence may come from raising a primate without any contact with other primates nor any form of guidance from humans, and measure the effect on operational intelligence at different ages. One problem with such an experiment would be that other factors might also explain a poor performance (such as the effect of psychological deprivation from social support, which could lead to depression and other strong causes of poor decisions), so the experiment would require a human that provides warmth and caring, but no guidance whatsoever, even indirectly through imitation. Choosing a more solitary species such as the orangutan would make more sense here (to reduce the impacts due to lack of social support). The question is whether the tested primate could learn to survive as well in the wild as other primates of the same species.

The **Guided Learning Hypothesis** could already be supported by empirical evidence of the effect of education on intelligence, and possibly by observations of

feral (wild) children. The important point here is that the intelligence tests chosen should not be about reproducing the academic knowledge acquired during education, but about decisions where having integrated knowledge of some learned high-level abstractions could be useful to properly interpret a situation and take correspondingly appropriate decisions. Using computational simulations with artificial neural networks and machine learning one should also test the validity of mechanisms for "escaping" local minima thanks to "hints" from another agent.

The **Memes Divide-and-Conquer Hypothesis** could probably be best tested by computational models where we simulate learning of a population of agents that can share their discoveries (what they learn from data) by communicating the high-level abstractions corresponding to what they observe (as in the scenario of Sect. 5 and Fig. 5). The question is whether one could set up a linguistic communication mechanism that would help this population of learners converge faster to good solutions, compared to a group of isolated learning individuals (where we just evaluate a group's intelligence by the fitness, i.e. generalization performance, of the best-performing individual after training). Previous computational work on the evolution of language is also relevant, of course. If such algorithms would work, then they could also be useful to advance research in machine learning and artificial intelligence, and take advantage of the kind of massive and loose parallelism that is more and more available (to compensate for a decline in the rate of progress of the computing power accessible by a single computer core). This type of work is related to other research on algorithms inspired by the evolution of ideas and culture (see the *Wikipedia* entry on *Memetic Algorithms* and [25, 26, 33]).

If many of these hypotheses (and in particular this last one) are true, then we should also draw conclusions regarding the *efficiency of cultural evolution* and how different *social structures* may influence that efficiency, i.e., yield greater group intelligence in the long run. Two main factors would seem to influence this efficiency: (1) the efficiency of exploration of new memes in the society, and (2) the rate of spread of good memes. Efficiency of exploration in meme-space would be boosted by a greater investment in scientific research, especially in high-risk high potential impact areas. It would also be boosted by encouraging diversity it all its forms because it would mean that individual humans explore a less charted region of meme-space. For example, diversity would be boosted by a non-homogeneous education system, a general bias favoring openness to new ideas and multiple schools of thought (even if they disagree), and more generally to marginal beliefs and individual differences. The second factor, the rate of spread of good memes, would be boosted by communication tools such as the Internet, and in particular by open and free access to education, information in general, and scientific results in particular. The investment in education would probably be one of the strongest contributors of this factor, but other interesting contributors would be social structures making it easy for every individual to disseminate useful memes, e.g., to publish on the web, and the operation of non-centralised systems of rating what is published (whether this is scientific output or individual blogs and posts on the Internet), helping the most interesting new ideas to bubble up and spread faster, and contributing both to diversity of new memes and more efficient dissemination of useful memes. Good rating systems could

help humans to detect selfish memes that "look good" or self-propagate easily for the wrong reasons (like cigarettes or sweets that may be detrimental to your health even though many people are attracted to them), and the attempts at objectivity and replicability that scientists are using may help there.

Acknowledgments The author would like to thank Caglar Gulcehre, Aaron Courville, Myriam Côté, and Olivier Delalleau for useful feedback, as well as NSERC, CIFAR and the Canada Research Chairs for funding.

References

1. D.H. Ackley, G.E. Hinton, T.J. Sejnowski, A learning algorithm for Boltzmann machines. Cogn. Sci. **9**, 147–169 (1985)
2. M.A. Arbib, *The Handbook of Brain Theory and Neural Networks* (MIT Press, Cambridge, 1995)
3. Y. Bengio, Learning deep architectures for AI. Found. Trends Mach. Lear. **2**(1), 1–127 2009. Also published as a book. Now Publishers, 2009
4. Y. Bengio, O. Delalleau, On the expressive power of deep architectures, in *Proceedings of the 22nd International Conference on Algorithmic Learning Theory*, 2011, ed. by J. Kivinen, C. Szepesvári, E. Ukkonen, T. Zeugmann
5. Y. Bengio, O. Delalleau, C. Simard, Decision trees do not generalize to new variations. Comput. Intell. **26**(4), 449–467 (2010)
6. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in *Advances in Neural Information Processing Systems 19* (*NIPS'06*), ed. by B. Schölkopf, J. Platt, T. Hoffman (MIT Press, Cambridge, 2007), pp. 153–160
7. Y. Bengio, Y. LeCun, Scaling learning algorithms towards AI. in *Large Scale Kernel Machines*, ed. by L. Bottou, O. Chapelle, D. DeCoste, J. Weston (MIT Press, Cambridge, 2007)
8. Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in *Proceedings of the Twenty-sixth International Conference on Machine Learning* (*ICML'09*), ed. by L. Bottou, M. Littman (ACM, 2009)
9. L. Bottou, Stochastic learning, in *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture notes in artificial intelligence, ed. by O. Bousquet, U. von Luxburg (Springer, Berlin, 2004), pp. 146–168
10. M.A. Carreira-Perpiñan, G.E. Hinton, On contrastive divergence learning, in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics* (*AISTATS'05*), ed. by R.G. Cowell, Z. Ghahramani (Society for Artificial Intelligence and Statistics, 2005) pp. 33–40.
11. R. Caruana, Multitask connectionist learning, in *Proceedings of the 1993 Connectionist Models Summer School*, 1993, pp. 372–379
12. R. Dawkins, *The Selfish Gene* (Oxford University Press, London, 1976)
13. K. Distin, *The Selfish Meme* (Cambridge University Press, London, 2005)
14. J.L. Elman, Learning and development in neural networks: the importance of starting small. Cognition **48**, 781–799 (1993)
15. D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? J. Mach. Lear. Res. **11**, 625–660 (2010)
16. J. Håstad, Almost optimal lower bounds for small depth circuits, in *Proceedings of the 18th annual ACM Symposium on Theory of Computing* (ACM Press, Berkeley, 1986), pp. 6–20
17. J. Håstad, M. Goldmann, On the power of small-depth threshold circuits. Comput. Complex. **1**, 113–129 (1991)

18. G.E. Hinton, T.J. Sejnowski, D.H. Ackley, Boltzmann machines: constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, (Dept. of Computer Science, Carnegie-Mellon University, 1984)
19. G.E. Hinton, Learning distributed representations of concepts, in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (Lawrence Erlbaum, Hillsdale, Amherst 1986, 1986), pp. 1–12
20. G.E. Hinton, Connectionist learning procedures. Artif. Intell. **40**, 185–234 (1989)
21. G.E. Hinton, S.J. Nowlan, How learning can guide evolution. Complex Syst. **1**, 495–502 (1989)
22. G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets. Neural Comput. **18**, 1527–1554 (2006)
23. G.E. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
24. J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, 1975)
25. E. Hutchins, B. Hazlehurst, How to invent a lexicon: the development of shared symbols in interaction, in *Artificial Societies: The Computer Simulation of Social Life*, ed. by N. Gilbert, R. Conte (UCL Press, London, 1995), pp. 157–189
26. E. Hutchins, B. Hazlehurst, Auto-organization and emergence of shared language structure, in *Simulating the Evolution of Language*, ed. by A. Cangelosi, D. Parisi (Springer, London, 2002), pp. 279–305
27. K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition? in *Proceedings of IEEE International Conference on Computer Vision* (*ICCV'09*), 2009, pp. 2146–2153
28. F. Khan, X. Zhu, B. Mutlu, How do humans teach: on curriculum learning and teaching dimension, in *Advances in Neural Information Processing Systems 24* (*NIPS'11*), 2011 pp. 1449–1457
29. K.A. Krueger, P. Dayan, Flexible shaping: how learning in small steps helps. Cognition **110**, 380–394 (2009)
30. H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks. J. Mach. Lear. Res. **10**, 1–40 (2009)
31. H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, ed. by L. Bottou, M. Littman (ACM, Montreal (Qc), Canada, 2009)
32. J. Martens. Deep learning via Hessian-free optimization, in *Proceedings of the Twenty-seventh International Conference on Machine Learning* (*ICML-10*), ed. by L. Bottou, M. Littman (ACM, 2010) pp. 735–742
33. E. Moritz, Memetic science: I–general introduction. J. Ideas **1**, 1–23 (1990)
34. G.B. Peterson, A day of great illumination: B. F. Skinner's discovery of shaping. J. Exp. Anal. Behav. **82**(3), 317–328 (2004)
35. R. Raina, A. Battle, H. Lee, B. Packer, A.Y. Ng, Self-taught learning: transfer learning from unlabeled data, in *Proceedings of the Twenty-fourth International Conference on Machine Learning* (*ICML'07*), ed. by Z. Ghahramani (ACM, 2007), pp. 759–766
36. M. Ranzato, C. Poultney, S. Chopra, Y. LeCun, Efficient learning of sparse representations with an energy-based model, in *Advances in Neural Information Processing Systems 19* (*NIPS'06*), ed. by B. Schölkopf, J. Platt, T. Hoffman (MIT Press, 2007) pp. 1137–1144
37. D.E. Rumelhart, J.L. McClelland, and the PDP Research Group *Parallel Distributed Processing Explorations in the Microstructure of Cognition*, (MIT Press, Cambridge, 1986)
38. R. Salakhutdinov, G.E. Hinton, Deep Boltzmann machines, in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics* (*AISTATS 2009*), vol. 8, 2009
39. R. Salakhutdinov, G.E. Hinton, Deep Boltzmann machines. in *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics* (*AISTATS'09*), vol. 5, 2009, pp. 448–455
40. R. Salakhutdinov, H. Larochelle, Efficient learning of deep Boltzmann machines, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (*AISTATS 2010*), JMLR W&CP, vol. 9, 2010, pp. 693–700

41. B.F. Skinner, Reinforcement today. Am. Psychol. **13**, 94–99 (1958)
42. F. Subiaul, J. Cantlon, R.L. Holloway, H.S. Terrace, Cognitive imitation in rhesus macaques. Science **305**(5682), 407–410 (2004)
43. R. Sutton, A. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, 1998)
44. J. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000)
45. L. van der Maaten, G.E. Hinton, Visualizing data using t-sne. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
46. J. Weston, F. Ratle, R. Collobert, Deep learning via semi-supervised embedding, in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, ed. by W.W. Cohen, A. McCallum, S.T. Roweis (ACM, New York, NY, USA, 2008), pp. 1168–1175
47. A. Yao, Separating the polynomial-time hierarchy by oracles, in *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, 1985, pp. 1–10
48. A.L. Yuille, The convergence of contrastive divergences, in *Advances in Neural Information Processing Systems 17* (*NIPS'04*), ed. by L.K. Saul, Y. Weiss, L. Bottou (MIT Press, 2005) pp. 1593–1600

Chapter 4
Learning Sparse Features with an Auto-Associator

Sébastien Rebecchi, Hélène Paugam-Moisy and Michèle Sebag

Abstract A major issue in statistical machine learning is the design of a representation, or feature space, facilitating the resolution of the learning task at hand. Sparse representations in particular facilitate discriminant learning: On the one hand, they are robust to noise. On the other hand, they disentangle the factors of variation mixed up in dense representations, favoring the separability and interpretation of data. This chapter focuses on auto-associators (AAs), i.e. multi-layer neural networks trained to encode/decode the data and thus de facto defining a feature space. AAs, first investigated in the 80s, were recently reconsidered as building blocks for deep neural networks. This chapter surveys related work about building sparse representations, and presents a new non-linear explicit sparse representation method referred to as *Sparse Auto-Associator* (SAA), integrating a sparsity objective within the standard auto-associator learning criterion. The comparative empirical validation of SAAs on state-of-art handwritten digit recognition benchmarks shows that SAAs outperform standard auto-associators in terms of classification performance and yield similar results as denoising auto-associators. Furthermore, SAAs enable to control the representation size to some extent, through a conservative pruning of the feature space.

S. Rebecchi · M. Sebag
CNRS, LRI UMR 8623, TAO, INRIA Saclay, Université Paris-Sud 11, 91405 Orsay, France
e-mail: sebastien.rebecchi@lri.fr

M. Sebag
e-mail: michele.sebag@lri.fr

H. Paugam-Moisy (✉)
CNRS, LIRIS UMR 5205, Université Lumière Lyon 2, 69676 Bron, France
e-mail: helene.paugam- oisy@liris.cnrs.fr

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_4,

1 Introduction

A major issue in statistical machine learning is the design of a representation, or *feature space*, facilitating the resolution of the learning task at hand. For instance, binary supervised learning often considers linear hypotheses, although a hyper-plane actually separating the two classes seldom exists for real-world data (non-separable data). A standard approach thus consists of mapping the initial input space onto a usually high-dimensional feature space, one favoring data separability. The feature space can be (i) explicitly defined using hand-crafted features; (ii) implicitly defined using the so-called kernel trick [32]; (iii) automatically learned by optimizing a criterion involving the available data. In the former two cases, the feature space ultimately relies on the user's expertise and involves a trial-and-error process, particularly so when dealing with high-dimensional data (e.g. images). In the third case, the feature space optimization considers either a linear search space (dictionary learning [26]) or a non-linear one, through neural networks.

This chapter focuses on the trade-off between the expressiveness and the computational efficiency of non-linear feature space optimization. On the one hand, non-linear functions can be represented more efficiently and compactly (in terms of number of parameters) by iteratively composing non-linear functions, defining deep architectures where each layer defines a more complex feature space elaborated on the basis of the previous one [3]. Indeed, the principle of hierarchically organized representations—corresponding to increasing levels of abstraction—is appealing from a cognitive perspective [31]. The direct optimization of deep architectures, however, raises severe challenges in a supervised setting: while the standard back-propagation of the gradient is effective for shallow neural architectures, the gradient information is harder to exploit when the number of layers increases. The breakthrough of deep neural architectures since the mid-2000s [4, 15] relies on a new training principle, based on the layer-wise unsupervised training of each layer. Note that the specific layer training depends on the particular setting, involving restricted Boltzmann machines [15], auto-associators [4] or convolutional networks [22, 25]. Only at a later stage is the pre-trained deep neural network architecture optimized (fine-tuned) using a supervised learning criterion (Fig. 1). The chapter focuses on feature space optimization within the auto-associator-based deep neural network setting.

Formally, an auto-associator (AA) is a one-hidden layer neural network implementing an encoding-decoding process [5, 13]. The AA is trained by backpropagation to reconstruct the instances in the dataset. Its hidden layer thus defines a feature space. In the context of deep networks, this first feature-based representation of the data is used to feed another AA, which is likewise trained as an encoder/decoder, thus defining a second layer feature space. The process is iterated in a layer-wise manner, thus defining a deep neural architecture. At each layer, the feature space can be assessed from its coding/decoding performance, a.k.a. the average reconstruction error or *accuracy* on the one hand, and its size or complexity on the other hand. As

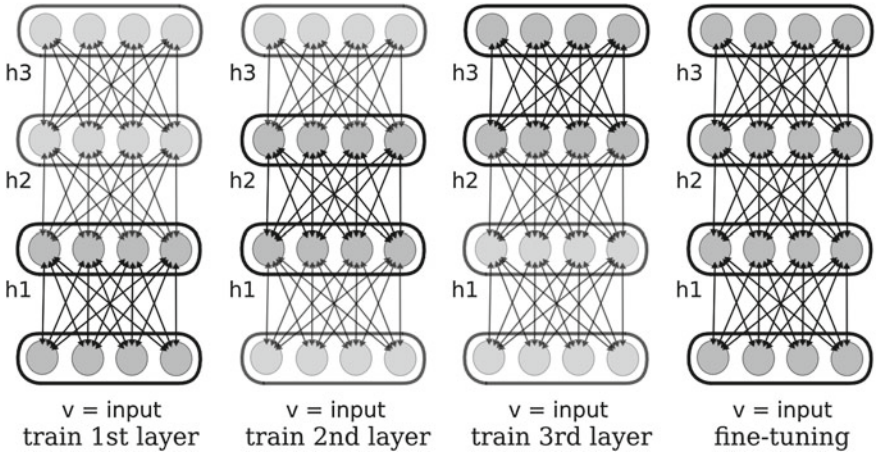

Fig. 1 Layer-wise training scheme of deep neural networks (e.g. stacked auto-associators or stacked RBMs) and overall fine-tuning

might be expected, the quality of k-th layer feature space dictates the quality of the $(k+1)$-th layer feature space, specifically so in terms of reconstruction error.

The sparsity of the feature space, that is, the fact that the coding of each data instance involves only a few features, is highly desirable for several reasons [3, 10, 28]. Firstly, sparse representations offer a better robustness w.r.t. the data noise. Secondly, they facilitate the interpretation of the data by disentangling the factors of variations mixed up in dense representations, and thus favor the linear separability of the data. Along this line, the complexity of the feature space might be assessed from its *sparsity*, i.e. the average number of features that are actively involved in example coding.

Formally, let $\mathbf{x}$ denote an instance in $\mathbb{R}^n$, let $\mathbf{y} \in \mathbb{R}^m$ denote its mapping onto the feature space. Feature i $(i = 1 \ldots m)$ is said to be active in $\mathbf{y}$ iff its cancelling out significantly decreases the ability to reconstruct $\mathbf{x}$ from $\mathbf{y}$. Sparse coding, involving a low percentage of active features on average from the data, has been intensively studied in the last decade in relation with compressed sensing [6], signal decomposition [7] and dictionary learning [26]. The presented approach, referred to as *Sparse Auto-Associator* (SAA), focuses on integrating a sparsity-inducing approach within the auto-associator framework.

The chapter is organized as follows. Section 2 presents several points of view related to learning sparse representations. The domain of sparse feature space learning has been advancing at a fast pace in the recent years, and the present chapter has no intention of presenting an exhaustive survey. The standard auto-associator model and its best known variant, the denoising auto-associator (DAA), are introduced in Sect. 3. Section 4 describes the proposed SAA algorithm, based on an alternate non-linear optimization process enforcing both accuracy and sparsity criteria. The Sparse

Auto-Associator is experimentally validated in Sect. 5 and discussed in Sect. 6. Section 7 summarizes the contributions and presents some perspectives on non-linear learning of sparse features.

2 Learning Sparse Representations of Data

Originated from signal processing, the *sparse coding* domain aims at expressing signals using a (given or learned) *dictionary*. This section briefly reviews some recent work related to sparse coding, in particular within the field of neural networks.

2.1 Dictionary-Based Sparse Coding

Sparse coding of an (n-dimensional) raw data signal $\mathbf{x}$ refers to the attempt at finding a linear decomposition of $\mathbf{x}$ using a small number of atoms of a given dictionary $\mathbf{D}$. Formally, $\mathbf{D}$ is a $(n \times m)$ real matrix, its m columns vectors $\mathbf{D_{*j}}$ denoting the atoms of the dictionary. The sparse code $\mathbf{y}$ is obtained by solving a combinatorial optimization problem, minimizing its L_0-norm $\|.\|_0$ subject to the reconstruction constraint:

$$\min_{\mathbf{y}\in\mathbb{R}^m} \|\mathbf{y}\|_0 \quad \text{subject to} \quad \mathbf{x} = \sum_{j=1}^{m} (\mathbf{D_{*j}} y_j), \tag{1}$$

where y_j denotes the j-th entry of $\mathbf{y}$. An approximate resolution of Eq. (1) is yielded by the greedy Orthogonal Matching Pursuit (OMP) algorithm [30], subject to upper bounding $\|\mathbf{y}\|_0$.

A convex relaxation of Eq. (1) is obtained by replacing the L_0-norm by the L_1-norm and minimizing a weighted sum of the reconstruction loss and the sparsity:

$$\min_{\mathbf{y}\in\mathbb{R}^m} \left(\left\| \mathbf{x} - \sum_{j=1}^{m} (\mathbf{D_{*j}} y_j) \right\|_2^2 + \lambda \|\mathbf{y}\|_1 \right), \tag{2}$$

where $\lambda > 0$ is the trade-off parameter, $\|.\|_1$ and $\|.\|_2$ denote the L_1-norm and L_2-norm, respectively. The minimization problem defined by Eq. (2) is known as Basis Pursuit (BP) denoising [7]. Under some conditions [9], the unique solution of Eq. (2) also is a solution of Eq. (1).

In both cases the objective is to find an approximate decomposition of $\mathbf{x}$ involving a small number of atoms of $\mathbf{D}$. The level of activity of $\mathbf{D_{*j}}$ relative to $\mathbf{x}$ is $|y_j|$. Cancelling the entry j, i.e. ignoring $\mathbf{D_{*j}}$ in the reconstruction process, consists in setting y_j to 0, thus incurring some additional reconstruction loss, the level of which

increases with $|y_j|$. Cancelling entries with very low levels of activity clearly entails a small loss of accuracy.

As a given dictionary may not provide every example with a sparse decomposition, it is natural to optimize the dictionary depending on the available examples [1, 26]. If sparsity is measured w.r.t. L_1-norm then *dictionary learning* for sparse coding is achieved by solving the following joint optimization problem:

$$\min_{\mathbf{D}\in\mathbb{R}^{n\times m},\mathbf{y}\in\mathbb{R}^{m\times l}} \left(\|\mathbf{x}-\mathbf{D}\mathbf{Y}\|_2^2 + \lambda \sum_{k=1}^{l} \|\mathbf{Y}_{*\mathbf{k}}\|_1 \right), \tag{3}$$

with l the number of training examples, $\mathbf{x}$ the $(n \times l)$ real matrix storing the training examples in columns and $\mathbf{y}$ the $(m \times l)$ real matrix storing the sparse representation of the k-th training example $\mathbf{X}_{*\mathbf{k}}$ in its k-th column $\mathbf{Y}_{*\mathbf{k}}$.

Besides the general merits of sparse coding, mentioned in Sect. 1, sparse dictionary coding features specific strengths and weaknesses. On the positive side, the minimization-based formulation in Eq. (3) yields a variable-size representation, since some examples may be reconstructed from very few dictionary atoms (i.e.active components) while other examples may require many more dictionary atoms. On the negative side, dictionary learning defines an implicit and computationally expensive sparse coding; formally, each new $\mathbf{x}$ requires a BP optimization problem (Eq. (2)) to be solved to find the associated code $\mathbf{y}$. A second limitation of sparse dictionary learning is to be restricted to linear coding.

2.2 Sparse Coding within a Neural Network

Aimed at overcoming both above limitations of dictionary learning, several neural network-based architectures have been defined to achieve sparse coding through considering additional learning criteria. The main merit of this is to yield a non-linear coding, directly computable for unseen patterns without solving any additional optimization problem. Another merit of such models is to accommodate the iterative construction of sparse non-linear features within a Deep Network architecture.

As already mentioned, sparse coding within the deep learning framework is a hot topic. While not an exhaustive review of the state of the art, this section summarizes the main four directions of research, which will be discussed comparatively to the proposed approach in Sect. 6.2.

Gregor and LeCun use a supervised approach to enforce sparse coding, where each input example is associated its optimal sparse representation (obtained through conventional dictionary-based methods). The neural encoder is trained to approximate this optimal code [12]. The proposed architecture interestingly involves a *local competition* between encoding neurons; specifically, if two (sets of neurons) can reconstruct an input equally well, the algorithm activates only one of them and deactivates the other.

Another strategy for encouraging sparse codes in a neuron layer is to apply back-propagation learning with an activation function that naturally favors sparsity, such as proposed by Glorot et al. with the so-called *rectifying* functions [10]. Rectifying neurons are considered more biologically plausible than sigmoidal ones. From the engineering viewpoint, rectifying neurons can be used in conjunction with an additional constraint, such as L_1-norm regularization, to further promote sparsity. Rectifying functions however raise several computational issues, adversely affecting the gradient descent used for optimization. The authors propose several heuristics to tackle these issues.

Lee et al. augment the Restricted Boltzmann Machine (RBM) learning rule [15] by setting a *temporal* sparsity constraint on each individual encoding neuron [24]. This constraint requires the average value ρ of encoding neuron activations to be close to the minimum of the activation $\hat{\rho}_i$ of each encoding neuron n_i. In other words, each encoding neuron is forced to be active for a small number of input examples only. By limiting the average activity of each neuron, this *selectivity* property tends to favor the code sparsity, as it makes it unlikely that many encoding neurons are active for many examples. In particular, a method for updating selectivity consists in minimizing the cross-entropy loss between ρ and $\hat{\rho}_i$ [14].

Goh et al. propose another modification to the RBM learning rule in order to simultaneously favor both the sparsity and the selectivity of the model [11]. The method consists in computing a matrix $\mathbf{M}$ which stores the training example components in its columns and the encoding neuron activations in its rows, so that the *selectivity* (resp. *sparsity*) property can be evaluated horizontally (resp. vertically). $\mathbf{M}$ is modified in order to fit some target distribution P before updating the model. P encodes the prior knowledge about the problem domain: e.g. for image datasets, the authors consider a distribution P which is positively skewed with heavy tails, since similar characteristics of activity distribution for both selectivity and sparsity have been observed from recordings in biological neural networks. Notably, this approach requires batch learning.

After briefly introducing auto-associator (AA) learning for the sake of completeness (Sect. 3), the rest of the chapter will present a new way of enforcing sparsity within AA learning (Sect. 4) and report on its comparative validation (Sect. 5).

3 Auto-Associator and Denoising Auto-Associator

The AA model, auto-associator a.k.a. auto-encoder, is a two-layer neural network trained for feature extraction [5, 13].

The first layer is the *encoding layer* or *encoder* and the second layer is the *decoding layer* or *decoder*. Given an input example $\mathbf{x}$, the goal of an AA is to compute a code $\mathbf{y}$ from which $\mathbf{x}$ can be recovered with high accuracy, i.e. to model a two-stage approximation to the identity function:

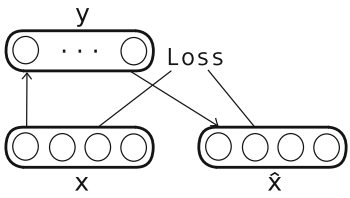

Fig. 2 The training scheme of an AA. The features **y** are learned to encode the input **x** and are decoded into $\hat{\mathbf{x}}$ in order to reconstruct **x**

$$\begin{cases} \mathbf{y} = f^{\mathrm{E}}(\mathbf{x}) = a^{\mathrm{E}}(\mathbf{W}^{\mathrm{E}}\mathbf{x} + \mathbf{b}^{\mathrm{E}}), \\ \hat{\mathbf{x}} = f^{\mathrm{D}}(\mathbf{y}) = a^{\mathrm{D}}(\mathbf{W}^{\mathrm{D}}\mathbf{y} + \mathbf{b}^{\mathrm{D}}), \\ \hat{\mathbf{x}} \simeq \mathbf{x}, \end{cases} \tag{4}$$

where f^{E} and f^{D} denote the function computed by the encoder and decoder, respectively. Parameters are the weight matrices, bias vectors and activation functions, respectively denoted by $\Theta^{\mathrm{E}} = \{\mathbf{W}^{\mathrm{E}}, \mathbf{b}^{\mathrm{E}}\}$ and a^{E} for the encoder, and $\Theta^{\mathrm{D}} = \{\mathbf{W}^{\mathrm{D}}, \mathbf{b}^{\mathrm{D}}\}$ and a^{D} for the decoder. The weight matrix $\mathbf{W}^{\mathrm{D}}$ is often (although not necessarily) the transpose of $\mathbf{W}^{\mathrm{E}}$. Since the target output of an AA is the same as its input, the decoder output dimension (number of neurons) equals the encoder input dimension.

The training scheme of an AA (Fig. 2) consists in finding parameters $\Theta = \Theta^{\mathrm{E}} \cup \Theta^{\mathrm{D}}$ (weights and biases) that minimize a reconstruction loss on a training dataset $\mathcal{S}$, with the following objective function:

$$\phi(\Theta) = \sum_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\mathbf{x})), \tag{5}$$

where $\mathcal{L}$ denotes the loss function.

An AA is usually trained by gradient descent, applying standard back-propagation of error derivatives [29] with **x** as target. Depending on the nature of the input examples, $\mathcal{L}$ can either be the traditional squared error or the cross-entropy for vectors of valued in [0, 1] interpreted as probabilities:

$$\text{squared\_error}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=1}^{n} (\hat{x}_i - x_i)^2, \tag{6}$$

$$\text{cross-entropy}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=1}^{n} \left[x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i) \right]. \tag{7}$$

With a linear encoding activation function a^{E}, an AA actually emulates principal component analysis [2]. When using non-linear a^{E} (e.g. a sigmoid function), an AA can learn more complex representations and has the ability to capture multi-modal aspects of the input distribution [17].

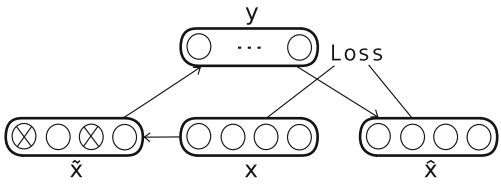

Fig. 3 The training scheme of a DAA. Input components modified by the corruption process are marked with a ⊗ cross

The denoising auto-associator (DAA) model is an AA variant that aims to remove input noise, in addition to extracting a non-linear feature space [33]. To achieve the denoising goal, the training example fed to a DAA is a corrupted version $\tilde{\mathbf{x}}$ of $\mathbf{x}$. There exist many ways of corrupting $\mathbf{x}$. The simplest corruption rule proceeds by cancelling out some uniformly selected components of $\mathbf{x}$. DAA thus involves an additional hyper-parameter compared to AA, namely the input corruption rate ν. The training scheme of a DAA is illustrated in Fig. 3.

The objective function is optimized by back-propagation, as in an AA:

$$\phi^{\mathrm{D}}(\Theta) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(\mathbf{x},\, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\tilde{\mathbf{x}})) \quad \text{with} \quad \tilde{\mathbf{x}} = \mathrm{corrupt}(\mathbf{x}). \tag{8}$$

Compared to the standard AA, the DAA learning rule forces the encoding layer to learn more robust features of the inputs, conducive to a better discriminative power. DAA learning can be thought of as manifold learning: the encoding layer maps the corrupted examples as close as possible to the manifold of the true examples.

4 Sparse Auto-Associator

The proposed sparse auto-associator (SAA) differs from standard AA only in that its learning rule is modified to favor sparse representations on the encoding (hidden) layer.

In numerical analysis and signal processing, a sparse vector is traditionally defined as a vector with small L_0-norm, i.e. involving few non-zero components. In the following, an encoder representation will be termed sparse iff it involves few active neurons, where an *active* neuron is one with activation value close to the maximum of the activation function. For instance, if $a^{\mathrm{E}} = \tanh$ then the encoding neuron n_i is considered maximally active for $\mathbf{x}$ if $y_i = 1$ and maximally inactive for $\mathbf{x}$ if $y_i = -1$.

In contrast with [12], we do not require any training example $\mathbf{x}$ to be provided with its target optimal sparse representation $\tilde{\mathbf{y}}$ to achieve sparse encoding through supervised learning. The working assumption used in the following is that the code $\mathbf{y}$, learned using standard AA to encode input $\mathbf{x}$ *constitutes* a non-sparse approximation of the desired sparse $\tilde{\mathbf{y}}$. Accordingly, one aims at recovering $\tilde{\mathbf{y}}$ by sparsifying $\mathbf{y}$, i.e.by cancelling out the least active encoding neurons (setting their activity to the minimal activation value, e.g. -1 if $a^{\mathrm{E}} = \tanh$). As summarized in Algorithm 1, for each

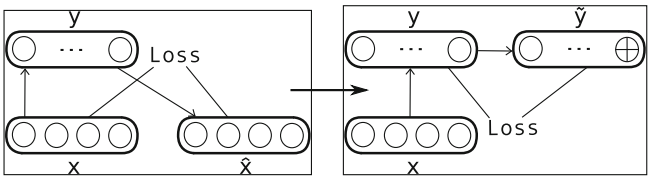

Fig. 4 The training scheme of an SAA. Code components modified by the sparsification process are marked with a ⊕ cross

input **x**, SAA alternates the standard accuracy-driven weight update, aimed at the reconstruction of **x**, and a sparsity-driven weight update applied to the encoding layer only, with **x** as input and $\tilde{\mathbf{y}}$ as target output (Fig. 4).

SAA thus iteratively and alternatively optimizes the accuracy and the sparsity of the encoder, as follows. Let us denote by $\mathbf{x^{(t)}}$ the training example fed to the AA at time t in an online learning setting and $\mathbf{y^{(t)}}$ the representation computed by the encoding layer once the AA has been trained with the previous examples $\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(t-1)}}$. By alternatively applying an accuracy optimization step followed by a sparsity optimization step for each example, it is expected that the level of sparsity of $\mathbf{y^{(t)}}$ will increase with t, until code **y** consistently yields a "sufficiently sparse" approximation of $\tilde{\mathbf{y}}$. At such a point, **y** reaches a high level of sparsity while it still enables to reconstruct **x** with high accuracy, and the encoding layer achieves a good compromise between coding sparsity and accuracy.

Algorithm 1: SAA alternate optimization learning rule.

1 **for** *several epochs (training examples presented in random order)* **do**
2 **foreach** *training example* $\mathbf{x^{(t)}}$ **do**
3 **Accuracy-driven learning rule**: perform one step back-propagation for updating Θ^{E} and Θ^{D} with $\mathbf{x^{(t)}}$ as input and $\mathbf{x^{(t)}}$ as target
4 (*see Fig. 4, left*);
5 Run a forward pass on the encoding layer of the AA with $\mathbf{x^{(t)}}$ as input to compute $\mathbf{y^{(t)}}$;
6 **Sparsify** $\mathbf{y^{(t)}}$ to obtain $\tilde{\mathbf{y}}^{(t)}$;
7 **Sparsity-driven learning rule**: perform one step back-propagation on the encoding layer only, for updating Θ^{E} with $\mathbf{x^{(t)}}$ as input and $\tilde{\mathbf{y}}^{(t)}$ as target 10mm (*see Fig. 4, right*);

The criterion for the accuracy-driven learning rule (line 3 in Algorithm 1) is the same as the AA criterion (Eq. (5)):

$$\phi^{\mathrm{S,accuracy}}(\Theta) = \phi(\Theta) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(\mathbf{x}, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\mathbf{x})), \tag{9}$$

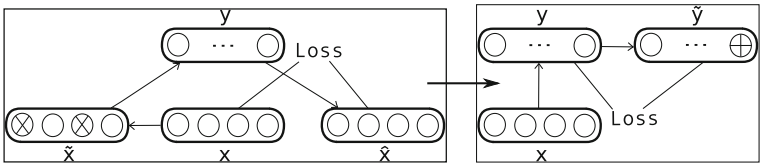

Fig. 5 The training scheme of an SDAA. Input components modified by the corruption process are marked with a ⊗ cross and code components modified by the sparsification process are marked with a ⊕ cross

whereas the criterion for the sparsity-driven learning rule (line 6) only regards the encoding parameters:

$$\phi^{\mathrm{S,sparsity}}(\Theta^{\mathrm{E}}) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(f^{\mathrm{E}}(\mathbf{x}), \tilde{\mathbf{y}}) \quad \text{with} \quad \tilde{\mathbf{y}} = \mathrm{sparsify} \circ f^{\mathrm{E}}(\mathbf{x}). \tag{10}$$

Several sparsification rules have been considered to map $\mathbf{y}$ onto $\tilde{\mathbf{y}}$ (line 5 of Algorithm 1). The first two rules are deterministic, parameterized from a fixed number of neurons, or a fixed activation threshold. Formally, the first one proceeds by cancelling out the α neurons with lowest activity, while the second one cancels out all neurons with activation less than threshold τ. For instance, for $\tau = 0$ and $a^{\mathrm{E}} = \tanh$, then $\tilde{y}_i = -1$ if y_i is negative and $\tilde{y}_i = y_i$ otherwise. A third sparsification rule proceeds stochastically, where the probability of canceling out an entry increases as its activation decreases, e.g. $P(\tilde{y}_i = -1) = (e^{1-y_i} - 1)/(e^2 - 1)$.

By construction, the deterministic size-driven sparsification rule results in a sparse coding with the same size for all examples, which might over-constrain the sparse coding solution. Indeed, many datasets involve examples with different information content: some examples may need a large number of active neurons to be well represented while some may only need a small number. The last two sparsification rules achieve a variable-size coding.

In summary, the SAA framework involves the same hyper-parameters as the standard AA (chiefly the back-propagation learning rate), plus one hyper-parameter controlling the sparsification step (line 5 of Algorithm 1), respectively α (for the size-driven sparsification rule) or τ (for the threshold-driven sparsification rule).

Note also that integrating the sparsification steps within a denoising auto-associator is straightforward, by replacing the AA update rule (line 3 of Algorithm 1) with the DAA update rule, thus yielding a hybrid model referred to as sparse DAA (SDAA). The training scheme of an SDAA is illustrated in Fig. 5, where the accuracy and sparsity criteria are derived from Eq. (8) and Eq. (10) as follows:

$$\phi^{\mathrm{SD,accuracy}}(\Theta) = \phi^{\mathrm{D}}(\Theta) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(\mathbf{x},\, f^{\mathrm{D}} \circ f^{\mathrm{E}}(\tilde{\mathbf{x}})), \tag{11}$$

Fig. 6 Images of a digit in class "8", taken from each dataset: MNIST-basic (*left*), MNIST-bg-rand (*middle*) and MNIST-bg-img (*right*)

$$\phi^{\mathrm{SD,sparsity}}(\Theta^{\mathrm{E}}) = \phi^{\mathrm{S,sparsity}}(\Theta^{\mathrm{E}}) = \sum_{\mathbf{x}\in\mathcal{S}} \mathcal{L}(f^{\mathrm{E}}(\mathbf{x}), \tilde{\mathbf{y}}). \tag{12}$$

The computational complexity of all above schemes (AA, DAA, SAA and SDAA) are $\mathcal{O}(n \times m)$ per training example, with n the input dimension and m the number of encoding neurons.

5 Experiments

This section reports on the experimental validation of the sparse auto-associator (SAA) and the sparse denoising auto-associator (SDAA), comparatively to the standard and denoising auto-associators (AA and DAA). After describing the experimental setting, the section discusses the trade-off between the size of the feature space and the predictive accuracy of the classifier built on this feature space.

5.1 Experimental Setting

All experiments have been carried out on three handwritten digit recognition, variants of the original MNIST dataset built by LeCun and Cortes.[1] These variants due to Larochelle et al. [20],[2] differ from the original MNIST by the background image (Fig. 6):

- MNIST-basic: white digits on a uniform black background;
- MNIST-bg-rand: white digits on a random grey-level background;
- MNIST-bg-img: white digits on grey-level natural images as a background.

Notably, MNIST-basic is a far more challenging learning problem than MNIST due to its reduced size (10,000 example training set; 2,000 example validation set and 50,000 example test set). Each dataset involves 10 classes, where each example is given as a (28×28) grey-level pixel image and processed as a vector in $\mathbb{R}^{784}$,

[1] Original MNIST database: http://yann.lecun.com/exdb/mnist/.

[2] MNIST variants site: http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations.

by scrolling the image in a left-to-right top-to-bottom fashion and recording each visited pixel.

Let C, X^{train}, X^{val}, X^{test}, L^{train}, L^{val} and L^{test} be defined as follows:

- $C = \{1, \ldots, c\}$ is the set of class labels, with c the number of classes ($c = 10$ for all datasets);
- X^{train} is the sequence of training examples $\mathbf{x}^{(\mathbf{k})}$, train $\in \mathbb{R}^n$;
- L^{train} is the sequence of the class label $l^{(k),\text{train}} \in C$ for each $\mathbf{x}^{(\mathbf{k}),\text{train}}$;
- X^{val} is the sequence of validation examples $\mathbf{x}^{(\mathbf{k})}$, val $\in \mathbb{R}^n$;
- L^{val} is the sequence of the respective class label $l^{(k),\text{val}} \in C$;
- X^{test} is the sequence of test examples $\mathbf{x}^{(\mathbf{k})}$, test $\in \mathbb{R}^n$;
- L^{test} is the sequence of the respective class label $l^{(k),\text{test}} \in C$.

All AA variants will be comparatively assessed on the basis of their discriminant power and their feature space dimension.

5.2 Discriminant Power

The discriminant power of an AA variant is measured as the predictive accuracy of a classifier learned from the feature space built from this AA variant, along the following procedure:

1. **Unsupervised learning:** from (X^{train}, X^{val}) train an AA, a DAA, an SAA and an SDAA, respectively denoted by $\mathcal{A}$, $\mathcal{A}^{\text{D}}$, $\mathcal{A}^{\text{S}}$ and $\mathcal{A}^{\text{SD}}$;
2. **Classifier building:** from $\mathcal{A}$, $\mathcal{A}^{\text{D}}$, $\mathcal{A}^{\text{S}}$ and $\mathcal{A}^{\text{SD}}$ respectively, initialize the two-layer neural network classifiers $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$ by removing the decoders and plugging c neurons at the output of the encoders, one for each class;
3. **Supervised fine-tuning:** from (X^{train}, X^{val}) and (L^{train}, L^{val}) train $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$;
4. **Performance measuring:** from X^{test} and L^{test} estimate the classification accuracy of $\mathcal{N}$, $\mathcal{N}^{\text{D}}$, $\mathcal{N}^{\text{S}}$ and $\mathcal{N}^{\text{SD}}$.

Figure 7 comparatively displays the auto-associator architecture $\mathcal{A}$ and the associated multi-layer classifier $\mathcal{N}$. With architecture $\mathcal{A}$, the AA is trained to build a feature space by reconstructing its inputs on its output neurons. With architecture $\mathcal{N}$, the feature space is further trained by back-propagation to yield the class associated to the AA inputs, using c output neurons (one for each class).

The unsupervised feature extraction training phase and the supervised classifier training phase (steps 1 and 3 of the above procedure) each involve 50 epochs of stochastic back-propagation over X^{train} and X^{val}, with the squared error as loss function. At each epoch the examples are presented in random order. The encoding dimension m of each AA variant has been set equal to the input dimension n, i.e. $m = 784$ neurons in the encoding layer.

The activation function is tanh. The bias are initialized to 0 while the weights are randomly initialized following the advice in [23]. The sparsification heuristics is the

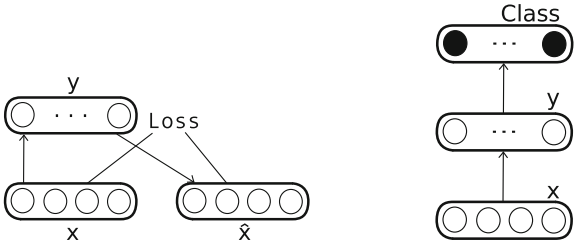

Fig. 7 From an auto-associator $\mathcal{A}$ (*left*) to the corresponding classifier $\mathcal{N}$ (*right*)

Table 1 Hyper-parameter range tested through the grid-search procedure

Name	Candidate values	Models
Learning rate η	0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5	All AA variants and all classifier variants
Input corruption rate ν	0, 0.1, 0.2, 0.4	DAA and SDAA

one with the threshold[3] (Sect. 4) where the threshold is set to 0. In other words, all features with negative values are set to -1 (the minimum value of tanh). For inputs normalized in $[-1, 1]$, the input corruption process in $\mathcal{A}^{\mathrm{D}}$ and $\mathcal{A}^{\mathrm{SD}}$ consists in setting independently each input entry to -1 with probability ν.

The values of the back-propagation learning rate η and input corruption rate ν have been selected by grid-search, where each hyper-parameter setting is assessed from 10 runs (with different weight initialization and example ordering), learning on X^{train} and evaluating the performance on X^{val}. By consistency, both training phases (steps 1 and 3) use the same η value. The candidate hyper-parameter values are reported in Table 1.

Table 2 displays the classification error rates averaged over 10 independent runs. Note that in some cases, the best ν value is 0, resulting in identical results for AA and DAA, or SAA and SDAA. On all datasets, SAA is shown to significantly[4] improve on AA in terms of predictive accuracy. The feature space built by $\mathcal{A}^{\mathrm{S}}$ thus seems to capture discriminant features better than the standard $\mathcal{A}$. It must be emphasized that both feature spaces have same size, are trained using unsupervised learning and only differ in the sparsity step. These results suggest that the SAA approach does take practical advantage of one of the more appealing theoretical assets of sparse representations, namely their tendency to favor class separability compared to dense representations.

Secondly, it is observed that DAA, SAA and SDAA do not offer any significant difference regarding their predictive accuracies. A tentative interpretation for this fact goes as follows. The denoising and sparsification heuristics respectively involved in

[3] The probabilistic sparsification heuristics has been experimented too and found to yield similar results (omitted for the sake of brevity).

[4] All statistical tests are heteroscedastic bilateral T tests. A difference is considered significant if the p-value is less than 0.001.

Table 2 Mean and standard deviation of the classification error rate when the encoding dimension is set to the input dimension

Dataset	Error rate (%)			
	AA	DAA	SAA	SDAA
MNIST-basic	4.49 (0.06)	3.87 (0.04)	3.98 (0.09)	3.98 (0.08)
MNIST-bg-rand	22.4 (0.10)	19.5 (0.09)	19.5 (0.23)	19.5 (0.23)
MNIST-bg-img	25.6 (0.37)	23.6 (0.15)	23.4 (0.57)	23.0 (0.43)

Table 3 Mean and standard deviation of the classification error rate and encoding dimension as obtained by removing useless neurons after $\mathcal{A}^{\mathrm{S}}$ or $\mathcal{A}^{\mathrm{SD}}$ training

Dataset	Reduced encoding dimension		Error rate (%)	
	SAA	SDAA	SAA	SDAA
MNIST-basic	719 (7.9)	757 (7.7)	3.96 (0.07)	4.00 (0.16)
MNIST-bg-rand	634 (9.9)	634 (9.9)	19.3 (0.29)	19.3 (0.29)
MNIST-bg-img	248 (12.0)	761 (5.0)	23.3 (0.36)	23.0 (0.43)

DAA and SAA aim at comparable properties (namely coding robustness), although through different ways. It seems that both ways are incompatible, in the sense that they cannot be combined effectively. This interpretation is confirmed as the best ν value selected for SDAA is close to 0 (0 or .1) whereas it is clearly higher for DAA (.2 or .4): SDAA in such different conditions almost coincides with DAA. In other words, the standard AA can hardly achieve at the same time a low reconstruction error, a good robustness to noise, and sparsity. Further investigation could be performed to search how the denoising and sparsity heuristics could be made more compatible through simultaneously adjusting the input corruption rate and the sparsification threshold.

5.3 Pruning the Feature Space

It is natural to investigate whether all features in the feature space are active, i.e. if there exist some encoding neurons which are never active during the unsupervised SAA training. Such features could be removed from the feature space at no cost in terms of reconstruction error, i.e. they would enable a pruning of the feature space—although there is no evidence naturally that sparsity would favor such a pruning.

The pruning of the feature space has been investigated through considering an additional pruning step on the top of learning the encoding layer of $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$. Formally, the pruning phase proceeds by removing all neurons whose activity is consistently negative over all training examples. The approach considers the same hyper-parameter values as previously selected by grid-search on $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$.

The relevance of the pruning heuristics is empirically demonstrated in Table 3. While the pruning heuristics strictly reduces the original feature space dimension

Table 4 Mean and standard deviation of the classification error rate when the encoding dimension is determined by the mean one obtained by the SAA in the previous experiment (see Table 3)

Dataset	Encoding dimension	Error rate (%)	
		AA	DAA
MNIST-basic	719	4.53 (0.19)	3.80 (0.07)
MNIST-bg-rand	634	22.0 (0.07)	19.4 (0.20)
MNIST-bg-img	248	25.9 (0.22)	25.9 (0.22)

(set to $m = n = 784$ in all experiments), it does not hinder the predictive accuracy (comparatively to Table 2). In other words, the neurons which have been removed along the pruning step did not convey discriminant information and/or could not be efficiently recruited in the supervised learning stage.

It can be observed that the efficiency of the pruning heuristics significantly varies depending on the datasets, that is, depending on the background image. A constant background (MNIST-basic, 719 features on average) entails more active features than a random one (MNIST-bg-rand, 634 features on average) and considerably more than a non-random one (MNIST-bg-img, 248 features on average). A tentative interpretation for this result is that, the more informative the background, the larger the number of neurons trained to reconstruct the background patterns, and consequently the smaller the number of neurons trained to reconstruct the digit patterns. The variability of the background patterns, higher than those of the digit patterns, might explain why background-related neurons are more often silenced than the others. Further investigation is required to confirm or infirm this conjecture, analyzing the internal state of the $\mathcal{A}^{\mathrm{S}}$ and $\mathcal{A}^{\mathrm{SD}}$ architectures.

Interestingly, the pruning heuristics is ineffective in the SDAA case, where the feature space dimension tends to be constant. This observation both confirms that the sparsity and the denoising heuristics hardly cooperate together, as discussed in the previous subsection. It also suggests that the sparsity heuristics is more flexible to take advantage of the input data structure.

Notably, the pruning heuristics can be seen as a particular case of the *common sparsity profile* constraint [27] involved in the dictionary learning field (Sect. 1), penalizing the use of atoms which are seldom used.

For the sake of completeness, let us investigate how AA or DAA would withstand the reduction of the feature space dimension as yielded by the pruning heuristics above. The comparative assessment proceeds by setting the size of the coding layer to the reduced $m^\star$ obtained as above. The hyper-parameters are set by grid-search, on the same candidate values (Table 1).

As shown in Table 4, the feature space reduction adversely affects the AA and DAA accuracy, particularly so in the case of the MNIST-bg-img dataset. The fact that AA and DAA yield the same accuracy is due to the fact that $\nu = 0$. Indeed, the case of an information-rich background makes it more difficult for DAA to achieve a low reconstruction error and a high predictive accuracy, all the more so as the noise rate is high. Overall, it is suggested that the DAA strategy is more demanding than the SAA

one in terms of the size of the feature space, as the former comprehensively aims at coping with all encountered noise patterns in the example, whereas the latter follows a destructive strategy ignoring all weak patterns in the features. These antagonistic goals might further explain why the two strategies can hardly be combined.

6 Discussion

As mentioned in Sect. 1, the main contribution of this chapter has been to present (Sects. 4 and 5) a sparsity-driven procedure to enforce the learning of sparse feature space in the auto-associator framework. The use of this procedure within the standard stochastic gradient AA learning procedure, referred to as sparse auto-association, promotes codes which are both *accurate* in terms of representation, and *sparse* in terms of the low number of encoding neurons activated by an input example on average. A primary merit of SAA over the more popular sparse dictionary learning is to yield a *non-linear* code. A second merit is that this code is explicit and can be computed for any new example with linear complexity (that is, a feedforward pass to compute the features, i.e. the hidden layer states), whereas it requires solving an optimization problem in the dictionary case.

After summarizing the main benefits of SAA w.r.t. its accuracy for classification purposes and its ability for sparse coding, the method will be viewed in light of the four directions of research briefly presented in Sect. 2.2.

6.1 Benefits of the Sparse Auto-Associator

Accuracy-wise, SAA yields similar results as the denoising auto-associator on three well-studied MNIST variants, involving different digit backgrounds (constant, uniform, or image-based). Both SAA and DAA significantly improve on AA, which is explained from the regularization effect of respectively the sparsity- and denoising-driven procedures. Both procedures implicitly take advantage of the fact that the data live in a low-dimensional feature space. Uncovering this low-dimensional space enforces the description robustness.

Interestingly, combining the sparsity- and denoising-driven procedures does not help: experimentally, SDAA outperforms neither SAA nor DAA. Our tentative interpretation for this fact, the incompatibility of both procedures, is that the denoising-driven procedure aims at getting rid of weak perturbation patterns on the input layer, whereas the sparsity-driven procedure aims at getting rid of weak patterns in the encoding layer.

Coding-wise, it seems that SAA can be made more efficient than DAA as far as the size of the feature space is concerned. The pruning heuristics, removing all features which are never activated by the training examples above a given threshold, yields a significant reduction of the feature space dimension *with no accuracy loss*,

even as it was implemented with a naive activity threshold.[5] The use of this pruning heuristics enables SAA to autonomously adjust the size of the feature space, starting with a large number of features and pruning the inactive ones after the AA training, thus yielding a sparse and *compressed* representation.

Admittedly, the effectiveness of the pruning rule depends on the dataset: experimentally, it is best when the variability of the input patterns is neither too high (random background) nor too low (constant background). Interestingly, the most realistic cases (image background) are the most favorable cases, since SAA enables a 3-fold reduction of the feature space size, actually uncovering the common sparsity profile of the examples.

A main merit of the pruning heuristics is to yield an appropriate feature space dimension for a given dataset. A second one, empirically demonstrated, is that it does preserve the discriminant power of the feature space. Note, however, that directly considering the same feature space dimension with AA or DAA significantly degrades the predictive accuracy.

6.2 Comparison with Related Work

Let us discuss the strengths and weaknesses of the SAA framework compared to the sparse coding neural-based methods, presented in Sect. 2.2. It must first be emphasized that all sparse feature learning methods have a quite similar computational cost that hardly scale up for *big data*, although recent achievements show that impressive results can be obtained when increasing the size of the dataset by several orders of magnitude [8].

On the positive side, SAA does not require any prior knowledge about the problem domain and the target optimal code, as opposed to [11] and [12]. SAA actually uncovers the common sparsity profile of the examples, although it does not explicitly consider any selectivity property, as opposed to [11]. It accommodates online learning, also as opposed to [11], through a simple stochastic gradient approach, one which does not require any indicators about the neurons to be maintained, as opposed to the average past activity used in [24] and [14]. Importantly, the straightforward sparsity-driven procedure makes SAA easy to understand and implement, without requiring any particular trick to make the standard back-propagation procedure to work, as opposed to [10].

On the negative side, SAA needs to be given theoretical foundations, or could be related to the theory of biological neuron computation as e.g. [10] or [11]. In particular, further work will investigate how the sparsity-driven procedure can be analyzed in terms of ill-posed optimization resolution.

[5] Complementary experiments, varying the pruning threshold in a range around 0, yield same performance (results omitted for brevity).

7 Conclusion and Perspectives

Focused on sparse and low-dimensional feature coding, this chapter has presented the new sparse auto-associator framework. The main motivation for this framework is rooted in information theory, establishing the robustness of sparse code w.r.t. transmission noise. In the field of machine learning, sparse coding further facilitates the separability of examples, which has made sparse coding a hot topic for the last decade [6, 9, 27]. These properties of sparse representations have been experimentally confirmed in terms of predictive accuracy compared to the standard auto-associator.

SAA should be viewed as an alternative to sparse dictionary learning [26] in several respects. On the one hand, it provides a *non-linear* feature space, more easily able to capture complex data structures than linear coding. On the other hand, while both SAA and dictionary-based codings are rather costly to be learned, the SAA coding is explicit and computationally cheap in the generalisation phase: the coding of a further example is computable by feed-forward propagation, whereas it results from solving an optimization problem in the dictionary framework.

The SAA approach offers several perspectives for further research. A primary perspective, inspired from the field of deep learning [3, 19], is to stack SAA in a layer-wise manner, expectedly yielding gradually more complex and abstract non-linear features. Preliminary experiments show that the direct stacking of an SAA however is ineffective as the reconstruction of an already sparse coding derives a degenerated optimization problem. Further work will be concerned with controlling and gradually adapting the sparsity level along the consecutive layers, using an appropriate sparsity criterion and schedule. Another possibility is to alternate SAA layers, with subsequent pruning, and AA layers, without sparse coding, in layer-wise building a deep architecture, thus taking inspiration from the alternative stacking of convolutional and pooling layers promoted by LeCun et al. [18, 21].

Another promising perspective is to see SAA in the light of the *dropout* procedure recently proposed by Hinton et al. [16], where some features are randomly omitted during the back-propagation step. The dropout heuristics assuredly resulted in dramatic performance improvements on hard supervised learning problems, a result attributed to the fact that this random perturbation breaks the spurious coalitions of features, each covering for the errors of others. Indeed the sparsity-driven procedure, especially in its stochastic form (Sect. 4) can be viewed as a dropout rule biased to low-activity neurons. Extensive further experiments are required to see how the sparsity-driven procedure can be best combined with the many other ingredients involved in the dropout-based deep neural network architecture.

Acknowledgments This work was supported by ANR (the French National Research Agency) as part of the ASAP project under grant ANR_09_EMER_001_04.

References

1. M. Aharon, M. Elad, A. Bruckstein, K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. IEEE Trans. Signal Process. **54**, 4311–4322 (2006)
2. P. Baldi, K. Hornik, Neural networks and principal component analysis: learning from examples without local minima. Neural Networks **2**, 53–58 (1989)
3. Y. Bengio, Learning deep architectures for AI. Found. Trends Mach. Learn. **2**, 1–127 (2009)
4. Y. Bengio, P. Lamblin, V. Popovici, H. Larochelle, in *Neural Information Processing Systems (NIPS)*. Greedy Layer-wise Training of Deep Networks (2007), pp. 1–8
5. H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition. Biol. Cybern. **59**, 291–294 (1988)
6. E.J. Candès, The restricted isometry property and its implications for compressed sensing. Comptes Rendus de l'Académie des Sci. **346**, 589–592 (2008)
7. S.S. Chen, D.L. Donoho, M.A. Saunders, Atomic decomposition by basis pursuit. SIAM J. Sci. Comput. **20**, 33–61 (1998)
8. A. Coates, A.Y. Karpathy, A. ans Ng, in *Neural Information Processing Systems (NIPS)*. Emergence of Object-Selective Features in Unsupervised Feature Learning (2012)
9. D.L. Donoho, M. Elad, Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ^1 minimization. Proc. Nat. Acad. Sci. U.S.A. **100**, 2197–2202 (2003)
10. X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks. in *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2011), pp. 315–323
11. H. Goh, N. Thome, M. Cord, Biasing restricted Boltzmann machines to manipulate latent selectivity and sparsity. in *Neural Information Processing Systems (NIPS): Workshop on Deep Learning and Unsupervised Feature, Learning* (2010), pp. 1–8
12. K. Gregor, Y. LeCun, Learning fast approximations of sparse coding. in *International Conference on Machine Learning (ICML)* (2010), pp. 399–406
13. G.E. Hinton, Connectionist learning procedures. Artif. Intell. **40**, 185–234 (1989)
14. G.E. Hinton, A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010–003, University of Toronto (2010)
15. G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets. Neural Comput. **18**, 1527–1554 (2006)
16. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. in *Neural information processing systems (NIPS)* (2012) arxiv.org/abs/1207.0580v1 [cs.NE] 3 July 2012
17. N. Japkowicz, S.J. Hanson, M.A. Gluck, Nonlinear autoassociation is not equivalent to PCA. Neural Comput. **12**, 531–545 (2000)
18. K. Kavukcuoglu, M.A. Ranzato, Y. LeCun, Fast inference in sparse coding algorithms with applications to object recognition. in *Neural Information Processing Systems (NIPS): Workshop on Optimization for Machine Learning* (2008) arXiv:1010.3467v1 [cs.CV] 18 Oct 2010
19. H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks. J. Mach. Learn. Res. **10**, 1–40 (2009)
20. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International conference on machine learning (ICML)* (2007), pp. 473–480
21. Y. LeCun, Learning invariant feature hierarchies. in *European Conference in Computer Vision (ECCV). Lecture Notes in Computer Science*, vol. 7583. (Springer, New York, 2012), pp. 496–505
22. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
23. Y. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop. in *Neural Networks: Tricks of the Trade* (1998). pp. 9–50
24. H. Lee, C. Ekanadham, A.Y. Ng, Sparse deep belief net model for visual area V2. in *Neural Information Processing Systems (NIPS)* (2007), pp. 873–880

25. H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. in *International Conference on Machine Learning (ICML)* (2009), p. 77
26. J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online dictionary learning for sparse coding. in *International Conference on Machine Learning (ICML)* (2009), pp. 689–696
27. A. Rakotomamonjy, Surveying and comparing simultaneous sparse approximation (or group-LASSO) algorithms. Signal Process. **91**, 1505–1526 (2011)
28. M.A. Ranzato, F.-J. Huang, Y.-L. Boureau, Y. LeCun, Unsupervised learning of invariant feature hierarchies with applications to object recognition. in *Computer Vision and Pattern Recognition (CVPR)* (2007), pp. 1–8
29. D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
30. J.A. Tropp, Greed is good: algorithmic results for sparse approximation. IEEE Trans. Inf. Theory **50**, 2231–2242 (2004)
31. P.E. Utgoff, D.J. Stracuzzi, Many-layered learning. Neural Comput. **14**, 2497–2539 (2002)
32. V.N. Vapnik, *Statistical Learning Theory* (Wiley, New York, 1998)
33. P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders. in *International Conference on Machine Learning (ICML)* (2008), pp. 1096–1103

Chapter 5
HyperNEAT: The First Five Years

David B. D'Ambrosio, Jason Gauci and Kenneth O. Stanley

Abstract HyperNEAT, which stands for *Hypercube-based NeuroEvolution of Augmenting Topologies*, is a method for evolving indirectly-encoded artificial neural networks (ANNs) that was first introduced in 2007. By exploiting a unique indirect encoding called *Compositional Pattern Producing Networks* (CPPNs) that does not require a typical developmental stage, HyperNEAT introduced several novel capabilities to the field of neuroevolution (i.e. evolving artificial neural networks). Among these, (1) large ANNs can be compactly encoded by small genomes, (2) the size and resolution of evolved ANNs can scale up or down even after training is completed, and (3) neural structure can be evolved to exploit problem geometry. Five years after its introduction, researchers have leveraged these capabilities to produce a broad range of successful experiments and extensions that highlight the potential for future research to build further on the ideas introduced by HyperNEAT. This chapter reviews these first 5 years of research that builds upon this approach, and culminates with thoughts on promising future directions.

1 Introduction

HyperNEAT lies at the intersection of two research areas. One of these, *neuroevolution*, aims to harness the power of evolutionary computation to evolve artificial neural networks (ANNs) [33, 39, 82, 96]. The other area, called *generative and developmental systems* (GDS), studies how compact encodings can describe large,

D. B. D'Ambrosio (✉) · J. Gauci · K. O. Stanley
University of Central Florida, 4000 Central Florida Blvd. Orlando, Orlando, FL 32816, USA
e-mail: ddambro@eecs.ucf.edu

J. Gauci
e-mail: jgauci@eecs.ucf.edu

K. O. Stanley
e-mail: kstanley@eecs.ucf.edu

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_5,

complex structures for the purpose of evolution [11, 13, 50, 83]. Such encodings are sometimes called *indirect encodings* because each gene in the encoding does not map to a single corresponding unit of structure in the phenotype. The hope in both areas is that evolved artifacts will someday approach the complexity and power of the products of evolution in nature. At their intersection is the idea that indirect encoding might aid the evolution of ANNs by leveraging the properties of development.

Before 2006, most indirect encodings worked by triggering a process of *development* that begins with a small embryonic structure that ultimately grows into the final phenotypic form [13, 32, 50, 60, 64, 74, 83]. The connection between development on the one hand and indirect encoding on the other is intuitive because natural DNA itself maps to the human or animal phenotype through a process of development that begins with the egg. However, in 2006 Stanley introduced a new kind of indirect encoding called a *Compositional Pattern Producing Network* (CPPN) that does not require an explicit developmental process to generate patterns [75, 76]. Instead it encodes patterns through *function composition*, in effect building spatial patterns out of the composition of groups of simple functions. Many interesting spatial patterns (i.e. pictures) were evolved with CPPNs, not least of which were bred by users of the Picbreeder online service [72, 73]. These patterns exhibit intriguing regularities reminiscent of patterns seen in nature, such as symmetry and repetition with variation.

CPPNs were also well-suited to evolving with the existing NeuroEvolution of Augmenting Topologies (NEAT) algorithm [82] because CPPNs are networks similar to the ANNs traditionally evolved by NEAT. Thus it was clear that NEAT could effectively evolve CPPNs that produce interesting patterns in space. However, at the time that CPPNs were introduced in 2006, the big question occupying our research group was whether CPPNs could encode *connectivity patterns* rather than just spatial patterns. If they could, then all the promising geometric properties of the patterns generated by CPPNs could also manifest in ANNs.

The question of how to interpret the output of a CPPN as an ANN was perplexing because there was no obvious way that two-dimensional patterns in space should describe a set of neural connections. HyperNEAT was the answer to this challenge [26, 35, 79]. The key realization was that a *connectivity pattern* is isomorphic to a spatial pattern in a higher-dimensional space. For example, four-dimensional spatial patterns can be mapped easily to two-dimensional connectivity patterns. Or, more generally, a spatial pattern in $2n$ dimensions can be viewed as a n-dimensional connectivity pattern, where the two sets of n coordinates each represent the two endpoints of the connection. In effect the connectivity pattern is encoded by a scalar field in $2n$ dimensions. Such a mapping also preserves all the regularities in the original four-dimensional pattern; that is, if the four-dimensional spatial pattern exhibits a regularity, so will the two-dimensional connectivity pattern. Therefore, the solution to the problem of encoding ANNs with CPPNs is actually simple: The CPPN need only generate patterns in four dimensions, which means nothing more than adding two more inputs.

In effect, the CPPN paints a pattern inside a four-dimensional *hypercube* that is interpreted as a two-dimensional connectivity pattern. (This principle works between

six and three dimensions as well, or more generally, between $2n$ and n dimensions.) This insight is why the method came to be called "HyperNEAT," which stands for *Hypercube-based NEAT*.

The ability to interpret high-dimensional spatial patterns as networks opened up a range of novel capabilities for neuroevolution. For example, (1) networks encoded by CPPNs can be much larger than the CPPNs themselves. In some early experiments, CPPNs with only dozens of connections describe ANNs with millions [79]. Another highly unusual capability is that (2) the size or resolution of an evolved ANN can be increased *after evolution* yet still work by re-querying the CPPN to regenerate the ANN at the higher resolution, yielding a new kind of post-training scalability. Also intriguing is that (3) because CPPNs generate ANNs as *functions* of the geometry of their neurons, HyperNEAT in effect *sees* the geometry of the problem domain and can learn from that geometry. In other words, HyperNEAT can correlate left-sensors to left-effectors because it knows they occupy the same side of the network, providing a powerful new kind of domain knowledge. Thus, just as the neurons in the visual cortex are arranged in a retinotopic pattern reflecting the positions of photoreceptors in the retina, HyperNEAT can create neural structures that exploit and reflect the geometry of their sensors and effectors.

Over the last 5 years, these new capabilities (as well as the ability to indirectly encode ANNs without an explicit developmental stage) have led to a fertile new research direction in neuroevolution through indirect encoding. From novel applications to extensions and enhancements, HyperNEAT has grown during this time into a more mature method applied and extended by a diversity of researchers. While its performance in a number of domains is notable, perhaps more important is that it may serve as a stepping stone to further innovations in the field of neuroevolution. In the hope of inspiring such continued innovation, this chapter reviews HyperNEAT and much of the research on it since its inception.

2 Background

This section provides context for HyperNEAT by reviewing some of the key concepts and approaches that preceded it.

2.1 Generative and Developmental Systems

A key similarity among many neuroevolution methods, including the NEAT method that preceded HyperNEAT, is that they employ a *direct encoding*, that is, each part of the solution's representation maps to a single piece of structure in the final solution [33, 39, 82, 96]. Yet direct encodings impose the significant disadvantage that when the solution contains repeated or similar parts, those parts must be encoded separately, and therefore discovered separately. In contrast, in biological genetic encoding the mapping between genotype and phenotype is *indirect*, which means that the

phenotype typically contains orders of magnitude more structural components than the genotype contains genes. For example, a human genome of about 30,000 genes (about three billion amino acids) encodes a human brain with 100 trillion connections [28, 29, 53]. Thus the only way to discover structures with trillions of parts may be through a mapping between genotype and phenotype that translates few dimensions into many, i.e. through an indirect encoding. Because phenotypic structures often occur in repeating patterns, each time a pattern repeats, the same gene group can provide the specification. The numerous left/right symmetries of vertebrates [65, p. 302–303], the receptive fields in the visual cortex [38, 51] and fingers and toes are examples of repeating patterns in biology.

Inspired by such compression and regularity, HyperNEAT is among a new class of methods that exploit the power of *indirect encoding*. In such an encoding, the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself. Indirect encodings are often motivated by *development* in biology, in which the genotype maps to the phenotype indirectly through a process of growth [11, 60, 83]. They are powerful because they allow solutions to be represented as a pattern of policy parameters, rather than requiring each parameter to be represented individually. This capability is the focus of the field called *generative and developmental systems* [11, 13, 32, 50, 60, 64, 74, 76, 83]. The remainder of this section reviews the NEAT method, and then explains the indirect encoding called *compositional pattern producing networks* that is well-suited to NEAT and that ultimately became the basis for HyperNEAT.

2.2 Neuroevolution of Augmenting Topologies

The NEAT method was first introduced over 5 years before HyperNEAT to evolve ANNs to solve difficult control and sequential decision tasks through a direct encoding [78, 82, 84]. The basic principles of NEAT, reviewed in this section, are preserved even as they are extended to work with the indirect encoding in HyperNEAT.

Traditionally, ANNs evolved by NEAT control agents that select actions based on their sensory inputs. NEAT is unlike many previous methods that evolved neural networks, i.e. *neuroevolution* methods, which historically evolved either fixed-topology networks [40, 71], or arbitrary random-topology networks [3, 42, 96]. Instead, NEAT begins evolution with a population of small, simple networks and increases the complexity of the network topology into diverse species over generations, leading to increasingly sophisticated behavior (Fig. 1). A similar process of gradually adding new genes has been confirmed in natural evolution [63, 91] and shown to improve adaptation in a few prior evolutionary [2] and neuroevolutionary [44] approaches. However, a key feature that distinguishes NEAT from prior work in evolving increasingly complex structures is its unique approach to maintaining a healthy diversity of structures of different complexity simultaneously, as this section reviews. This approach has proven effective in a wide variety of domains [1, 45, 46, 48, 49, 56,

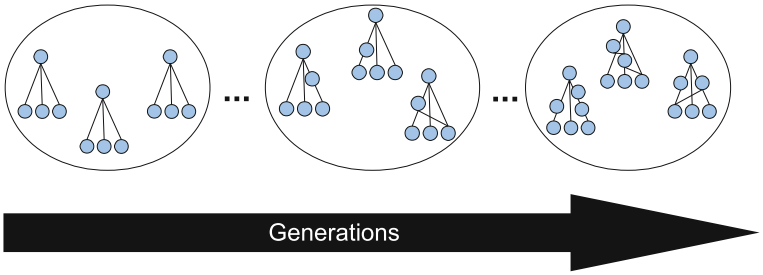

Fig. 1 Complexification in NEAT. The NEAT algorithm begins with simple ANNs with randomly generated weights. Over evolution, complexity is gradually added to the networks, allowing for increasingly diverse and sophisticated behaviors

70, 73, 77, 80, 81, 84–87]. Complete descriptions of the NEAT method, including experiments confirming the contributions of its components, are available in Stanley and Miikkulainen [82, 84] and Stanley et al. [78].

The NEAT method is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which in a population of diverse structures, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover among diverse topologies without the need for expensive topological analysis.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. Because adding new structure is often initially disadvantageous, this separation means that unique topological innovations are protected and therefore have the opportunity to optimize their structure without direct competition from other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong.

Third, many approaches that evolve network topologies and weights begin evolution with a population of random topologies [42, 96]. In contrast, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. Because of speciation, novel topologies gradually accumulate over evolution, thereby allowing diverse and complex phenotype topologies to be represented. No limit is placed on the size to which topologies can grow. New nodes and connections are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In effect, then, NEAT searches for a compact, appropriate topology by incrementally adding complexity to existing structure.

The next section reviews compositional pattern producing networks, which combine later with NEAT to make the HyperNEAT method.

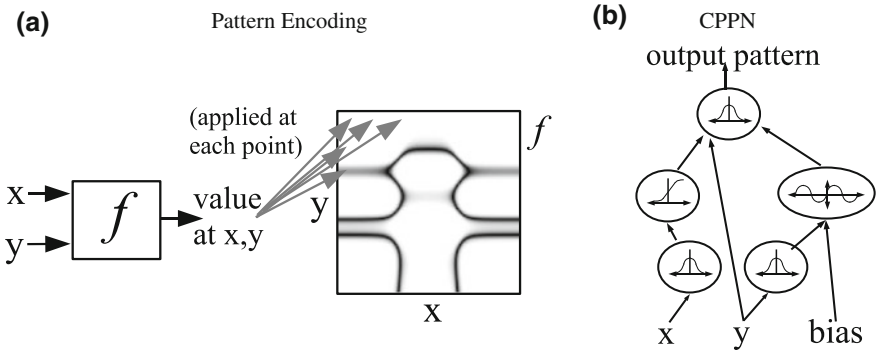

Fig. 2 CPPN Encoding. **a** The CPPN takes arguments x and y, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of the CPPN, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is the CPPN. **b** Internally, the CPPN is a graph that determines which functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. The CPPN in (**b**) actually produces the pattern in (**a**)

2.3 Compositional Pattern Producing Networks

Before CPPNs and HyperNEAT, indirect encodings ranged from low-level cell chemistry simulations to high-level grammatical rewrite systems [83]. CPPNs introduced a novel abstraction of development (unlike these prior encodings) that can represent sophisticated repeating patterns in Cartesian space [75, 76]. Unlike most generative and developmental encodings, CPPNs do not require an explicit simulation of growth or local interaction, yet still realize their essential functions. CPPNs also happen to be represented as networks (similarly to ANNs), which makes them particularly amenable to evolution through NEAT. This section reviews CPPNs, which are augmented in the HyperNEAT method (Sect. 3) to represent the connectivity patterns of ANNs.

To understand CPPNs, it helps to think of a phenotype as a pattern in space. This pattern could be anything from a body morphology to a two-dimensional image. This view of the phenotype as a spatial pattern is useful because it can then be considered as a function of n dimensions, where n is the number of dimensions in physical space. For each coordinate in that space, its level of expression is then an output of the function that encodes the phenotype. Figure 2a shows how a two-dimensional phenotype can be generated by a function of two parameters (x and y).

Stanley [75, 76] showed how simple canonical functions can be composed to create an overall network that produces complex regularities and symmetries. Each component function creates a novel geometric *coordinate frame* within which other functions can reside. The main idea is that these simple canonical functions are abstractions of specific events in development such as establishing bilateral symmetry (e.g. with a symmetric function such as Gaussian) or the division of the body into

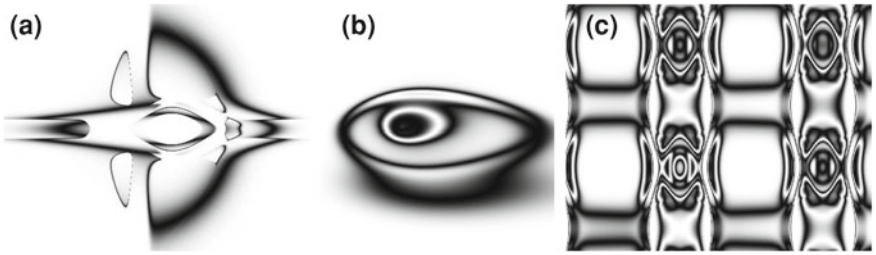

Fig. 3 CPPN-generated Regularities. Spatial patterns exhibiting **a** bilateral symmetry, **b** imperfect symmetry, and **c** repetition with variation (notice the nexus of each repeated motif) are depicted. These patterns demonstrate that CPPNs effectively encode fundamental regularities of several different types

discrete segments (e.g. with a periodic function such as sine). Figure 2b shows how such a composition can be represented by a network.

Such networks are called *compositional pattern producing networks* because they produce spatial patterns by composing basic functions. Unlike ANNs, which often contain only sigmoid functions (or sometimes Gaussian functions), CPPNs can include both types of functions and many others. Furthermore,the term *artificial neural network* would be misleading in the context of this research because ANNs were so named to establish a metaphor with a different biological phenomenon, i.e. the brain. The terminology should avoid making the implication that biological, thinking brains are in effect the same as developing embryos or genetic encodings. In this chapter, because CPPNs are ultimately used to encode ANNs in HyperNEAT, it is especially important to differentiate these concepts.

Through interactive evolution, Stanley [75, 76] demonstrated that CPPNs can produce spatial patterns with important geometric motifs that are expected from generative and developmental encodings and seen in nature. Among the most important such motifs are symmetry (e.g. left-right symmetries in vertebrates), imperfect symmetry (e.g. right-hand-ed-ness), repetition (e.g. receptive fields in the cortex [98]), and repetition with variation (e.g. cortical columns [41]). Figure 3 shows examples of several such important motifs produced through interactive evolution of CPPNs.

The choice of available activation functions for a CPPN is guided by the desired motifs. A sigmoid function is typically included in order to support non-linearities and irregularities. A symmetric function, such as a Gaussian or absolute value can be used to induce symmetry. Periodic functions such as sine or cosine are included to facilitate repetition. These types of functions make up the canonical set of available CPPN functions, but other functions can be added to the set depending on the problem being solved.

It is fortuitous that CPPNs and ANNs are so similar from a structural perspective because it means that methods designed to evolve ANNs can also evolve CPPNs. In particular, the NEAT method is a good choice for evolving CPPNs because NEAT increases the complexity of evolving networks over generations,

allowing increasingly elaborate regularities to accumulate. The next section describes how HyperNEAT combines CPPNs and NEAT to evolve ANNs with geometric regularities.

3 Method: Hypercube-Based Neuroevolution of Augmenting Topologies

After CPPNs were first introduced [75, 76], an important challenge remained to be addressed before HyperNEAT could be realized. It was clear at the time that CPPNs could encode promising regularities in *spatial patterns* (such as in Fig. 3), and that if such regularities could also be realized in the *connectivity patterns* of neural networks, a powerful neuroevolution method might result. However, what was not known was how to best interpret the output of CPPNs to effectively encode connectivity patterns. That is, the two-dimensional patterns produced in Fig. 3 present a challenge: How can such spatial patterns describe connectivity? The advantage of HyperNEAT is that it solved the problem of how to map such spatial patterns generated by CPPNs to connectivity patterns while simultaneously disentangling task structure from network dimensionality.

3.1 Mapping Spatial Patterns to Connectivity Patterns

It turns out that there is an effective mapping between spatial and connectivity patterns that can elegantly exploit geometry. The main idea is to input into the CPPN the coordinates of the *two points* that define a connection rather than inputting only the position of a single point as in Sect. 2.3. The output is then interpreted as the *weight* of the connection rather than as the intensity of a point. This way, connections can be defined in terms of the locations that they connect, thereby taking into account the network's geometry.

The CPPN in effect computes a four-dimensional function $CPPN(x_1, y_1, x_2, y_2) = w$, where the first node is at (x_1, y_1) and the second node is at (x_2, y_2). This formalism returns a weight for every connection between every potential node in the network, including recurrent connections. By convention in the original HyperNEAT, a connection is not expressed if the magnitude of its weight, which may be positive or negative, is below a minimal threshold w_{min}. The magnitude of weights above this threshold are scaled to be between zero and a maximum magnitude in the substrate. That way, the pattern produced by the CPPN can represent any network topology (Fig. 4).

For example, consider a 5×5 grid of nodes. The nodes are assigned coordinates corresponding to their positions within the grid (labeled *substrate* in Fig. 4), where $(0, 0)$ is the center of the grid. Assuming that these nodes and their positions are given

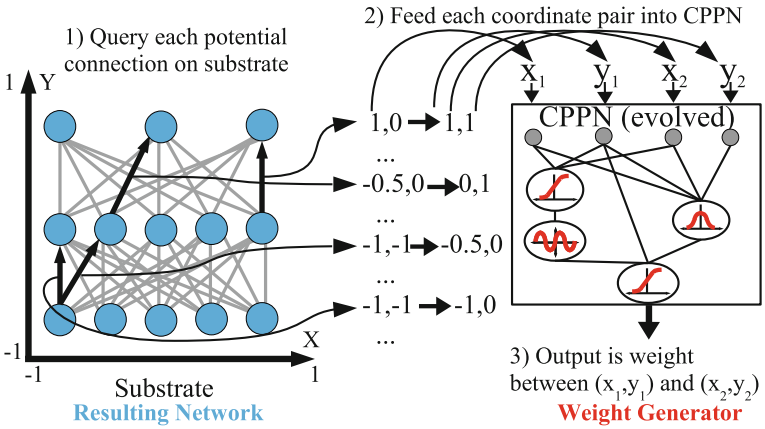

Fig. 4 Hypercube-based Geometric Connectivity Pattern Interpretation. A collection of nodes, called the *substrate*, is assigned coordinates that range from −1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) Internally, the CPPN (which is evolved by NEAT) is a graph that determines which activation functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. In this way, *connective CPPNs* produce regular patterns of connections in space

a priori, a connectivity pattern among nodes in two-dimensional space is produced by a CPPN that takes any two coordinates (source and target) as input, and outputs the weight of their connection. The CPPN is queried in this way for every potential connection on the grid. Because the connection weights are thereby a function of the *positions* of their source and target nodes, the distribution of weights on connections throughout the grid will exhibit a pattern that is a function of the geometry of the coordinate system.

The connectivity pattern produced by a CPPN in this way is called the *substrate* so that it can be verbally distinguished from the CPPN itself, which has its own internal topology. Furthermore, CPPNs that are interpreted to produce connectivity patterns can be called *connective CPPNs* to distinguish them from CPPNs that generate spatial patterns, which are called *spatial CPPNs*. HyperNEAT means neural substrates produced by connective CPPNs.

Because the connective CPPN is a function of four dimensions, the two-dimensional connectivity pattern expressed by the CPPN is isomorphic to a spatial pattern embedded in a four-dimensional hypercube (which is the origin of the *Hyper* part of *HyperNEAT*). This observation is important because it means that spatial patterns with symmetries and regularities correspond to connectivity patterns with related regularities. Thus, because CPPNs generate regular spatial patterns (Sect. 2.3), by extension they can be expected to produce connectivity patterns with corresponding regularities. The next section demonstrates this capability.

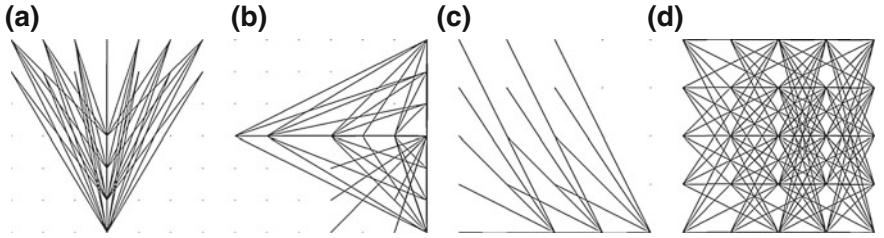

Fig. 5 Connectivity Patterns Produced by Connective CPPNs. These patterns, produced through interactive evolution, exhibit several important connectivity motifs: **a** bilateral symmetry, **b** imperfect symmetry, **c** repetition, and **d** repetition with variation. That these fundamental motifs are compactly represented and easily produced suggests the power of this encoding

3.2 Producing Regular Connectivity Patterns

Simple, easily discovered substructures in the connective CPPN produce important connective regularities in the substrate. The key difference between connectivity patterns and spatial patterns is that each discrete unit in a connectivity pattern has *two* x values and *two* y values. Thus, for example, symmetry along x can be discovered simply by applying a symmetric function (e.g. Gaussian) to x_1 or x_2 (Fig. 5a).

Imperfect symmetry is another important structural motif in biological brains. Connective CPPNs can produce imperfect symmetry by composing *both* symmetric functions of one axis along with an asymmetric coordinate frame such as the axis itself. In this way, the CPPN produces varying degrees of imperfect symmetry (Fig. 5b).

Similarly important is repetition, particularly repetition with variation. Just as symmetric functions produce symmetry, periodic functions such as sine produce repetition (Fig. 5c). Patterns with variation are produced by composing a periodic function with a coordinate frame that does not repeat, such as the axis itself (Fig. 5d). Repetitive patterns can also be produced in connectivity as functions of invariant properties between two nodes, such as distance along one axis. Thus, symmetry, imperfect symmetry, repetition, and repetition with variation, key structural motifs in all biological brains, are compactly represented and therefore easily discovered by CPPNs. The capability to produce such regularities easily is a key motivation for applying HyperNEAT to evolving neural structures.

3.3 Substrate Configuration

The layout of the nodes that the CPPN connects in the substrate can take forms other than the planar grid (Fig. 6a) discussed thus far. Different such *substrate configurations* are likely suited to different kinds of problems.

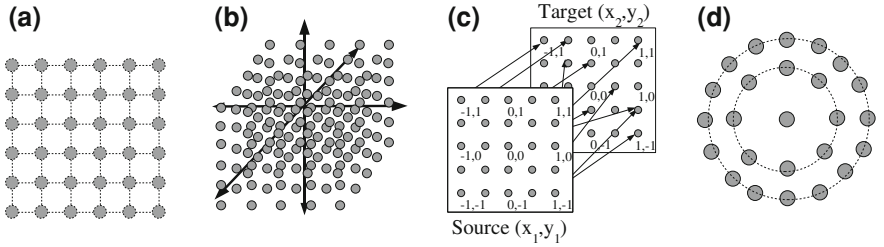

Fig. 6 Alternative Substrate Configurations. This figure shows **a** a two-dimensional grid, **b** a three-dimensional configuration of nodes centered at (0, 0, 0), **c** a state-space projection configuration in which a source sheet of neurons connects directly to a target sheet, and **d** a radial configuration. Different configurations are likely suited to problems with different geometric properties

For example, CPPNs can also produce *three-dimensional* connectivity patterns, as shown in Fig. 6b, by representing spatial patterns in the *six-dimensional* hypercube $CPPN(x_1, y_1, z_1, x_2, y_2, z_2)$. This formalism is interesting because the topologies of biological brains, including the human brain, theoretically exist within this search space.

It is also possible to restrict substrate configurations to particular structural motifs to learn about their viability in isolation. One example is a single two-dimensional sheet of neurons that connects to another two-dimensional sheet that acts as a *state-space projection* [16, 92]. The projection is a restricted three-dimensional structure in which one layer can send connections only in one direction to one other layer. Thus, because of this restriction, it can be expressed by the single four-dimensional $CPPN(x_1, y_1, x_2, y_2)$, where (x_2, y_2) is interpreted as a location on the *target* sheet rather than as being on the same plane as the source coordinate (x_1,y_1). In this way, CPPNs can search for useful patterns within state-space projection substrates (Fig. 6c).

Finally, the nodes need not be distributed in a grid. For example, nodes within a substrate that controls a radial entity such as a starfish might be best laid out with radial geometry, as shown in Fig. 6d, so that the connectivity pattern can be situated with perfect polar coordinates. Many such alternate configurations have been explored since HyperNEAT's introduction [36, 79, 88].

3.4 Input and Output Placement

Part of substrate configuration is determining which nodes are inputs and which are outputs. The flexibility to assign inputs and outputs to specific coordinates in the substrate creates an opportunity to exploit geometric relationships advantageously.

In many ANN applications, the inputs are drawn from a set of sensors that exist in a geometric arrangement in space. Unlike traditional ANN learning algorithms

that are not aware of such geometry, connective CPPN substrates *are* aware of their inputs' and outputs' geometry, and thus can use this information to their advantage.

By arranging inputs and outputs in a sensible configuration on the substrate, regularities in the geometry can be exploited by the encoding. There is room to be creative and try different configurations with different geometric advantages.

Biological neural networks rely on exploiting such regularities for many of their functions. For example, neurons in the visual cortex are arranged in the same retinotopic two-dimensional pattern as photoreceptors in the retina [15]. That way, they can exploit *locality* by connecting to adjacent neurons with simple, repeating motifs. Connective CPPNs have the same capability. In fact, geometric information in effect provides evolution with domain-specific bias, which is necessary if it is to gain an advantage over generic black-box optimization methods [93].

3.5 Substrate Resolution

As opposed to encoding a specific pattern of connections among a specific set of nodes, connective CPPNs in effect encode a general *connectivity concept*, i.e. the underlying mathematical relationships that produce a particular pattern. The consequence is that *same connective CPPN* can represent an equivalent concept at different resolutions (i.e. different node densities).

For neural substrates, the important implication is that the same ANN functionality can be generated at different resolutions. *Without further evolution*, previously-evolved connective CPPNs can be re-queried to specify the connectivity of the substrate at a new, higher resolution, thereby producing a working solution to the same problem at a higher resolution. For example, the resolution of inputs to an "eye"-like network could be increased without any need to retrain the network [79]. There is no upper bound on substrate resolution, that is, a connectivity concept is infinite in resolution. While the higher-resolution connectivity pattern may contain artifacts that were not expressed at the lower resolution at which it was evolved, it will still embody a good approximation of the general solution at the higher resolution. Thus, increasing substrate resolution introduces a powerful new kind of complexification to ANN evolution.

3.6 Evolving Connective CPPNs

Connective CPPNs are naturally evolved with NEAT. This approach is thus called *HyperNEAT* because NEAT evolves CPPNs that represent spatial patterns in hyperspace. Each point in the pattern, bounded by a hypercube, is interpreted as a connection in a lower-dimensional connected graph.

The HyperNEAT algorithm proceeds as shown in Algorithm 1.

1. Choose substrate configuration (i.e. node layout and input/output assignments).
2. Initialize population of minimal CPPNs with random weights.
3. Repeat until solution is found:
    a. For each member of the population:
        i. Query its CPPN output for the weight of each possible connection in the substrate. If the absolute value of that output exceeds a threshold magnitude, create the connection with a weight scaled proportionally to the output value (Fig. 4).
        ii. Run the substrate as an ANN in the task domain to ascertain fitness.
    b. Reproduce the CPPNs according to the NEAT method to produce the next generation's population.

Algorithm 1: HyperNEAT

Evolving a connective CPPN is not very different than a normal evolutionary algorithm. Fitness for a CPPN is determined by generating the weights for the substrate to create an ANN that is then evaluated in the problem domain. The CPPNs are then selected for reproduction based on their fitnesses and reproduced according the to NEAT algorithm.

In effect, as HyperNEAT adds new connections and nodes to the connective CPPN it is discovering new *global dimensions of variation* in connectivity patterns across the substrate. Early on it may discover overall symmetry, whereas later it may discover the concept of receptive fields. Each new connection or node in the CPPN represents a new way that an entire pattern can vary, i.e. a new regularity. In this way, HyperNEAT introduced a powerful new approach to evolving large-scale connectivity patterns and ANNs.

The next sections review work since HyperNEAT's introduction that has demonstrated its promise and extended its capabilities.

4 Key Properties

Several key features of HyperNEAT make it attractive as a method for machine learning. These features include the abilities to (1) compactly encode large networks with regularities and symmetries, (2) scale the size and resolution of solutions, and (3) leverage problem geometry. The initial HyperNEAT experiments in Stanley et al. [79], reviewed next, demonstrated each of these properties.

These initial experiments covered two domains: visual discrimination and food gathering. Both tasks are simple, but require clear regularities to solve. In visual discrimination, a $N \times N$ grid of nodes receives an input signal that contains two boxes, one large and one small. The network must identify the center of the large box by outputting its position on another $N \times N$ grid of neurons. Food gathering requires controlling a single agent that must efficiently collect food by exploiting geometrically correlated sensors and effectors. A solution for both tasks is to repeat the same connectivity pattern for all inputs, a concept that is easily captured by HyperNEAT.

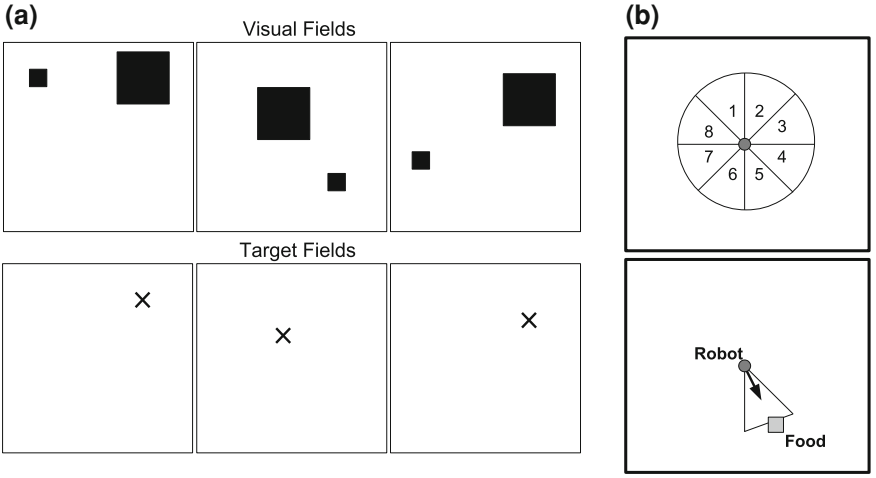

Fig. 7 Preliminary Experiments. The visual discrimination task (**a**) is to identify the *center* of the larger box. Example visual field activation patterns (*top*) and the corresponding correct target fields (*bottom*) are depicted. The "X" in each target field denotes the point of highest activation, which is how the ANN specifies the location of the *center* of the larger box. In the food gathering task (**b**), a robot with eight sensors and eight corresponding effectors (that is, each effectors moves the robot in the direction of a sensor) must detect and collect food that appears in the environment. The main goal is for the robot to learn to efficiently activate its effectors so that it moves towards the food as quickly as possible

Similarly, in both experiments the importance of geometry is clear: The location of the boxes on the grid and the location of the sensors and effectors in the food gathering robot are critical to the solution. By designing a substrate to exploit the geometry of these tasks, HyperNEAT was able to discover effective solutions. In both cases, an important concept is locality: Sensors should excite nearby effectors. However, a direct encoding can have no understanding of the locations of sensors and thus must discover this idea in a piece-wise, inefficient manner while HyperNEAT can simply discover the concept once and repeat it throughout the substrate.

Both problems also require discovering key regularities: For example, in food gathering, all inputs must excite a single output while inhibiting others. A direct encoding would have to learn this pattern for each input individually, but for an indirect encoding like HyperNEAT this concept need only be discovered once and then it can be repeated as needed. Figure 8 gives an example of such regularities discovered in the food-gathering domain.

Finally, both experiments showcased the ability to significantly scale up solutions found on smaller networks to larger networks without additional learning. In the case of visual discrimination, networks were trained on an 11×11 grid, but were later scaled to 33×33 and 55×55 grids, requiring *one million and nine million connections*, respectively. The resulting networks were still able to solve the problem, as shown in Fig. 9. For food gathering, networks were trained with 8 sensors and

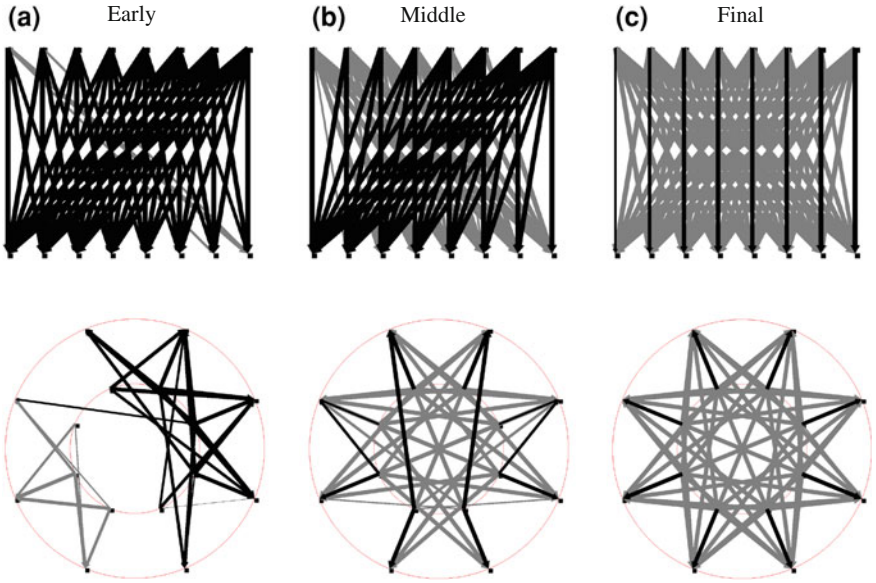

Fig. 8 HyperNEAT Discovers Regularities in Food Gathering. Each column shows networks for the food gathering tasks from two different substrate configurations. The best solution to this problem is for a sensor to excite a corresponding effector and inhibit all other effectors (i.e. if it sees a piece of food, go towards it and no other direction). Early-generation connective CPPNs for both configurations typically produce simple connectivity patterns (**a**). Eventually, HyperNEAT begins to exploit regularities (**b**), though they may not be the most fundamental ones. Finally, HyperNEAT discovers the fundamental regularity that underlies the task for the given substrate configuration (**c**). Thus, instead of optimizing individual connection weights, evolution is a process of gradually discovering and exploiting holistic regularities

effectors, but were later scaled to 16, 32, 128, and 1,024 sensors and effectors (for a maximum of over one million connections). The sensitivity of this task to precise changes meant that while scaled agents could still solve the task when scaled, their efficiency was reduced. However, if the agents were allowed to evolve further at the new scale, perfect solutions were found rapidly, which means that the general concept was retained and simply needed slight tweaking to work at the new size. This capability to scale networks to new sizes, which was new to neuroevolution, is possible because HyperNEAT represents solutions as patterns that can be sampled at any resolution rather than one-to-one mappings as in direct encodings.

Beyond these initial experiments, there have been several other studies that focus on the capabilities of HyperNEAT. [18–20] explored the effects of solution regularity on the ability of HyperNEAT to solve a problem in several domains. These experiments showed that the more regular a problem, the better HyperNEAT is at discovering a solution. Results also showed that solutions found by HyperNEAT were typically more general and exhibited more regularities in both phenotypic behavior and ANN structure than the solutions discovered by direct encodings such as

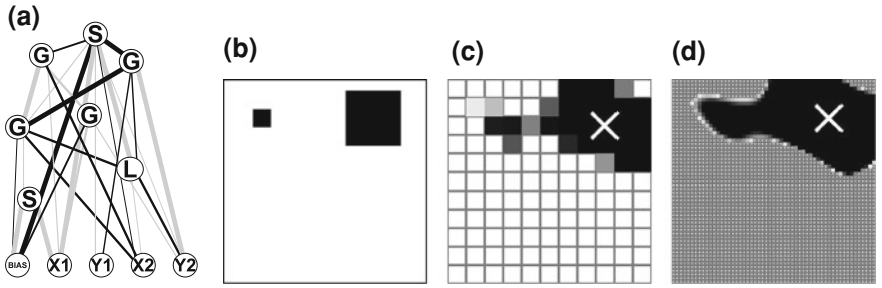

Fig. 9 Activation Patterns of the Same Connective CPPN at Different Resolutions in the Boxes Domain. Activation patterns on the target field of a substrate generated by the CPPN in (**a**) from the input trial shown in (**b**) are displayed at resolution 11 × 11 in (**c**) and 55 × 55 in (**d**). Darker color signifies higher activation and the position of highest activation is marked with a *white* "X." The same 26-connection CPPN generates solutions at both resolutions, with 10,328 and 8,474,704 connections, respectively, demonstrating the ability of the solution to scale significantly

regular NEAT. Coleman [22] also revisited the visual processing domain to further explore the capabilities of HyperNEAT and found it beneficial across a variety of more difficult visual tasks.

5 Applications of HyperNEAT

Since its introduction, HyperNEAT has been applied to a variety of problem domains, which are reviewed in this section.

5.1 Game Playing

Board games are a popular domain for artificial intelligence because of their explicit rules and the opportunity for looking ahead. HyperNEAT is a natural choice for learning strategies in many such games because of the inherent regularities of both their rules and board structure. For example, Gauci and Stanley [36] trained HyperNEAT to act as a board evaluator for a checkers player by inputting an entire board state and outputting a score for that state (Fig. 10a). HyperNEAT was able to exploit the regularities inherent in checkers (e.g. jumping a piece is good almost everywhere) to create effective checkers players. This work was later extended to the more computationally challenging game of Go [37], wherein HyperNEAT-trained players were again able to exploit geometry and repetition to quickly discover effective strategies. More interestingly, like a human player learning to play Go, networks could be trained on small board sizes (e.g. 5 × 5) and scaled up to larger sizes while still maintaining knowledge gained at the smaller size. With additional training at the new

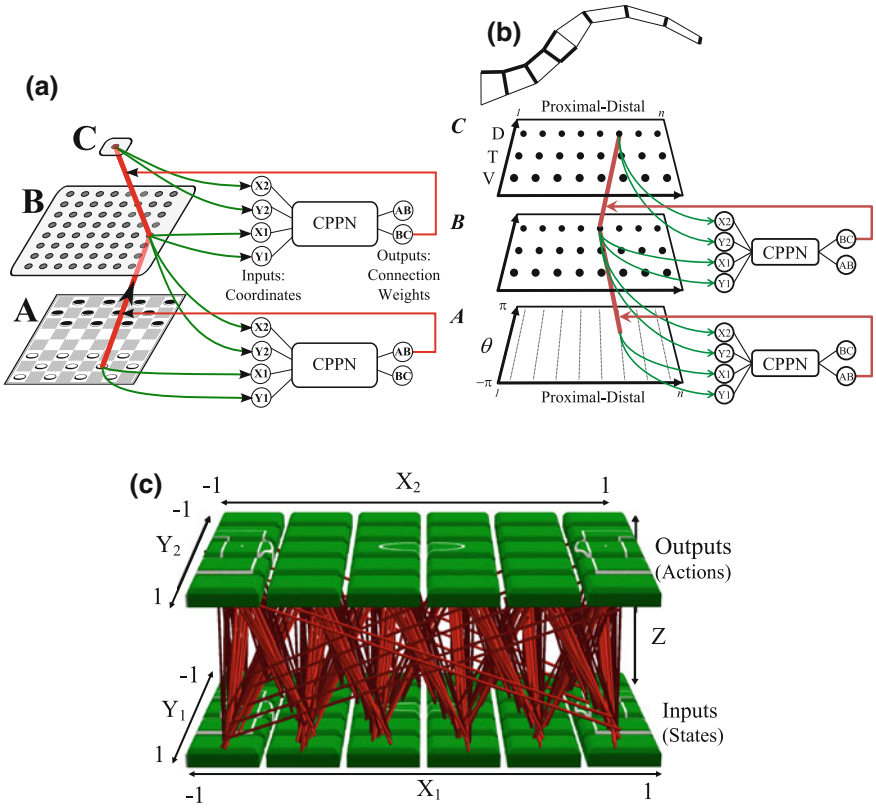

Fig. 10 HyperNEAT Domains. The HyperNEAT approach has been applied to a variety of domains including checkers (**a**), octopus arm control (**b**), and Robocup soccer (**c**) **a** Checkers; reproduced from Gauci and Stanley [36] **b** Octopus Arm; reproduced from Woolley and Stanley [95] **c** Robocup Soccer; reproduced from Verbancsics and Stanley [88]

sizes, such networks could further improve as they developed strategies only possible at the larger sizes. Bahceci and Miikkulainen [9] demonstrated similar results by training a player in a simple board game to recognize patterns at small board sizes and then scaling the size up.

5.2 Control

HyperNEAT is also an attractive option for learning autonomous agent controllers because the worlds these agents inhabit (including the real world) typically include regularities that can be exploited. Such domains also may contain a large number of inputs (a strong suit of HyperNEAT), such as the Balanced Diet contest at GECCO 2008, which required an agent to navigate a complex world while collecting food.

The winner of this contest, NeuroHunter [52], took advantage of HyperNEAT's ability to integrate large numbers of inputs to solve the task. Learning to drive simulated cars is another control application of HyperNEAT, as shown by Drchal et al. [30, 31]. Their experiment demonstrated a group of simulated cars learning to drive on the same side of the road to avoid collisions. In effect, the agents invented their own traffic rules. Woolley and Stanley [94] applied HyperNEAT to the control of a simulated octopus arm (Fig. 10b) that can dynamically scale the number of arm segments. HyperNEAT has also been demonstrated in robot navigation [57]. Finally, Clune et al.[19] and Yosinski et al. [97] applied HyperNEAT to evolving gaits for simulated and real quadruped robots, demonstrating its real-world potential.

5.3 Robocup

HyperNEAT has also been applied to the Robocup simulated soccer domain of Keepaway. Verbancsics and Stanley [89] employed a substrate configuration called the Bird's-Eye View (BEV) to simultaneously input entire soccer Keepaway scenarios into an ANN as seen from above (Fig. 10c), resulting in the longest Keepaway benchmark holding time yet recorded at the time of it's publication. This approach was extended later to take skills learned in Keepaway and transfer them to other domains [88], including harder versions of Keepaway. Additionally, Lowell et al. [61] also demonstrated the effectiveness of HyperNEAT in the Robocup domain.

6 Extensions and Implementations

Perhaps the most significant contribution of HyperNEAT is that it opens up a new research direction in evolving indirectly-encoded neural networks, which creates many opportunities for extending the core idea. This section reviews several such extensions introduced so far.

6.1 Plasticity: Adaptive HyperNEAT

Much research with HyperNEAT so far has focused on producing networks whose weights are fixed. However, it has been shown previously that plastic networks, that is, networks whose connection weights change over time, can solve problems more efficiently than those whose with fixed weights [12, 34]. Risi and Stanley [66] thus developed an extension to HyperNEAT that allows it to create such plastic networks, with the advantage over traditional such systems that HyperNEAT can also learn the learning rules that control how weights were updated. Another potential advantage of HyperNEAT-encoded plasticity is that HyperNEAT can encode a *pattern* of

learning rules (i.e. as opposed to only weights) across the geometry of the network. Two approaches were tested: *iterated*, which means re-querying the CPPN for new weights at every time step, and *ABCD*, in which the CPPN outputs parameters for a generalized Hebbian rule in addition to the initial weight value of the network.

6.2 Indirect-then-Direct: HybrID

In an attempt to improve the performance of HyperNEAT in domains that are highly irregular, Clune et al. [20, 21] introduced *Hybridized Indirect and Direct encoding* (HybrID), an extension to HyperNEAT that combines the ability of an indirect encoding to find and exploit regularities with a direct encoding's ability to optimize individual parameters. The approach works by running HyperNEAT for a fixed number of generations and then switching to regular NEAT to decide the weights of the substrate (without allowing topological mutations) for the remainder of search. The idea is that HyperNEAT can quickly find the gross regularities of the problem and then NEAT can tweak the final weights. The following two extensions to HyperNEAT also seek to address the issue of performance on problems with varying levels of regularity.

6.3 Decoupling Topology and Weighting: HyperNEAT-LEO

The human experimenter must define the nodes that can be connected in a neural network produced by HyperNEAT. However, the original HyperNEAT can choose not to express a particular connection by outputting a value below a defined threshold. The problem is, as shown by Clune et al. [17], that this simple rule creates an obstacle when the domain requires specific discontinuities or modules in its neural architecture. Verbancsics and Stanley [90] proposed a solution to this problem called the Link Expression Output (LEO) that adds an output to the CPPN that determines whether the currently-queried connection should be expressed or not, thereby *decoupling* link expression and link weight. This approach was successful at solving a variant of the retina problem [54] (which requires explicit modularity) that Clune et al. [17] earlier showed was challenging for the original HyperNEAT (Fig. 11a).

6.4 Dynamic Substrate Design: ES-HyperNEAT

As shown in many HyperNEAT experiments, substrates allow the injection of knowledge about a domain, such as the relationship of a sensor to an effector or the relative positioning of inputs, which are typically intuitive and easy to define. What is less clear to the experimenter, however, is the appropriate positions and number of hidden nodes that should be available on the substrate. Typically the approach has been to

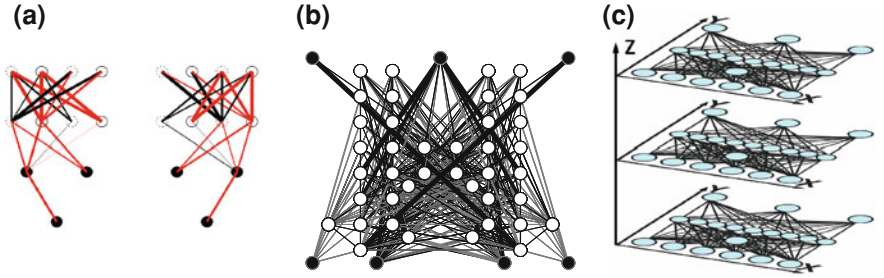

Fig. 11 HyperNEAT Extensions. The HyperNEAT approach has been extended in a variety of ways. Some examples include: evolving modular networks (**a**), evolving the substrate layout (**b**), and evolving multiagent teams (**c**). **a** HyperNEAT-LEO; reproduced from Verbancsics and Stanley [90] **b** ES-HyperNEAT; reproduced from Risi and Stanley [67] **c** Multiagent HyperNEAT; reproduced from D'Ambrosio et al. [24]

designate some number of hidden layers, each with the same number of nodes (usually equal to the number of inputs or outputs). However, it is desirable that HyperNEAT, like NEAT before it, can *learn* its internal node placement, while still exploiting the relationships present in inputs and outputs. Risi and Stanley [67, 68] introduced such an extension called Evolvable-Substrate HyperNEAT (ES-HyperNEAT). By searching the pattern generated by a CPPN for clues to where nodes and connections should be placed, ES-HyperNEAT can determine the best locations to place hidden nodes. Figure 11b shows an example network created by ES-HyperNEAT. This approach was demonstrated to be more efficient than the original HyperNEAT at solving a maze navigation task and a task that required switching sensors [67, 68]. ES-HyperNEAT has also been successfully combined with HyperNEAT-LEO [68] and Adaptive HyperNEAT [69].

6.5 Morphological Evolution with CPPNs

Although not strictly an extension of HyperNEAT, Auerbach and Bongard [5] introduced an approach wherein a CPPN determines the structure of a three-dimensional body-plan morphology. This approach was extended [4, 6] also to allow the CPPN to construct the control scheme for these morphologies, which leads to efficient body-brain evolution. Auerbach and Bongard [7, 8] also recently introduced new ways for CPPNs to generate morphology by creating creatures directly from shapes output by the CPPN. Liapis et al. [59] also applied CPPNs to evolving morphologies of spaceships.

6.6 Replacing NEAT: HyperGP

Buk et al. [14] replaced the traditional CPPN in HyperNEAT with a genetic programming tree. This idea demonstrates that any type of pattern generator may be used to query the substrate to generate ANNs. It is likely that some patterns can be more easily evolved with different representations than others. However, NEAT's ability to increase complexity over generations and the natural ability of CPPNs to encode arbitrary regularities (given enough hidden nodes) remain appealing advantages offered by CPPNs.

6.7 Multiagent HyperNEAT

Another extension to HyperNEAT is the capability to evolve multiple, related neural networks simultaneously. The benefit of this approach is that teams of agents typically significantly overlap in their capabilities and can thus benefit from being generated by the same source (Fig. 11c). Additionally, much as HyperNEAT can exploit geometry in the layout of sensors and effectors, this extension can exploit geometry in the structure of the *team*, either physically (i.e. their layout on a field) or conceptually (i.e. based on capabilities). This approach has been applied successfully in several domains including predator-prey [27], room-clearing [24], mobile sensor networks [55], and patrol [25], where it was deployed in real robots. It has also been extended to define the communication scheme for a team [23]. Additionally, both Hiller and Lipson [47] and Haasdijk et al. [43] applied techniques reminiscent of multiagent HyperNEAT to evolving the controller of modular robots.

7 Discussion and Future Directions

At the time HyperNEAT was introduced, most research in the GDS community focused on the *development* process. However, HyperNEAT abstracts away this complex and time-consuming system while still effectively capturing its benefits, e.g. by generating regularities and patterns. The success of HyperNEAT across the wide range of domains and applications that have followed, and the extensions it has inspired, lend further credence to this choice.

The large number of extensions made to the basic HyperNEAT approach also suggest that there is still a great deal to learn about the method. Many of these extensions focus on the substrate, which is likely because topology-evolving neuroevolution methods such as the original NEAT have proven to be simpler and more effective than methods that require the experimenter to define the topology *a priori*. Another promising direction is to enable encoding different types of neurons, such as the plastic neurons in [66]. Neural models like continuous time recurrent neural networks (CTRNNs) [10] and spiking neural networks (SNNs) [62] are not only more

biologically plausible than traditional ANNs, but also exhibit interesting properties that may be facilitated by HyperNEAT.

HyperNEAT has made it possible to create very large and complex ANNs with millions or more connections, yet few of the applications to date fully exploit this capability. Researchers have not typically had the ability to work with and create such large networks before, so it is understandable that we do not yet fully envision how to best take advantage of them in nontrivial ways. It is also possible that the kinds of control and decision-making problems that are being posed to HyperNEAT so far are not sufficient to require such large networks; while one possible path to complexity is through more challenging problems, another route may be through more open-ended approaches like novelty search [58].

Thus HyperNEAT may be yet to reach its full potential. Interestingly, by combining the ability to encode very large and regular neural structures with sophisticated neural models (such as plastic or spiking neurons), it is possible that in the future HyperNEAT-like algorithms may eventually approach designs reminiscent of natural brains.

Acknowledgments Much of the work on HyperNEAT at the Evolutionary Complexity Research Group at the University of Central Florida was supported by DARPA through its Computer Science Study Group program, including Phases 1, 2, and 3 (grants HR0011- 08-1-0020, HR0011-09-1-0045 and N11AP20003). This chapter does not necessarily reflect the position or policy of the government, and no official endorsement should be inferred.

Appendix: Implementations

Many researchers mentioned above have made their implementations of HyperNEAT available:

- HyperSharpNEAT by David D'Ambrosio is an extension of Colin Green's SharpNEAT1, which is written in C#. HyperSharpNEAT contains an implementation of multiagent HyperNEAT and has been augmented by Joel Lehman and Sebastian Risi to contain the LEO, ES, and adaptive HyperNEAT extensions.
- SharpNEAT2 by Colin Green, which includes a different version of HyperNEAT, is also written in C# and contains several example experiments as well as tutorials on making new ones.
- Keepaway HyperNEAT C# by Philip Verbancsics is distributed with a C# implementation of the Robocup simulator.
- Another HyperNEAT Implementation (ANHI) by Oliver Coleman extends the Java-based ANJI implementation of NEAT (itself by Derek James and Philip Tucker). AHNI includes several visual processing experiments.
- HyperNEAT C++ by Jason Gauci comes with visual discrimination and checkers experiments.

Links to all these software packages can be found at http://eplex.cs.ucf.edu/hyperNEATpage/.

References

1. T. Aaltonen et al., (over 100 authors). Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF. Phys. Rev. Lett. **102**(15), 2001 (2009)
2. L. Altenberg, Evolving better representations through selective genome growth. in *Proceedings of the IEEE World Congress on Computational Intelligence* (IEEE Press, Piscataway, NJ, 1994), pp. 182–187
3. P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Networks **5**, 54–65 (1993)
4. J.E. Auerbach, J.C. Bongard, Dynamic resolution in the co-evolution of morphology and control. in *Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems (ALife XII)* (2010)
5. J.E. Auerbach, J.C. Bongard, Evolving CPPNs to grow three dimensional structures. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)* (ACM Press, New York, NY, 2010)
6. J.E. Auerbach, J.C. Bongard, Evolving complete robots with CPPN-NEAT: the utility of recurrent connections. in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation* (ACM, 2011)
7. J.E. Auerbach, J.C. Bongard, On the relationship between environmental and mechanical complexity in evolved robots. in *13th International Conference on the Synthesis and Simulation of Living Systems (ALife XIII)* (ACM, 2012)
8. J.E. Auerbach, J.C. Bongard, On the relationship between environmental and morphological complexity in evolved robots. in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation* (ACM, 2012)
9. E. Bahceci, R. Miikkulainen, Transfer of evolved pattern-based heuristics in games. in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-2008)* (IEEE Press, Piscataway, NJ, 2008)
10. R.D. Beer, J.C. Gallagher, Evolving dynamical neural networks for adaptive behavior. Adapt. behav. **1**(1), 91–122 (1992)
11. P.J. Bentley, S. Kumar, Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)* (1999), pp. 35–43
12. J. Blynel, D. Floreano, Levels of dynamics and adaptive behavior in evolutionary neural controllers. in *Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior on From Animals to Animats* (2002), pp. 272–281
13. J.C. Bongard, Evolving modular genetic regulatory networks. in *Proceedings of the 2002 Congress on Evolutionary Computation* (2002)
14. Z. Buk, J. Koutník, M. Šnorek, NEAT in HyperNEAT substituted with genetic programming. in *Adaptive and Natural Computing Algorithms* (2009), pp. 243–252
15. D.B. Chklovskii, A.A. Koulakov, Maps in the brain: what can we learn from them? Annu. Rev. Neurosci. **27**, 369–392 (2004)
16. P.M. Churchland, Some reductive strategies in cognitive neurobiology. Mind **95**, 279–309 (1986)
17. J. Clune, B.E. Beckmann, P.K. McKinley, C. Ofria, Investigating whether HyperNEAT produces modular neural networks. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)* (ACM Press, New York, NY, 2010)
18. J. Clune, C. Ofria, R.T. Pennock, How a generative encoding fares as problem-regularity decreases. in *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN 2008)* (Springer, Berlin, 2008), pp. 258–367
19. J. Clune, R.T. Pennock, C. Ofria. The sensitivity of HyperNEAT to different geometric representations of a problem. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)* (ACM Press, New York, NY, USA, 2009)
20. J. Clune, K.O. Stanley, R.T. Pennock, C. Ofria, On the performance of indirect encoding across the continuum of regularity. IEEE Trans. Evol. Comput. **15**(3), 346–367 (2011)

21. J. Clune, K.O. Stanley, R.T. Pennock, C. Ofria, On the performance of indirect encoding across the continuum of regularity. IEEE Trans. Evol. Comput. **15**(3), 346–367 (2011)
22. O.J. Coleman et al., Evolving neural networks for visual processing. Ph.D. thesis, BS Thesis. The University of New South Wales, 2010
23. D.B. D'Ambrosio, S. Goodell, J. Lehman, S. Risi, K.O. Stanley, *Multirobot Behavior Synchronization Through Direct Neural Network Communication* (Springer, New York, 2012)
24. D.B. D'Ambrosio, J. Lehman, S. Risi, K.O. Stanley, Evolving policy geometry for scalable multiagent learning. in *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)* (International Foundation for Autonomous Agents and Multiagent System, 2010), pp. 731–738
25. D.B. D'Ambrosio, J. Lehman, S. Risi, K.O. Stanley, Task switching in multiagent learning through indirect encoding. in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2011)* (IEEE, Piscataway, NJ, 2011)
26. D.B. D'Ambrosio, K.O. Stanley, A novel generative encoding for exploiting neural network sensor and output geometry. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)* (ACM Press, New York, NY, 2007)
27. D.B. D'Ambrosio, K.O. Stanley, Generative encoding for multiagent learning. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)* (ACM Press, New York, NY, 2008)
28. F. Dellaert, Toward a biologically defensible model of development. Master's thesis, Case Western Reserve University, Clevekand, OH, 1995
29. P. Deloukas, G.D. Schuler, G. Gyapay, E.M. Beasley, C. Soderlund, P. Rodriguez-Tome, L. Hui, T.C. Matise, K.B. McKusick, J.S. Beckmann, S. Bentolila, M. Bihoreau, B.B. Birren, J. Browne, A. Butler, A.B. Castle, N. Chiannilkulchai, C. Clee, P.J. Day, A. Dehejia, T. Dibling, N. Drouot, S. Duprat, C. Fizames, D.R. Bentley, A physical map of 30,000 human genes. Science **282**(5389), 744–746 (1998)
30. J. Drchal, O. Kapra, J. Koutnik, M. Snorek, Combining multiple inputs in HyperNEAT mobile agent controller. in *19th International Conference on Artificial Neural Networks (ICANN 2009)* (Berlin, Springer, 2009), pp. 775–783
31. J. Drchal, J. Koutnik, M. Snorek, HyperNEAT controlled robots learn to drive on roads in simulated environment. in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009)* (IEEE Press, Piscataway, NJ, USA, 2009)
32. P. Eggenberger, *Evolving Morphologies of Simulated 3D Organisms Based on Differential Gene Expression* (MIT Press, Boston, 1997), pp. 205–213
33. D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning. Evol. Intell. **1**, 47–62 (2008)
34. D. Floreano, J. Urzelai, Evolutionary robots with on-line self-organization and behavioral fitness. Neural Netw. **13**, 431–4434 (2000)
35. J. Gauci, K.O. Stanley, Generating large-scale neural networks through discovering geometric regularities. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)* (ACM Press, New York, NY, 2007)
36. J. Gauci, K.O. Stanley, Autonomous evolution of topographic regularities in artificial neural networks. Neural Comput. **22**(7), 1860–1898 (2010)
37. J. Gauci, K.O. Stanley, Indirect encoding of neural networks for scalable go. in *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN-2010)* (Springer, 2011), pp. 354–363
38. C.D. Gilbert, T.N. Wiesel, Receptive field dynamics in adult primary visual cortex. Nature **356**, 150–152 (1992)
39. F. Gomez, R. Miikkulainen, Incremental evolution of complex general behavior. Adapt. Behav. **5**, 317–342 (1997)
40. F. Gomez, R. Miikkulainen, Solving non-Markovian control tasks with neuroevolution (1999), pp. 1356–1361
41. G.J. Goodhill, M.A. Carreira-Perpinn, Cortical Columns, in *Encyclopedia of Cognitive Science*, volume 1, ed. by L. Nadel (MacMillan Publishers Ltd., London, 2002), pp. 845–851

42. F. Gruau, D. Whitley, L. Pyeatt, in *A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks*, ed. by R. John Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo. Genetic Programming 1996: Proceedings of the First Annual Conference (MIT Press, 1996), pp. 81–89
43. E. Haasdijk, A.A. Rusu, A.E. Eiben, HyperNEAT for locomotion control in modular robots. in *Proceedings of the 9th International Conference on Evolvable Systems (ICES 2010)* (2010)
44. I. Harvey, The artificial evolution of adaptive behavior. Ph.D. thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993
45. E. Hastings, R. Guha, K.O. Stanley, Evolving content in the galactic arms race video game. in *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-09)* (IEEE Press, Piscataway, NJ, 2009)
46. E.J. Hastings, R.K. Guha, K.O. Stanley, Automatic content generation in the galactic arms race video game. IEEE Trans. Comput. Intell. AI Games **1**(4), 245–263 (2010)
47. J.D. Hiller, H. Lipson, Evolving amorphous robots. in *Proceedings of the Twelfth International Conference on Artificial Life (ALIFE XII)* (2010)
48. A.K. Hoover, Michael P. Rosario, K.O. Stanley, in *Scaffolding for Interactively Evolving Novel Drum Tracks for Existing Songs*, ed. by M. Giacobini. Proceedings of the Sixth European Workshop on Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMUSART 2008) (Springer, March 2008), pp. 412–422
49. A.K. Hoover, K.O. Stanley, Exploiting functional relationships in musical composition. Connect. Sci. Spec. Issue Music Brain Cogn. **21**(2 and 3), 227–251 (2009)
50. G.S. Hornby, J.B. Pollack, Creating high-level components with a generative representation for body-brain evolution. Artif. Life **8**(3), 223–246 (2002)
51. D.H. Hubel, T.N. Wiesel, Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. J. Neurophysiol. **28**, 229–289 (1965)
52. W. Jaskowski, K. Krawiec, B. Wieloch, Neurohunter-an entry for the balanced diet contest (2008)
53. E.R. Kandel, J.H. Schwartz, T.M. Jessell, *Principles of Neural Science*, 3rd edn. (Elsevier, New York, 1991)
54. N. Kashtan, U. Alon, Spontaneous evolution of modularity and network motifs. Proc. Nat. Acad. Sci. U.S.A. **102**(39), 13773 (2005)
55. D.B. Knoester, H.J. Goldsby, P.K. McKinley, Neuroevolution of mobile ad hoc networks. in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (ACM, 2010), pp. 603–610
56. N. Kohl, K.O. Stanley, R. Miikkulainen, M. Samples, R. Sherony, Evolving a real-world vehicle warning system. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)* (July 2006), pp. 1681–1688
57. J. Lehman, S. Risi, D.B. D'Ambrosio, K.O. Stanley, *Rewarding Reactivity to Evolve Robust Controllers Without Multiple Trials or Noise* (MIT Press, Cambridge, 2012)
58. J. Lehman, K.O. Stanley, Abandoning objectives: evolution through the search for novelty alone. Evol. Comput. **19**(2), 189–223 (2011)
59. A. Liapis, G.N. Yannakakis, J. Togelius, Optimizing visual properties of game content through neuroevolution. in *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference* (2011)
60. A. Lindenmayer, Adding Continuous Components to L-Systems, in *Lecture Notes in Computer Science 15*, ed. by G. Rozenberg, A. Salomaa (Springer, Heidelberg, 1974), pp. 53–68
61. J. Lowell, S. Grabkovsky, K. Birger, Comparison of NEAT and HyperNEAT performance on a strategic decision-making problem. in *Fifth International Conference on Genetic and Evolutionary Computing (ICGEC) 2011* (IEEE, 2011), pp. 102–105
62. W. Maass, C.M. Bishop, *Pulsed Neural Networks* (The MIT Press, Cambridge, 2001)
63. A.P. Martin, Increasing genomic complexity by gene duplication and the origin of vertebrates. Am. Nat. **154**(2), 111–128 (1999)
64. J.F. Miller, Evolving a self-repairing, self-regulating, French flag organism. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)* (Springer, Berlin, 2004)

65. R.A. Raff, *The Shape of Life: Genes, Development, and the Evolution of Animal Form* (The University of Chicago Press, Chicago, 1996)
66. S. Risi, K.O. Stanley, Indirectly encoding neural plasticity as a pattern of local rules. in *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)* (Springer, Berlin, 2010)
67. S. Risi, K.O. Stanley, Enhancing ES-HyperNEAT to evolve more complex regular neural networks. in *GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011)
68. S. Risi, K.O. Stanley, An enhanced hypercube-based encoding for evolving the placement, density and connectivity of neurons. Artif. Life **18**(4), 331–363 (2012)
69. S. Risi, K.O. Stanley, *A Unified Approach to Evolving Plasticity and Neural Geometry* (IEEE, Piscataway, 2012)
70. S. Risi, S.D. Vanderbleek, C.E. Hughes, K.O. Stanley, How novelty search escapes the deceptive trap of learning to learn. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)* (ACM Press, New York, NY, USA, 2009)
71. N. Saravanan, D.B. Fogel, Evolving neural control systems. IEEE Expert **10**(3), 23–27 (1995)
72. J. Secretan, N. Beato, D.B. D'Ambrosio, A. Rodriguez, A. Campbell, J.T. Folsom-Kovarik, K.O. Stanley, Picbreeder: a case study in collaborative evolutionary exploration of design space. Evol. Comput. **19**(3), 345–371 (2011)
73. J. Secretan, N. Beato, D.B. D'Ambrosio, A. Rodriguez, A. Campbell, K.O. Stanley, Picbreeder: evolving pictures collaboratively online. in *CHI '08: Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems* (ACM, New York, NY, USA, 2008), pp. 1759–1768
74. K. Sims, *Evolving 3D Morphology and Behavior by Competition* (MIT Press, Cambridge, MA, 1994), pp. 28–39
75. K.O. Stanley, Exploiting regularity without development. in *Proceedings of the AAAI Fall Symposium on Developmental Systems* (AAAI Press, Menlo Park, CA, 2006)
76. K.O. Stanley, Compositional pattern producing networks: a novel abstraction of development. Genet. Program. Evolvable Mach. Spec. Issue Dev. Syst. **8**(2), 131–162 (2007)
77. K.O. Stanley, B.D. Bryant, R. Miikkulainen, Evolving neural network agents in the NERO video game. in *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games* (2005)
78. K.O. Stanley, B.D. Bryant, R. Miikkulainen, Real-time neuroevolution in the NERO video game. IEEE Trans. Evol. Comput. Spec. Issue Evol. Comput. Games **9**(6), 653–668 (2005)
79. K.O. Stanley, D.B. D'Ambrosio, J. Gauci, A hypercube-based indirect encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
80. K.O. Stanley, N. Kohl, R. Miikkulainen, Neuroevolution of an automobile crash warning system. in *Proceedings of the Genetic and Evolutionary Computation Conference* (2005)
81. K.O. Stanley, R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)* (2002)
82. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**, 99–127 (2002)
83. K.O. Stanley, R. Miikkulainen, A taxonomy for artificial embryogeny. Artif. Life **9**(2), 93–130 (2003)
84. K.O. Stanley, R. Miikkulainen, Competitive coevolution through evolutionary complexification. J. Artif. Intell. Res. **21**, 63–100 (2004)
85. K.O. Stanley, R. Miikkulainen, Evolving a roving eye for go. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)* (Springer, Berlin, 2004)
86. M.E. Taylor, S. Whiteson, P. Stone, Comparing evolutionary and temporal difference methods in a reinforcement learning domain. in *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference* (July 2006), pp. 1321–1328
87. L. Trujillo, G. Olague, E. Lutton, F.F. de Vega, Discovering Several Robot Behaviors, through Speciation. Applications of Evolutionary Computing: Evoworkshops, Evocomnet. Evofin, Evohot, Evoiasp, Evomusart, Evonum, Evostoc, and Evotranslog (2008), p. 164

88. P. Verbancsics, K.O. Stanley, Evolving static representations for task transfer. J. Mach. Learn. Res. (JMLR) **11**, 1737–1769 (2010)
89. P. Verbancsics, K.O. Stanley, Task transfer through indirect encoding. in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)* (ACM Press, New York, NY, 2010)
90. P. Verbancsics, K.O. Stanley, Constraining connectivity to encourage modularity in HyperNEAT. in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (ACM, 2011), pp. 1483–1490
91. J.D. Watson, N.H. Hopkins, J.W. Roberts, J.A. Steitz, A.M. Weiner, *Molecular Biology of the Gene*, 4th edn. (The Benjamin Cummings Publishing Company Inc, Menlo Park, 1987)
92. D.J. Willshaw, C. Von Der Malsburg, How patterned neural connections can be set up by self-organization. Proc. R. Soc. Lond. B Biol. Sci. **194**(1117), 431–445 (1976)
93. D.H. Wolpert, W. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
94. B. Woolley, K.O. Stanley, Evolving a single scalable controller for an octopus arm with a variable number of segments. in *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN-2010)* (Springer, 2011), pp. 270–279
95. B.G. Woolley, K.O. Stanley, On the deleterious effects of a priori objectives on evolution and representation. in *GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011)
96. X. Yao, Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)
97. J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J.C. Zagal, H. Lipson, Generating gaits for physical quadruped robots: evolved neural networks vs. local parameterized search. in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation* (ACM, 2011), pp. 31–32
98. M.J. Zigmond, F.E. Bloom, S.C. Landis, J.L. Roberts, L.R. Squire (eds.), *Fundamental Neuroscience* (Academic Press, London, 1999)

Chapter 6
Using the Genetic Regulatory Evolving Artificial Networks (GReaNs) Platform for Signal Processing, Animat Control, and Artificial Multicellular Development

Borys Wróbel and Michał Joachimczak

Abstract Building a system that allows for pattern formation and morphogenesis is a first step towards a biologically-inspired developmental-evolutionary approach to generate complex neural networks. In this chapter we present one such system, for Genetic Regulatory evolving artificial Networks (GReaNs). We review the results of previous experiments in which we investigated the evolvability of the encoding used in GReaNs in problems which involved: (i) controlling development of multicellular 2-dimensional (2D) soft-bodied animats; (ii) controlling development of 3-dimensional (3D) multicellular artificial bodies with asymmetrical shapes and patterning; (iii) directed movement of unicellular animats in 2D; and (iv) processing signals at the level of single cells. We also report a recent introduction of spiking neuron models in GReaNs. We then present a road map towards using this system for evolution and development of neural networks.

1 Introduction

Biological development is a process that starts from one cell and, through multiple cell divisions, results in a complex 3-dimensional multicellular structure. This process is remarkably robust to variability in environmental conditions and to damage at the genetic and cellular level. It is believed that this robustness stems from the fact

B. Wróbel (✉) · M. Joachimczak
Systems Modelling Laboratory, IOPAS, Sopot, Poland
e-mail: wrobel@evosys.org

M. Joachimczak
e-mail: mjoach@gmail.com

B. Wróbel
Evolutionary Systems Laboratory, Adam Mickiewicz University in Poznan, Poznan, Poland

B. Wróbel
Institute for Neuroinformatics, University of Zurich/ETHZ, Zurich, Switzerland

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_6,

that in biological development the interactions are local and that the whole process is modular and hierarchical [16, 22, 23]. The hope for biologically-inspired engineered systems is that they will display similar properties. These properties include not only robustness to noise and damage, but also a remarkable efficiency of the encoding of complex multicellular structure in compact genomes, and—perhaps linked with it—the fact that this encoding can be moulded by natural selection to allow quick adaptation (the property of evolvability).

The process of biological development is shaped by pattern formation and morphogenesis. Pattern formation allows for the laying out of a body plan according to which cells differentiate in space. Morphogenetic mechanisms, which include differential growth and cell migration, result in changes of the shape of the developing embryo. These mechanisms are based on local interactions between cells. These interactions can involve physical contact between cells or diffusive substances (morphogens). Local interactions and morphogens allow for the expression of different genes (and production of different proteins) in cells which share (in principle) the same genome (the same DNA sequence). The set of proteins that are produced is controlled by other proteins in the cell which act at various steps of gene expression. Because these regulatory genes do not need to lie in the genome physically close to regulated genes, they are called trans-regulators. One important class of trans-regulators are genes which code for proteins which act at the first step of gene expression (transcription). Such proteins are called transcriptional factors (TFs). TFs physically bind to the DNA, to regulatory sequences that are close to sequences of regulated genes, often close to the regions where transcription starts (promoters). Such regulatory sequences in DNA are sometimes called cis-regulators. A gene regulatory network (GRN) is an abstract structure (a graph) that consists of nodes connected by edges or links. Nodes correspond to regulatory units (cis-regulators together with regulated genes, which may be trans-regulators). Links correspond to regulatory interactions—for example, physical binding of transcription factors to DNA.

Investigation of the general principles of evolution of development and of GRNs in biological organisms is difficult because the time scales involved are very large. Our knowledge of evolutionary relationships is incomplete because it is based on imperfect fossil records or similarities between existing biological entities (genes, genomes, or phenotypes of contemporary organisms). These similarities do not necessarily stem from shared evolutionary history, and the phylogenetic signal (information that allows one to infer evolutionary relationships) is often weakened by the fact that some traits in some lineages may change in parallel, while other traits may be affected by multiple changes, each obliterating the trace of the previous one. Moreover, our knowledge about the development of contemporary organisms is quite limited, even for several organisms (so called model organisms) for which the development was investigated in more detail. Although the genomic sequence is known for quite a number of multicellular organisms, it is difficult to decipher the information there contained. Finding which DNA sequences encode products (proteins or RNA) is a difficult endeavour, finding cis-regulators even more so. Our knowledge of the connectivity of biological GRNs is even more limited.

One of the fields of theoretical biology, Artificial Life, provides a parallel approach to the investigation of the general principles of evolution of life by proposing that these principles can be inferred using artificial bio-inspired systems (Artificial Life platforms). We have build such a platform, for Genetic Regulatory evolving artificial Networks (GReaNs). We have shown that our platform allows for evolution of GRNs that regulate asymmetrical 3D morphogenesis [10] and pattern formation [14]. We have also investigated the computational abilities of evolving GRNs at the level of single cells [12, 13]. GRN topology is encoded in a linear genome in a way that is inspired by previous work by Eggenberger Hotz [7], but with some important modifications. Similar models were recently used also by other authors (e.g., [20]), and several other models were formulated for evolving artificial gene regulatory networks regulating artificial embryogenesis [1, 3, 4, 6, 8, 18, 19]. An important feature of our model is that we use no grid in 2D or 3D space in which the cells divide (continuous space is used). Another feature is that we do not set any limit on the size of the genome, and thus no limit on the number of nodes in the GRN or the number of connections between nodes (in practice, the limit is, of course, imposed by the available computer memory). We believe that removing such constraints allows for the use of a model to test hypotheses about biological systems that are difficult to address otherwise.

In this chapter we present our research portfolio by reviewing some results previously obtained when using GReaNs to control development of 3D multicellular bodies [10], including the first successful attempt we are aware of at solving the so called "French flag problem" [24] in 3D [14]. We have also recently used GReaNs to evolve the development of body and control in 2D multicellular soft-bodied animats [9, 15]. We present also the results obtained when the regulatory networks in GReaNs were evolved for computational and signal processing tasks at the level of single cells [12, 13]. We finally mention the most recent extension to GReaNs—the introduction of spiking neural models [25]. We conclude with a discussion of how GReaNs could be extended further to evolve-develop complex neural networks in a biologically-inspired manner.

2 Gene Regulatory Evolving Artificial Networks Encoded in Linear Genomes

Artificial gene regulatory networks in GReaNs are specified by a linear genome. A genome (Fig. 1) is a list of *genetic elements* grouped in *regulatory units*. Genetic elements have several types. The most important division is between *trans-regulators* (*genes*—elements that encode *products*) and *cis-regulators* (*promoters*). Trans-regulators which have *affinity* to cis-regulators belonging to a regulatory unit regulate genes in this unit. In other words, they act like biological transcriptional factors. In experiments on multicellular development we allow some products to diffuse between cells. Such products are called *morphogens*. In addition, there are *special*

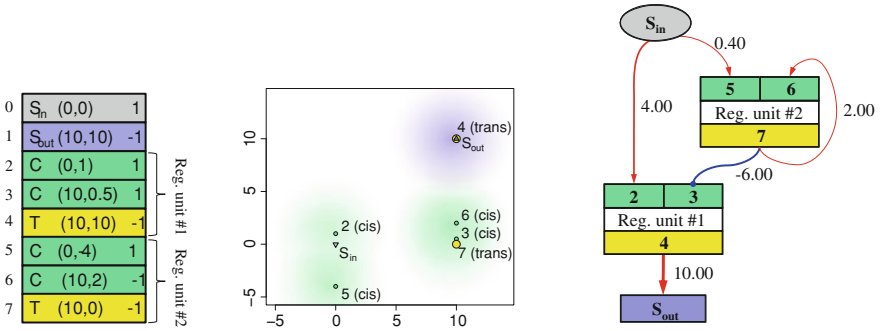

Fig. 1 Schematic structure of the genome encoding a GRN. Each 'genetic element' in the genome (*left*) is an ordered set of numbers: an integer specifying the type ('cis-regulator', *C*; 'trans-regulator', *T*; or 'special element', *S*), real numbers that specify a point in space (which determines trans-cis affinity), and a sign field (1 or −1, which determine if a trans-cis interaction is inhibitory or activatory). A 'regulatory unit', a node in the 'gene regulatory network' (GRN), is at least one cis-regulator followed by at least one trans-regulator. Special elements correspond to GRN inputs and outputs. Once the distances between the elements are calculated (*middle*), the topology of the regulatory network is determined (*right*)

elements. These elements correspond to inputs to the GRN (for example, *maternal factors*, whose gradient helps cellular differentiation) and outputs, which can be specific cell actions, such as division, death or differentiation, cell colour, or cell actions related to actuation.

Connectivity between the nodes in the GRN (nodes correspond to regulatory units) is determined by the affinity between the trans-regulators and cis-regulators. Special elements that are outputs can be seen as cis-elements hardwired to a gene coding one product with a specific function (for example, cell division). Inputs can be seen as special products whose concentration is determined externally to the cell.

Each genetic element (Fig. 1) corresponds to a point in abstract N-dimensional space (N is a parameter in the model; we typically use $N = 2$; the role of this parameter in the evolvability has been investigated in [11]). This space can be seen as the abstraction of the space of chemical interactions between macromolecules in biological space, in which the 'distance' (in terms of the similarity of 3D structure, electrostatic charge, etc.) affects affinity between molecules. In GReaNs, Euclidean distance between points corresponding to a cis-element and a trans-element determines product-promoter (trans-cis) *affinity*.

Concentrations of products in GReaNs change in each simulation step. In most of our work ([9, 11–17, 26], but see below for the discussion of a model in which the nodes in the network act as spiking neurons), these concentrations take a real value in the interval [0,1] and depend on the balance between the production rates calculated for each regulatory unit, and the degradation rate, equal for each product:

$$\Delta L = (\tanh \frac{A}{2} - L)\Delta t \tag{1}$$

where Δt (the integration time step) determines how fast the concentrations change in relation to the simulation time step, L is the current concentration of all product encoded in a regulatory unit, and A is the summed activation of all cis-regulators in the unit. The activation is calculated by taking into account concentrations which bind to the cis-regulator, having multiplied the concentration of products by their respective affinities, and taking into account if the interaction is inhibitory or activatory (in the recent work only additively [15, 16], but see also [13, 14]).

We believe that our model is biologically plausible. For example, cis-elements in many-to-many structure of regulatory units correspond to the biological reality of a sometimes high number of regions regulating gene expression, coded ultimately in the DNA, but which can act also at the level of RNA and/or after translation. The fact that there can also be many trans-regulators in one regulatory unit corresponds to the presence of co-expressed genes and/or genes coding multi-domain proteins in biological genomes. Biologically plausible encoding allows for the introduction of biologically plausible evolutionary mechanisms, which in turn opens up a possibility to test hypotheses on the role of these mechanisms in the evolution of genomes and biological networks.

In a typical evolutionary run in our experiments, the population is initiated with random genomes, and the size of the population is fixed (for example, to 100 or 300 individuals). When a genetic algorithm is used, it typically takes a few thousand generations, but we have recently investigated a more open-ended evolutionary algorithm, novelty search, in GReaNS [17]. Genetic operators in GReaNs can act either within a single *genetic element* or affect the number or order of elements in the genome. Operators acting on the level of individual elements cause changes in the numbers associated with the element. They can result in a modification of the element type, of whether the element will take an inhibitory or activatory role, or of to coordinates, which affects the *affinity* of the element (trans-cis interactions). Operators acting on the level of the genome include duplications and deletions of several continuous elements and exchange of elements between two genomes (recombination).

3 Asymmetrical Morphogenesis/Pattern Formation and Development of Multicellular Soft-bodied Animats

The embryogenesis model implemented in GReaNS allows for evolving the development of relatively large, non-trivial 2D or 3D morphologies [9, 10, 14–17]. In our model, development takes place in a continuous 3D (or 2D) fluid-like environment with elastic cells. The embryo growth starts from a single cell and proceeds through cell divisions. Each cell is controlled by the same genome and GRN, and has an associated *division vector*. Division is asymmetric and can be seen as a creation of a daughter cell by a mother in the direction of the mother's division vector. The daughter inherits product concentrations from the mother. At the time of division the direction of the daughter's division vector is set; it depends on the activation of spe-

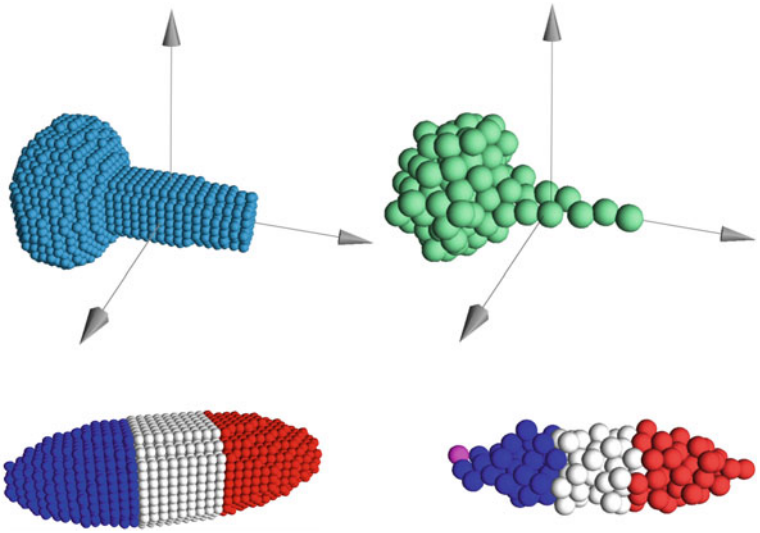

Fig. 2 Evolved 3D multicellular bodies with an asymmetrical shape (*top*) and patterning (*bottom*). *Right panels* show evolved bodies; *left panels* show voxelized targets

cial genetic elements in the mother. When two cells are close enough, they adhere to each other, but when they are too close they push one another away. At division, the daughter cell is placed sufficiently close to the mother that these two cells push each other away. Fluid drag is simulated to prevent erratic movements. Division occurs when a cellular output (a product coded by a special element) responsible for division crosses a pre-set threshold. The concentrations of morphogens perceived in a cell depend on the distance from the cells which produce them. Otherwise, in terms of their effects on promoters, they behave in the same way as internally produced TFs.

We use either a specific 3D (or 2D) target for development (see Fig. 2 for examples), with a fitness function which measures similarity of the developed shape or pattern of differentiation to the target, or—more recently—a more open-ended approach in which the search is for multicellular structures which are different from the structures obtained thus far (novelty search; [17]). This latter approach can be used to investigate the repertoire of shapes which can evolve in an Artificial Embryology system. When evolving embryos with asymmetrical patterning, special products in the genome determine cell colour.

We have recently extended GReaNs [9, 15] with a transformation of the multicellular structures into 2D soft-bodied animats. These animats are lacking, at this stage of the development of the GReaNs platform, neuronal control of actuation. Perhaps, however, this is what makes them interesting—they provide a possibility to investigate the properties of other ways in which locomotion can be controlled (for example, through diffusive factors, like in the locomotion of slime molds, see [9] for discussion), possibly in comparison to neuronal control.

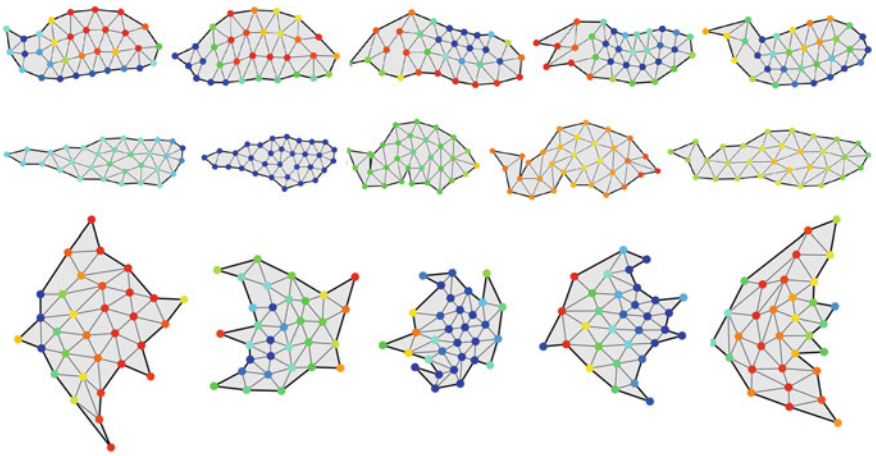

Fig. 3 Evolved 2D multicellular soft-bodied animats swimming in a fluid environment with different strategies. One strategy is based on undulation (*top*; evolved when the drag was high), in the other a "tail" (moderate drag; *middle*), and symmetrical "appendages" (low drag; *bottom*) are used. All animats move towards the *right*

The transformation of a multicellular structure to a soft-bodied animat takes several steps. First neighbouring cells are connected by edges, then edges are converted to springs and cells (vertices in the graph) to masses. After that, pressurized chambers are formed from polygons delineated by the edges. Finally, the outmost edges form the "skin" of the animat. The locomotion of the animats is possible because each cell can contract or expand the springs that are connected to it, provoking changes of the areas of chambers around that cell. The pressurized chambers act as a "hydrostatic skeleton" of the animat. The animats move in a physical environment with stationary fluid and drag force acting on each edge of the "skin".

In the experimental setup explored thus far [9, 15], the fitness function does not promote directly any particular body shape or mode of locomotion, only the distance travelled by the centre of animat's mass during a fixed time. Under such conditions, various strategies of control of locomotion evolve, including undulation (Fig. 3, top), the use of a "tail" (Fig. 3, middle) and of symmetrical protrusions ("appendages"; Fig. 3, bottom). When the fluid drag is low, the use of "appendages" tends to evolve more commonly (when independent evolutionary runs are considered), while undulatory movement evolves more often when the fluid drag is high, and a "tail" strategy when it is moderate. All these swimming strategies rely on synchronous contraction and expansion of the springs, with phase shifts along the animat's front-back and/or left-right axes.

4 Towards Biologically-inspired Development of Artificial Neural Networks

A model of 3D patterning of complex shapes is a first step towards a biological-inspired approach to the generation of artificial neural networks with non-trivial behaviour. A platform implementing such a model could aid in the understating of how the environment shapes biological cognitive systems through evolution and development. This issue has been thus far investigated to only a limited extent, using models with apparently poor evolvability (e.g., [6]), lacking biological realism (e.g., [1], in which genes are limited in number and body subunits do not correspond to cells), or in which only the brain was evolved (e.g., [8]).

Our long-term plans include the introduction of developing and learning neural networks in GReaNs and investigation of co-evolution/co-development of multicellular brains and bodies. Before it is possible, our model will have to be extended to allow cell differentiation into neurons. We hope that this can be achieved efficiently in a manner similar to that used when investigating differentiation with coloured cells. In other words, when a product coded by a special element will cross a concentration threshold, the cell would become a neuron with specific properties (or a sensor, or actuator). This can be achieved in a manner similar to differentiation into coloured cells. The parameters of the particular neuron model would be then specified by the concentrations of special products in the cell (using, for example, 4-parameter exponential [2] or quartic [21] adaptive neurons; concentration of additional products might influence synaptic properties, including learning).

A larger challenge is to find an appropriate way to specify the connections between neurons. Our initial proposal will be to use the chemical affinity of special products in each cell (this is inspired by the role of membrane proteins in determining the connectivity in the nervous system) rather than to model the growth of neurites (explicitly or by setting the direction/length of the neurites).

Perhaps the most important and difficult part of this future work will be defining suitable targets for the evolutionary process. Evolution and development of brains for artificial bodies requires building a model of the interaction of the bodies with the physical world. A biologically-inspired approach to this, based on local interactions, modularity etc., is a challenge in itself. It may be advisable to avoid trying addressing several large challenges at once. We think that the question of building a model for specifying the biophysical parameters of the neurons, and of specifying the connectivity in the network, could be approached first using a simpler challenge, perhaps by evolving artificial neural networks able to perform signal processing tasks.

We have previously investigated both the issue of animat control and signal processing in GReaNs, but so far only at the level of single cells. In these experiments, multicellular development is missing—a GRN is evolved to directly control the movement of a unicellular organism in a gradient of chemical substances (chemotaxis; [12, 26]) or to process signals at the level of single cells [13].

In the experiments on uni-cellular chemotaxis, a cell is converted to a simple animat with sensors for chemical substances at the front and actuators which work

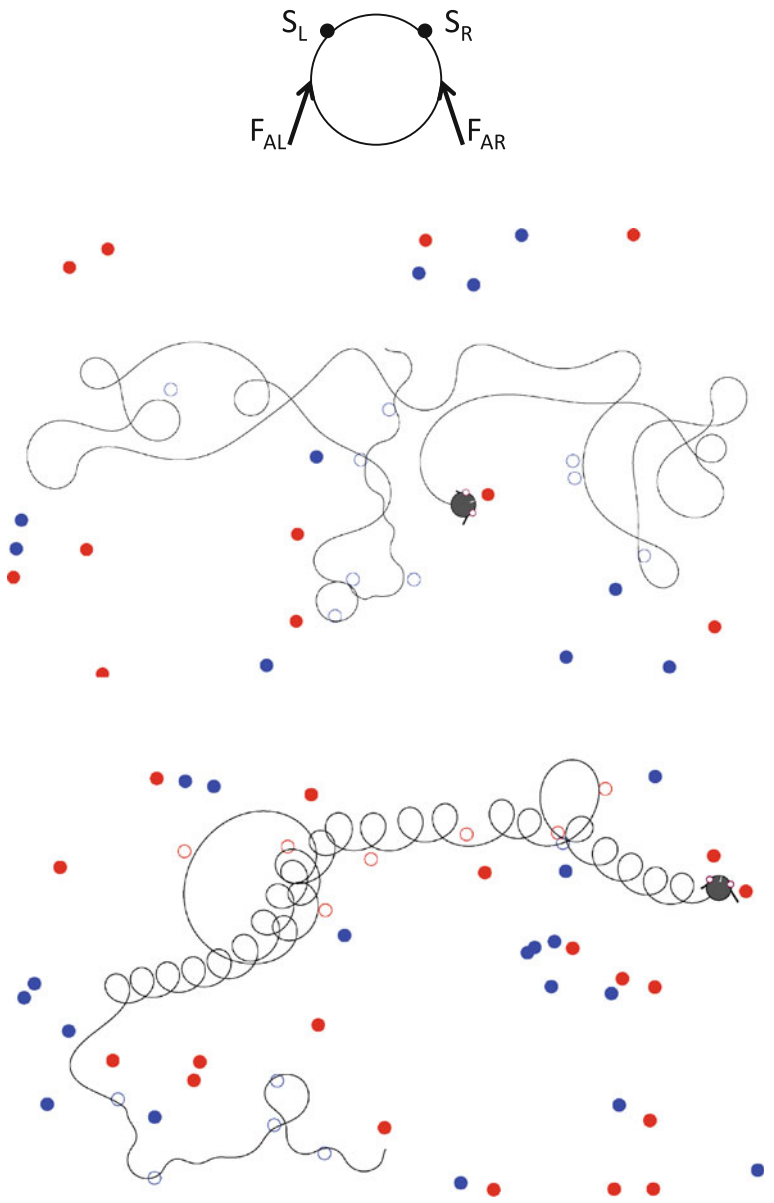

Fig. 4 Evolved behaviour of unicellular animats controlled by GRNs. An animat (*top*) has two sensors and two actuators (thrusters). The best evolved animat efficiently searches for one chemical ("food") and avoids the other ("poison") on a random map (*middle*). Initially, *blue* sources correspond to "food" and *red* to "poison". After 5 "'food" sources are reached (*empty circles*), the function of the substances changes. The fitness function rewarded animats that searched for "food" and avoided "poison" during both phases of the evaluation. The best individual obtained in another independent evolutionary run (*bottom*) searches efficiently for the *blue* sources but uses a suboptimal strategy for *red*

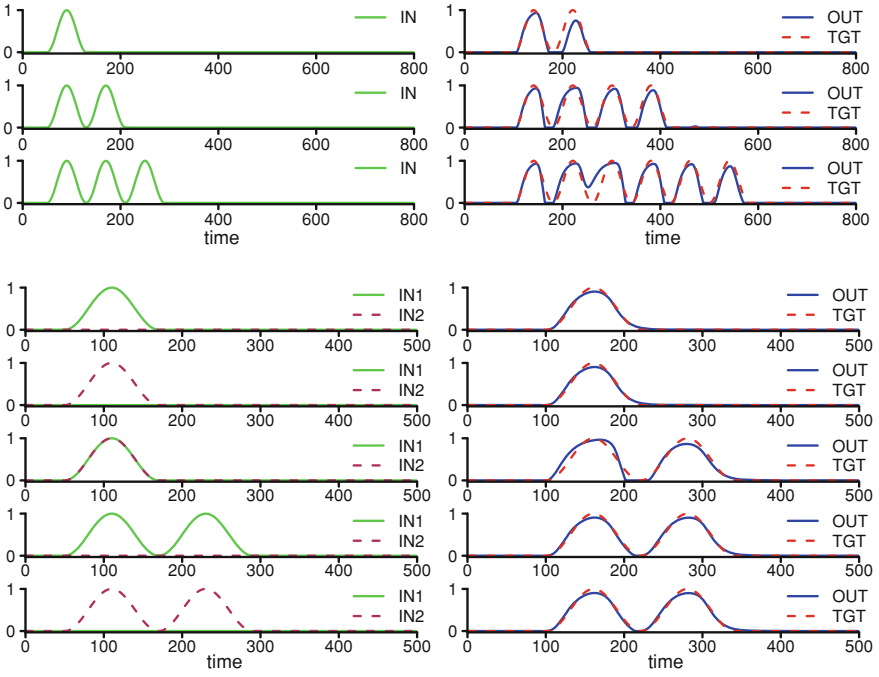

Fig. 5 Processing signals using evolving GRNs. *Top* three time courses (input-output pairs) of the best GRN evolved for multiplication of the spikes in the concentration of the input substance by two. *Bottom* the behaviour of a different GRN, evolved to add the spikes in two inputs, using five input-output pairs. The fitness function rewarded the proximity of the output to target (*dashed lines*) and changing concentration of the TF read as output

as thrusters at the back (Fig. 4 top). Animats are placed individually in a continuous 2D environment with randomly placed sources from which chemical substances diffuse. If the cell reaches the source, the source is removed together with all the substance that diffused from this source. The fitness function during the genetic algorithm rewards the individuals for reaching the sources, and thus collecting chemical resources. It is also possible to reward the animats for searching for one substance ("food") and avoiding the other ("poison"), and to investigate the evolution of various behaviours (which evolve because of the of the presence of suboptimal peaks in the fitness landscape) in experiments in which the role of the substances switches (Fig. 4 middle) or in experiments in which some resources have a provisional role, while others are important for reproduction (we have called this paradigm "the Search for Beauty" [26]).

Our experiments on the evolution of chemotaxis show that the regulatory networks in GReaNs are able to process simple sensory signals. But they can also be used for more complex signal processing [12]. A genetic algorithm can be used to obtain cells able to perform simple computational tasks, for example, multiplication by two (Fig. 5 top), or addition (Fig. 5 bottom). In such experiments, input to a cell is

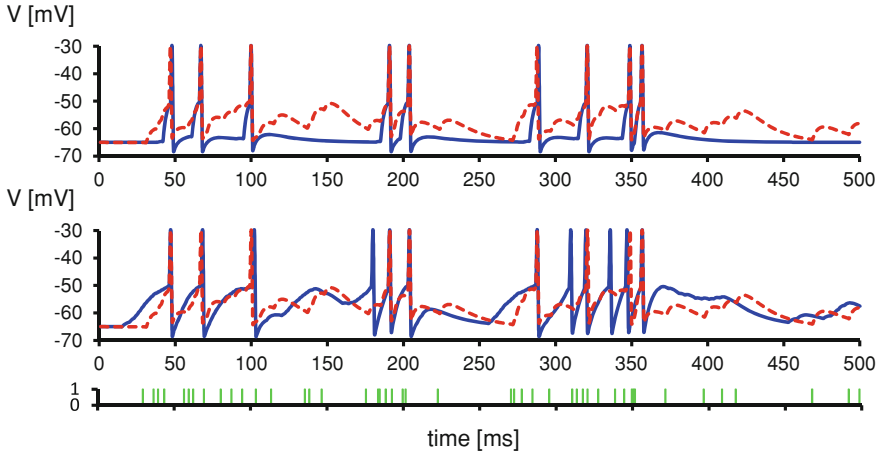

Fig. 6 Evolving spiking neural networks in GReaNs. A network of LIF neurons was evolved with GReaNs so that the spikes produced by the output neuron (*blue line*) match the spikes of 'one' AdEx neuron (*red dashed line*), shifted by 10 ms (*top*) or 20 ms (*middle*) in response to a specific Poisson spike train (*green*, *bottom*)

provided as an externally determined concentration of one or two substances, while the concentration of one of *products* is read as the output. We do not permit direct connections between the input(s) and the output.

Our most recent extension to GReaNs [25] is the introduction of spiking neural models—the leaky integrate and fire model with a fixed threshold (LIF; [5]), and the adaptive exponential model [2]. This is the one of the stepping stones towards evolving, developing, and learning neural networks in GReaNs, but at this stage it amounts to exchanging the equation governing the regulatory units (Eq. 1) for the equations appropriate for a given neuron model (LIF or AdEx). In our preliminary experiments, the network of such spiking computational units is evolved to approximate an arbitrary spike train or to process a specific input. An example of a latter task is to match the spike train produced by one neuron (again, either LIF or AdEx) as a response to a specific input (a Poission spike train) (Fig. 6). Since in these experiments we do not permit direct connections between input and output regulatory units, and because synaptic connections introduce delay, a perfect match is not possible unless the fitness function rewards a match with a shifted desired pattern. The task can be made more difficult by increasing the shift. Although it is possible in principle to solve this task using a feed-forward network, our preliminary results indicate that the evolved networks do not use this approach.

To sum up, it seems that we are now in a position to create such a multi-scale system using GReaNs. This system would include evolution, development, and behaviour, with the separation of these three time scales, and also a biologically-inspired model of the genome, of the developmental process, and of the neural network. On the road towards building this system, we plan to re-use parts of the existing software

(the simulation platform that allows for development and interaction with artificial physics, and the already implemented models of spiking neurons).

Of course, building such a platform always requires trading off some (or quite a lot) of the biological realism for computational efficiency—efficiency at the level of simulating cells, bodies, and the physical world, but also, and perhaps more importantly, in terms of the search space for the evolutionary process. In the platform we plan to create there would be, similarly to biological organisms, two levels of control using regulatory networks—each cell would be controlled by a GRN, while the behaviour of the whole animat would be controlled by the artificial neural network formed by these cells.

5 Summary

Our Artificial Life/Artificial Embryogenesis platform, GReaNs, provides a biologically-inspired approach towards the generation of multicellular bodies. The properties of these bodies emerge from the local interactions between cells, and the behaviour of the cells emerges from local molecular interactions at the level of genes. We have previously used this approach to evolve development of multicellular bodies with complex shapes and with simple cell differentiation. Recent extensions to GReaNs include (i) a procedure to convert the multicellular structures to soft-bodied animats, and (ii) two models of spiking neurons. We are now in a position to integrate various elements in GReaNs into a multi-scale system for evolution of development of multicellular bodies with brains which could process information and which could possibly control these bodies in a simulated physical environment.

Acknowledgments The work in BW's lab is supported by the Polish National Science Centre (Project 2011/03/B/ST6/00399), with computational resources provided by the Tri-city Academic Computer Centre (TASK) and the Interdisciplinary Centre for Molecular and Mathematical Modeling (ICM, University of Warsaw; Project G33-8). We are grateful to Rene Doursat, Taras Kowaliw and Volker Steuber for discussions, and to Ahmed Abdelmotaleb for technical assistance in preparing Fig. 6.

References

1. J.C. Bongard, R. Pfeifer, Evolving complete agents using artificial ontogeny, in *Morpho-functional Machines: The New Species*, ed. by F. Hara, R. Pfeifer (Springer, Japan, 2003), pp. 237–258
2. R. Brette, W. Gerstner, Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. J. Neurophysiol. **94**(5), 3637–3642 (2005)
3. A. Chavoya, I.R. Andalon-Garcia, C. Lopez-Martin, M.E. Meda-Campaña, Use of evolved artificial regulatory networks to simulate 3D cell differentiation. Biosystems **102**(1), 41–48 (2010)

4. S. Cussat-Blanc, H. Luga, Y. Duthen, From single cell to simple creature morphology and metabolism, in *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2008), pp. 134–141
5. P. Dayan, L.F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems* (The MIT Press, 1st edition, Cambridge, 2001)
6. F. Dellaert, R.D. Beer, A developmental model for the evolution of complete autonomous agents, in *From Animals to Animats 4: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB 1996)* (MIT Press, 1996), pp. 393–401
7. P. Eggenberger Hotz, Evolving morphologies of simulated 3D organisms based on differential gene expression, in *Proceedings of the 4th European Conference on Artificial Life (ECAL 1997)* (MIT Press, 1997), pp. 205–213
8. N. Jakobi, Harnessing morphogenesis, in *Proceedings of Information Processing in Cells and Tissues* (1995), pp. 29–41
9. M. Joachimczak, T. Kowaliw, R. Doursat, B. Wróbel, Brainless bodies: controlling the development and behavior of multicellular animats by gene regulation and diffusive signals, in *Artificial Life XIII: Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2012), pp. 349–356
10. M. Joachimczak, B. Wróbel, Evo-devo in silico: a model of a gene network regulating multicellular development in 3D space with artificial physics, in *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2008), pp. 297–304
11. M. Joachimczak, B. Wróbel, Complexity of the search space in a model of artificial evolution of gene regulatory networks controlling 3D multicellular morphogenesis. Adv. Complex Syst. **12**(03), 347–369 (2009)
12. M. Joachimczak, B. Wróbel, Evolving gene regulatory networks for real time control of foraging behaviours, in *Artificial Life XII: Proceedings of the 12th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2010), pp. 348–355
13. M. Joachimczak, B. Wróbel, Processing signals with evolving artificial gene regulatory networks, in *Artificial Life XII: Proceedings of the 12th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2010), pp. 203–210
14. M. Joachimczak, B. Wróbel, Evolution of the morphology and patterning of artificial embryos: scaling the tricolour problem to the third dimension, in *Advances in Artificial Life. Darwin Meets von Neumann: Proceedings of the 10th European Conference on Artificial Life (ECAL 2009)*, vol. 5777, Lecture Notes in Computer Science (Springer, 2011), pp. 35–43
15. M. Joachimczak, B. Wróbel, Co-evolution of morphology and control of soft-bodied multicellular animats, in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation, GECCO '12* (ACM, 2012), pp. 561–568
16. M. Joachimczak, B. Wróbel, Evolution of robustness to damage in artificial 3-dimensional development. Biosystems **109**(3), 498–505 (2012)
17. M. Joachimczak, B. Wróbel, Open ended evolution of 3D multicellular development controlled by gene regulatory networks, in *Artificial Life XIII: Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 2012), pp. 67–74
18. J.F. Knabe, C.L. Nehaniv, M.J. Schilstra, Evolution and morphogenesis of differentiated multicellular organisms: autonomously generated diffusion gradients for positional information, in *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems* (MIT Press, 20080, pp. 321–328
19. S. Kumar, P.J. Bentley, Biologically inspired evolutionary development, in *Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, vol. 2606, Lecture Notes in Computer Science (Springer, 2003), pp. 57–68
20. L. Schramm, B. Sendhoff, An animat's cell doctrine, in *ECAL 2011: Proceedings of the 11th European Conference on the Synthesis and Simulation of Living Systems* (MIT Press, 2011), pp. 739–746
21. J. Touboul, Bifurcation analysis of a general class of nonlinear integrate-and-fire neurons. SIAM J. Appl. Math **68**(4), 1045–1079 (2008)

22. A. Wagner, Robustness and evolvability: a paradox resolved. Proc. R. Soc. B: Biol. Sci. **275**(1630), 91–100 (2008)
23. M.J. West-Eberhard, *Developmental Plasticity and Evolution* (Oxford University Press, 1st edition, USA, 2003)
24. L. Wolpert, The French Flag problem: a contribution to the discussion on pattern development and regulation, in *The Origin of Life: Toward a Theoretical Biology* ed. by C.H. Waddington (Edinburgh University Press, Edinburgh, 1968), pp. 125–133
25. B. Wróbel, A. Abdelmotaleb, M. Joachimczak, Evolving spiking neural networks in the GReaNs (gene regulatory evolving artificial networks) platform, in *EvoNet2012: Evolving Networks, from Systems/Synthetic Biology to Computational Neuroscience Workshop at Artificial Life XIII* (2012) pp. 19–22
26. B. Wróbel, M. Joachimczak, A. Montebelli, R. Lowe, The search for beauty: evolution of minimal cognition in an animat controlled by a gene regulatory network and powered by a metabolic system, vol. 7426 (Springer, Berlin Heidelberg, 2012), pp. 198–208

Chapter 7
Constructing Complex Systems Via Activity-Driven Unsupervised Hebbian Self-Organization

James A. Bednar

Abstract How can an information processing system as complex and as powerful as the human cerebral cortex be constructed from the limited information available in the genome? Answering this scientific question has the potential to revolutionize how computing systems for manipulating real-world data are designed and built. Based on an extensive array of physiological, anatomical, and imaging data from the primary visual cortex (V1) of mammals, we propose a relatively simple biologically based developmental architecture that accounts for most of the demonstrated functional properties of V1 neurons. Given the overall similarity between cortical regions, and the absence of V1-specific circuitry in the model architecture, we expect similar principles to apply throughout the cerebral cortex. The architecture consists of a network of simple artificial V1 neurons with initially unspecific connections that are modified by Hebbian learning and homeostatic plasticity, driven by input patterns from other neural regions and ultimately from the external world. Through an unsupervised developmental process, the model neurons begin to display the major known functional properties of V1 neurons, including receptive fields and topographic maps selective for all of the major low-level visual feature dimensions, realistic specific lateral connectivity underlying surround modulation and adaptation such as visual aftereffects, realistic behavior with visual contrast, and realistic temporal responses. In each case these relatively complex properties emerge from interactions between simple neurons and between internal and external drivers for neural activity, without any requirement for supervised learning, top-down feedback or reinforcement, neuromodulation, or spike-timing dependent plasticity. The model also unifies explanations of a wide variety of phenomena previously considered distinct, with the same adaptation mechanisms leading to both long-term development and short-term plasticity (aftereffects), the same subcortical lateral interactions providing both gain control and accounting for the time course of neural responses,

J. A. Bednar (✉)
Institute for Adaptive and Neural Computation, The University of Edinburgh,
10 Crichton St, Edinburgh EH8 9AB, UK
e-mail: jbednar@inf.ed.ac.uk

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_7,

and the same cortical lateral interactions leading to complex cell properties, map formation, and surround modulation. This relatively simple architecture thus sets a baseline for explanations of neural function, suggesting that most of the development and function of V1 can be understood as unsupervised learning, and setting the stage for demonstrating the additional effects of higher- or lower-level mechanisms. The architecture also represents a simple, scalable approach for specifying complex data-processing systems in general.

1 Introduction

Current technologies for building complex information processing machines, particularly for dealing with continuous and noisy real-world data, remain far behind those of animals and particularly humans. Artificial neural networks originally inspired by biological nervous systems are in wide use for some tasks, and indeed have been shown to be able to perform any Turing-computable function in theory [48]. However, actually specifying and constructing a network capable of performing a particular complex task remains an open problem. With this in mind, it is important to study how such networks are specified and developed in animals and humans, to give us clues for how to build similarly powerful artificial systems. Understanding how systems as complex as the human brain can be built has the potential to revolutionize how computing systems for manipulating real-world data are designed and constructed.

The cerebral cortex of mammals is a natural starting point for study, since the cortex is the largest part of the human brain and serves a wide variety of sensory and motor functions in mammals, yet has a relatively uniform structure. The cortical surface can be divided into anatomically distinguishable cortical areas, of which there are dozens in humans, but perhaps the most commonly studied is the primary visual cortex (V1). After processing by circuitry in the retina, visual information travels from the Retinal Ganglion Cells (RGCs) of the eye to the lateral geniculate nucleus (LGN) of the thalamus, and from the thalamus goes directly to cells in V1. This simple, direct pathway, along with the unparallelled ease with which visual patterns can be controlled in the laboratory, means that V1 provides a unique opportunity for running well-controlled experiments for determining neural function.

Like the rest of the cerebral cortex, V1 has a laminar and columnar structure, i.e., it is composed of multiple thin layers (typically numbered 1–6) parallel to the cortical surface, with neurons at corresponding locations in each layer forming "columns" that have similar functional properties, while more distant columns tend to differ in their properties. This predominantly two-dimensional functional organization is often measured using experimental imaging techniques that record activity across the cortical surface in response to a visual image, which obscures any differences in neural activity across the layers but does provide a useful large-scale measure of neural organization. Complementary information about visual responses of individual neurons or small groups (typically measured as firing rates, i.e., spikes per

second) can be measured using microelectrodes placed nearby, providing detailed temporal responses but necessarily sampling from only a few neurons.

Studies using these techniques in monkeys, cats, ferrets, and tree shrews over the past half century have established a wide range of properties of V1 neurons that relate to their function in visual processing (reviewed in [7, 31]):

1. V1 neurons respond selectively, in terms of their average firing rate, to specific low-level visual features such as the position, orientation, eye of origin, motion direction, spatial frequency, interocular disparity, or color of a small patch of an image.
2. V1 neurons in most laboratory-animal species are organized into smooth topographic maps for some or all of these visual features, with specific patterns of feature preference variation (e.g. in orientation preference) across the cortical surface, and specific interactions between these maps.
3. V1 neurons in these maps are laterally connected with connection strengths and probabilities that reflect their selectivities (e.g. with stronger connections between neurons preferring similar orientations).
4. Due in part to these lateral connections, V1 neuronal responses depend on activities of both neighboring and more distant V1 neurons, yielding complex but systematic visual surround modulation effects.
5. V1 neurons exhibit contrast-invariant tuning for the features for which they are selective, such that selectivity is preserved even for strong input patterns. This property rules out most simple (linear) models of visual feature selectivity.
6. Many V1 neurons have complex spatial pattern preferences that cannot be characterized using a simple template of their preferred pattern, e.g. responding to similar patterns with some tolerance to the exact retinal position of the pattern.
7. Response properties of V1 neurons exhibit long-term and short-term plasticity and adaptation, measurable psychophysically as visual aftereffects, which suggests ongoing dynamic regulation of responses.
8. V1 neuron responses to changes in visual inputs exhibit a stereotyped temporal pattern, with transiently high responses at pattern onset and offset and a lower sustained response, which biases neural responses towards non-static stimuli.

These properties suggest a wide range of specific roles for V1 neurons in visual processing, and explaining how V1 neurons come to behave in this way would be a significant step towards understanding the cerebral cortex in general.

One way to account for the above V1 properties would be to build specific mathematical models for each property or a small set of them, and indeed many such models have been devised to account for aspects of V1 visual processing in adult animals. However, such an approach is unlikely to lead to a general explanation for cortical function, both because such models cannot easily be combined into a full visual processing system, and also because the models are specifically tailored by the modeller to account for that particular function, and thus do not generalize to arbitrary data processing tasks.

This chapter outlines and reviews results from an alternative "system building" approach, focusing on providing a general domain-independent explanation for how

all of the above properties of V1 could arise from a biologically plausible and initially unspecific cortical circuit. Specifically, my colleagues and I have developed closely interrelated models of V1 accounting for each property using only a small set of plausible principles and mechanisms, within a consistent biologically grounded framework:

1. Single-compartment (point neuron) firing-rate (i.e., non-spiking) retinal ganglion cell, lateral geniculate nucleus, and V1 model neurons (see Fig. 1),
2. Hardwired subcortical pathways to V1, including the main LGN or RGC cell types that have been identified,
3. Initially roughly retinotopic topographic projections from the eye to the LGN and from the LGN to V1, connecting corresponding areas of each region,
4. Initially roughly isotropic (i.e., radially uniform) local connectivity to and between neurons in layers in V1, connecting neurons non-specifically to their local and more distant neighbors,
5. Natural images and spontaneous subcortical input activity patterns that lead to V1 responses,
6. Hebbian (unsupervised activity-dependent) learning with normalization for synapses on V1 neurons,
7. Homeostatic plasticity (whole-cell adaptation of excitability to keep the average activity of each V1 neuron constant), and
8. Various modeller-determined parameters associated with each of these mechanisms, eventually intended to be set through self-regulating mechanisms.

Properties and mechanisms not necessary to explain the phenomena listed above, such as spiking, spike-timing dependent plasticity, detailed neuronal morphology, feedback from higher areas, neuromodulation, reinforcement learning, and supervised learning have been omitted, to clearly focus on the aspects of the system most relevant to those phenomena. The overall hypothesis is that much of the complex structure and properties observed in V1 emerges from interactions between relatively simple but highly interconnected computing elements, with connection strengths and patterns self-organizing in response to visual input and other sources of neural activity. Through visual experience, the geometry and statistical regularities of the visual world become encoded into the structure and connectivity of the visual cortex, leading to a complex functional cortical architecture that reflects the physical and statistical properties of the visual world.

At present, many of the results have been obtained independently in a wide variety of separate projects performed with different collaborators at different times. However, all of the models share the same underlying principles outlined above, and all are implemented using the same simulator and a small number of underlying components. See [7] for an overview of each of the different models and how they fit together; here we focus on two representative models that that account for the bulk of the above properties. First, we present a simple example of a single-V1-layer GCAL (gain-control adaptive laterally connected) model of the development of orientation preferences and orientation maps for a single eye (Fig. 1). Second, we present some results for a larger model that includes motion direction and ocular dominance as

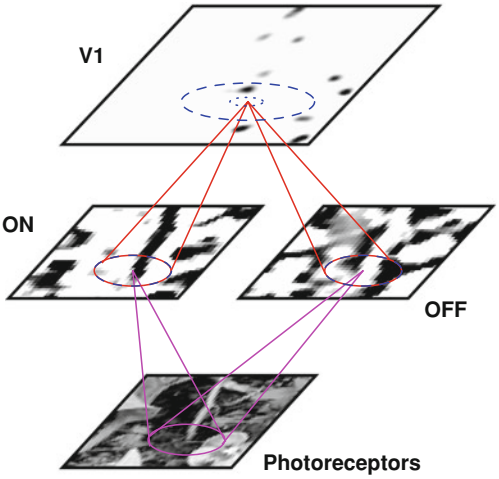

Fig. 1 *Basic GCAL model architecture*. In the simplest case, GCAL consists of a *greyscale* matrix representing the photoreceptor input, a pair of neural sheets representing the ON-center and OFF-center pathways from the photoreceptors to V1, and a single sheet representing V1. Each sheet is drawn here with a sample activity pattern resulting from one natural image patch. Each projection between sheets is illustrated with an oval showing the extent of the connection field in that projection, with lines converging on the target of the projection. Lateral projections, connecting neurons within each sheet, are marked with *dashed* ovals. Projections from the photoreceptors to the ON and OFF sheets, and within those sheets, are hardwired to mimic a specific class of response types found in the retina and LGN, in this case monochromatic center-surround neurons with a fixed spatial extent. Connections to and between V1 neurons adapt via Hebbian learning, allowing initially unselective V1 neurons to exhibit the range of response types seen experimentally, by differentially weighting each of the subcortical inputs (from the ON and OFF sheets) and inputs from neighboring V1 neurons

well (Fig. 2). Results for color, disparity, spatial frequency, complex cells, and surround modulation require still larger models not discussed here, but implemented using similar principles [3, 5, 7, 38, 40]. The goal for each of these models is the same—to explain how a cortical network can start from an initially undifferentiated state, to wire itself into a collection of neurons that behave, at a first approximation, like those in V1. Because such a model starts with no specializations (at the cortical level) specific to vision and would organize very differently when given different inputs, it also represents a general explanation for the development and function of any sensory or motor area in the cortex.

2 Architecture

All of the models whose results are presented here are implemented in the Topographica simulator, and are freely available along with the simulator from www.topographica.org. Both the basic and large models are described using the same

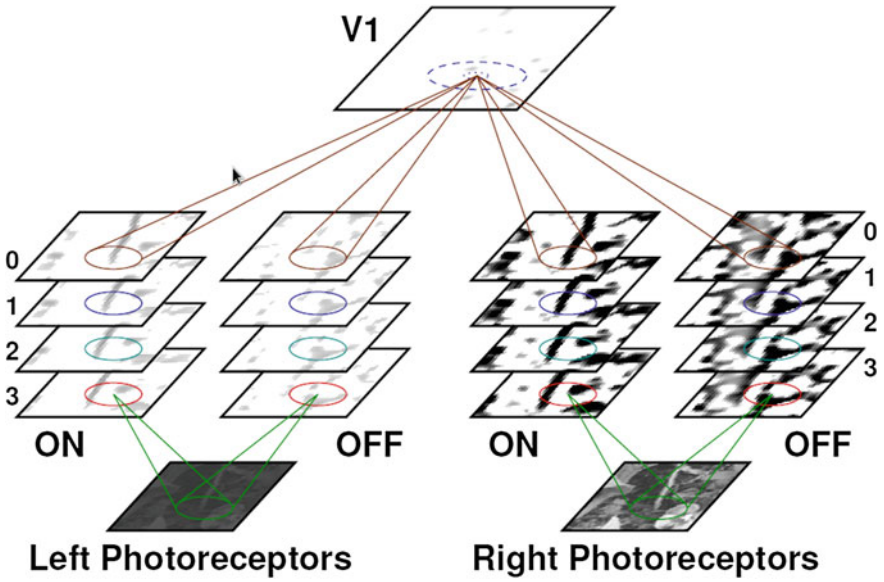

Fig. 2 *Larger GCAL model architecture*. Single-V1-sheet GCAL model for orientation, ocular dominance, and motion direction. This model consists of 19 neural sheets and 50 separate projections between them. Again, each sheet is drawn with a sample activity pattern resulting from one natural image, and all sheets below V1 are hardwired to cover a range of response types found in the retina and LGN. In this model, V1 neurons can become selective along many possible dimensions by various weightings for the incoming activity patterns. Other GCAL models add additional subcortical inputs (e.g. for color cone types) or additional populations and layers in V1. Reprinted from [13]

equations shown below, previously presented in Refs. [7, 29] but here extended to include temporal calibration from the TCAL model [51]. The model is intended to represent the visual system of the macaque monkey, but relies on data from studies of cats, ferrets, tree shrews, or other mammalian species where clear results are not yet available from monkeys.

2.1 Sheets and Projections

Each Topographica model consists of a set of sheets of neurons and projections (sets of topographically mapped connections) between them. A model has sheets representing the visual input (as a set of activations in photoreceptor cells), sheets implementing the transformation from the photoreceptors to inputs driving V1 (expressed as a set of ON and OFF LGN cell activations), and sheets representing neurons in V1. The simple GCAL model (Fig. 1) has 4 such sheets, the larger one (Fig. 2) has 19, and the complete unified model described in [7] has 29, each representing different topographically organized populations of cells in a particular region.

Each sheet is implemented as a two-dimensional array of firing-rate neurons. The Topographica simulator allows parameters for sheets and projections to be specified in measurement units that are independent of the specific grid sizes used in a particular run of the simulation. To achieve this, Topographica sheets provide multiple spatial coordinate systems, called *sheet* and *matrix* coordinates. Where possible, parameters (e.g. sheet dimensions or connection radii) are expressed in sheet coordinates, expressed as if the sheet were a continuous neural field rather than a finite grid. In practice, of course, sheets are always simulated using some finite matrix of units. Each sheet has a parameter called its density, which specifies how many units (matrix elements) in the matrix correspond to a length of 1.0 in continuous sheet coordinates, which allows conversion between sheet and matrix coordinates. When sizes are scaled appropriately [8], results are independent of the density used, except at very low densities or for simulations with complex cells, where complexity increases with density [4]. Larger areas can be simulated easily [8], but require more memory and simulation time.

A projection to an $m \times m$ sheet of neurons consists of m^2 separate *connection fields*, one per target neuron, each of which is a spatially localized set of connections from the neurons in one input sheet that are near the location corresponding topographically to the target neuron. Figures 1 and 2 show one sample connection field (CF) for each projection, visualized as an oval of the corresponding radius on the input sheet (drawn to scale), connected by a cone to the neuron on the target sheet (if different). The connections and their weights determine the specific properties of each neuron in the network, by differentially weighting inputs from neurons of different types and/or spatial locations. Each of the specific types of sheets and projections is described in the following sections.

2.2 Images and Photoreceptor Sheets

The basic GCAL model (Fig. 1) has one input sheet, representing responses of photoreceptors of one cone class in one retina, while the larger GCAL model considered here has two, adding an additional set from another eye (Fig. 2). The full unified GCAL model of all the input dimensions includes six input sheets (three different cone types in each eye; not shown or analyzed further here). For the larger model, input image pairs (left, right) were generated by choosing one image randomly from a database of single calibrated images, selecting a random patch within the image, a random direction of motion translation with a fixed speed (described in Ref. [10]), and a random brightness difference between the two eyes (described in Ref. [31]). These modifications are intended as a simple model of motion and eye differences, to allow development of direction preference, ocular dominance, and disparity maps, until suitable full-motion stereo calibrated-color video datasets of natural scenes are available. Simulated retinal waves can also be used as inputs, to provide initial receptive-field and map structure before eye opening, but are not required for receptive-field or map development in the model [11].

2.3 Subcortical Sheets

The subcortical pathway from the photoreceptors to the LGN and then to V1 is represented as a set of hardwired subcortical cells with fixed connection fields (CFs) that determine the response properties of each cell. These cells represent the complete processing pathway to V1, including circuitry in the retina (including the retinal ganglion cells), the optic nerve, the lateral geniculate nucleus, and the optic radiations to V1. Because the focus of the model is to explain cortical development given its thalamic input, the properties of these RGC/LGN cells are kept fixed throughout development, for simplicity and clarity of analysis.

Each distinct RGC/LGN cell type is grouped into a separate sheet, each of which contains a topographically organized set of cells with identical properties but responding to a different topographically mapped region of the retinal photoreceptor input sheet. Figure 1 shows the two main different spatial response types used in the GCAL models illustrated here, ON (with an excitatory center) and OFF (with an excitatory surround). All of these cells have Difference-of-Gaussian (DoG) receptive fields, and thus perform edge enhancement at a particular size scale. Additional cell classes can easily be added as needed for spatial frequency (with multiple DoG sizes) or color (with separate cone types for the center and surround Gaussians) simulations. Figure 2 shows additional ON and OFF cell classes with different delays, added to allow development of motion preferences.

For the ON and OFF cells in the larger model, there are multiple copies with different delays from the retina. These delays represent the different latencies in the lagged versus non-lagged cells found in cat LGN [44, 59], and allow V1 neurons to become selective for the direction of motion. Many other sources of temporal delays would also lead to direction preferences, but have not been tested specifically.

2.4 Cortical Sheets

The simulations reported in this chapter use only a single V1 sheet for simplicity, but in the full unified model, V1 is represented by multiple cortical sheets representing different cell types and different V1 layers [3, 7]. In this simplified version, cells make both excitatory and inhibitory connections (unlike actual V1 neurons), and all cells receive direct input from LGN cells (unlike many V1 neurons). Even so, the single-sheet V1 can demonstrate most of the phenomena described above, except for complex cells (which can be obtained by adding a separate population of cells without direct thalamic input [4]) and contrast-dependent surround modulation effects (which require separate populations of inhibitory and excitatory cells [3]).

The behavior of the cortical sheet is primarily determined by the projections to and within it. Each of these projections is initially non-specific other than the initial rough topography, and becomes selective only through the process of self-organization (described below), which increases some connection weights at the expense of others.

2.5 Activation

The model is simulated in a series of discrete time steps with step size $\delta t = 0.05$ (roughly corresponding to 12.5 ms of real time). At time 0.0, the first image is drawn on the retina, and the activation of each unit in each sheet is updated for the remaining 19 steps before time 1.0, when a new pattern is selected and drawn on the retina (and similarly until the last input pattern is drawn at time 10,000). Each image patch on the retina represents one visual fixation (for natural images) or a snapshot of the relatively slowly changing spatial pattern of spontaneous activity (such as the well-documented retinal waves [60]). Thus the training process consists of a constant retinal activation, followed by recurrent processing at the LGN and cortical levels. For one input pattern, assume that the input is drawn on the photoreceptors at time t and the connection delay (constant for all projections) is defined as 0.05. Then at $t + 0.05$ the ON and OFF cells compute their responses, and at $t + 0.010$ the thalamic output is delivered to V1, where it similarly propagates recurrently through the intracortical projections to the cortical sheet(s) every 0.05 time steps. As described in Sect. 3.4, a much smaller step size of $\delta t = 0.002$ allows replication of the detailed time course of responses to individual patterns, but this relatively coarse step size of 0.05 is more practical for simulations of long-term processes like neural development.

Images are presented to the model by activating the retinal photoreceptor units. The activation value $\Psi_{i,P}$ of unit i in photoreceptor sheet P is given by the brightness of that pixel in the training image.

For each model neuron in the other sheets, the activation value is computed based on the combined *activity contributions* to that neuron from each of the sheet's incoming projections. The activity contribution from a projection is recalculated whenever its input sheet activity changes, after the corresponding connection delay. For each unit j in a target sheet and an incoming projection p from sheet s_p, the activity contribution is computed from activations in the corresponding connection field F_{jp}. F_{jp} consists of the local neighborhood around j (for lateral connections), or the local neighborhood of the topographically mapped location of j on s_p (for a projection from another sheet); see examples in Figs. 1 and 2. The activity contribution C_{jp} to j from projection p is then a dot product of the relevant input with the weights in each connection field:

$$C_{jp}(t + \delta t) = \sum_{i \in F_{jp}} \eta_i(t)\omega_{ij,p} \tag{1}$$

where unit i is taken from the connection field F_{jp} of unit j, $\eta_i(t)$ is the activation of unit i, and $\omega_{ij,p}$ is the connection weight from i to j in that projection. Across all projections, multiple direct connections between the same pair of neurons are possible, but each projection p contains at most one connection between i and j, denoted by $\omega_{ij,p}$.

For a given cortical unit j, the activity $\eta_j(t + \delta t)$ is calculated from a rectified weighted sum of the activity contributions $C_{jp}(t + \delta t)$:

$$\eta_{jV}(t+\delta t) = \lambda f\left(\sum_p \gamma_p C_{jp}(t+\delta t)\right) + (1-\lambda)\eta_{jV}(t) \tag{2}$$

where f is a half-wave rectifying function with a variable threshold point (θ) dependent on the average activity of the unit, as described in the next subsection, and V denotes one of the cortical sheets. λ is a time-constant parameter that defines the strength of smoothing of the recurrent dynamics in the network, chosen to match the transient behaviour of V1 neurons; here $\lambda = 1$ throughout except that $\lambda = 0.01$ for the simulations of the detailed time course of responses (Sect. 3.4).

Each γ_p is an arbitrary multiplier for the overall strength of connections in projection p. The γ_p values are set in the approximate range 0.5–3.0 for excitatory projections and -0.5 to -3.0 for inhibitory projections. For afferent connections, the γ_p value is chosen to map average V1 activation levels into the range 0–1.0 by convention, for ease of interconnecting and analyzing multiple sheets. For lateral and feedback connections, the γ_p values are then chosen to provide a balance between feedforward, lateral, and feedback drive, and between excitation and inhibition; these parameters are crucial for making the network operate in a useful regime.

RGC/LGN neuron activity is computed similarly to Eq. 2, except to add divisive normalization and to fix the threshold θ at zero:

$$\eta_{jL}(t+\delta t) = \lambda f\left(\frac{\sum_p \gamma_p C_{jp}(t+\delta t)}{\gamma_S C_{jS}(t+\delta t) + k}\right) + (1-\lambda)\eta_{jL}(t) \tag{3}$$

where L stands for one of the RGC/LGN sheets. Projection S here consists of a set of isotropic Gaussian-shaped lateral inhibitory connections (see Eq. 6, evaluated with $u = 1$), and p ranges over all the other projections to that sheet. k is a small constant to make the output well-defined for weak inputs. The divisive inhibition implements the contrast gain control mechanisms found in RGC and LGN neurons [2, 3, 17, 23]. Here again $\lambda = 1$ throughout except that $\lambda = 0.03$ for the detailed simulations in Sect. 3.4.

Each time the activity is computed using Eq. 2 or 3, the new activity values are sent to each of the outgoing projections, where they arrive after the projection delay. The process of activity computation then begins again, with a new contribution C computed as in Eq. 1, leading to new activation values by Eq. 2 or 3. Activity thus spreads recurrently throughout the network, and can change, die out, or be strengthened, depending on the parameters.

With typical parameters that lead to realistic topographic map patterns, initially blurry patterns of afferent-driven cortical activity are sharpened into well-defined "activity bubbles" through locally cooperative and more distantly competitive lateral interactions [31]. Nearby neurons are thus influenced to respond more similarly, while more distant neurons receive net inhibition and thus learn to respond to different input patterns. The competitive interactions "sparsify" the cortical response into

patches, in a process that can be compared to the explicit sparseness constraints in non-mechanistic models [26, 36], while the local facilitatory interactions encourage spatial locality so that smooth topographic maps will be developed.

As described in more detail below, the initially random weights to cortical neurons are updated in response to each input pattern, via Hebbian learning. Because the settling (sparsification) process typically leaves only small patches of the cortical neurons responding strongly, those neurons will be the ones that learn the current input pattern, while other nearby neurons will learn other input patterns, eventually covering the complete range of typical input variation. Overall, through a combination of the network dynamics that achieve sparsification along with local similarity, plus homeostatic adaptation that keeps neurons operating in a useful regime, plus Hebbian learning that leads to feature preferences, the network will learn smooth, topographic maps with good coverage of the space of input patterns, thereby developing into a functioning system for processing patterns of visual input without explicit specification or top-down control of this process.

2.6 Homeostatic Adaptation

For this model, the threshold for activation of each cortical neuron is a very important quantity, because it directly determines how much the neuron will fire in response to a given input. Mammalian neurons appear to regulate such thresholds automatically, a process known as homeostatic plasticity or homeostatic adaptation [54] (where homeostatic means to keep similar and stable). To set the threshold automatically, each neural unit j in V1 calculates a smoothed exponential average of its own activity ($\overline{\eta_j}$):

$$\overline{\eta_j}(t) = (1 - \beta)\eta_j(t) + \beta\overline{\eta_j}(t - 1) \tag{4}$$

The smoothing parameter ($\beta = 0.999$) determines the degree of smoothing in the calculation of the average. $\overline{\eta_j}$ is initialized to the target average V1 unit activity (μ), which for all simulations is $\overline{\eta_{jA}}(0) = \mu = 0.024$. The threshold is updated as follows:

$$\theta(t) = \theta(t - 1) + \kappa(\overline{\eta_j}(t) - \mu) \tag{5}$$

where $\kappa = 0.0001$ is the homeostatic learning rate. The effect of this scaling mechanism is to bring the average activity of each V1 unit closer to the specified target. If the average activity of a V1 unit moves away from the target during training, the threshold for activation is thus automatically raised or lowered in order to bring it closer to the target.

2.7 Learning

Initial connection field weights are random within a two-dimensional Gaussian envelope. E.g., for a postsynaptic (target) neuron j located at sheet coordinate (0, 0), the weight $\omega_{ij,p}$ from presynaptic unit i in projection p is:

$$\omega_{ij,p} = \frac{1}{Z_{\omega p}} u \exp\left(-\frac{x^2 + y^2}{2\sigma_p^2}\right) \tag{6}$$

where (x, y) is the sheet-coordinate location of the presynaptic neuron i, u is a scalar value drawn from a uniform random distribution for the afferent and lateral inhibitory projections ($p = A, I$), σ_p determines the width of the Gaussian in sheet coordinates, and Z_ω is a constant normalizing term that ensures that the total of all weights $\omega_{ij,p}$ to neuron j in projection p is 1.0, where all afferent projections are treated together as a single projection so that their sum total is 1.0. Weights for each projection are only defined within a specific maximum circular radius r_p; they are considered zero outside that radius.

Once per input pattern (after activity has settled), each connection weight ω_{ij} from unit i to unit j is adjusted using a simple Hebbian learning rule. (Learning could instead be performed at every simulation time step, but doing so would require significantly more computation time). This rule results in connections that reflect correlations between the presynaptic activity and the postsynaptic response. Hebbian connection weight adjustment for unit j is dependent on the presynaptic activity η_i, the post-synaptic response η_j, and the Hebbian learning rate α:

$$\omega_{ij,p}(t) = \frac{\omega_{ij,p}(t-1) + \alpha \eta_j \eta_i}{\sum_k \left(\omega_{kj,p}(t-1) + \alpha \eta_j \eta_k\right)} \tag{7}$$

Unless it is constrained, Hebbian learning will lead to ever-increasing (and thus unstable) values of the weights. The weights are constrained using divisive post-synaptic weight normalization (denominator of Eq. 7), which is a simple and well understood mechanism. All afferent connection weights from RGC/LGN sheets are normalized together in the model, which allows V1 neurons to become selective for any subset of the RGC/LGN inputs. Weights are normalized separately for each of the other projections, to ensure that Hebbian learning does not disrupt the balance between feedforward drive, lateral and feedback excitation, and lateral and feedback inhibition. Subtractive normalization with upper and lower bounds could be used instead, but it would lead to binary weights [32, 33], which is not desirable for a firing-rate model whose connections represent averages over multiple physical connections. More biologically motivated homeostatic mechanisms for normalization such as multiplicative synaptic scaling [54] or a sliding threshold for plasticity [15] could be implemented instead, but these have not been tested so far.

Note that some of the results below use the earlier LISSOM model [31], which follows the same equations but lacks gain control and homeostatic adaptation (equivalent

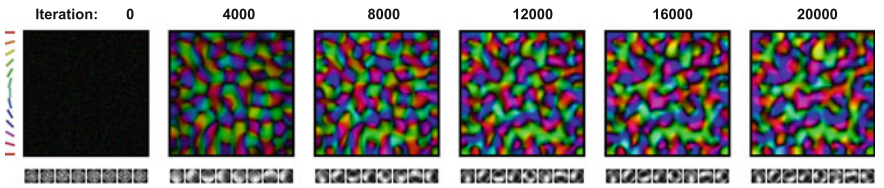

Fig. 3 *Development of maps and afferent connections*. Over the course of 20,000 input presentations, GCAL model V1 neurons develop selectivity for typical features of the input patterns. Here simulated retinal waves were presented for the first 6,000 inputs (modelling prenatal development), and monochromatic images of natural scenes were presented for the remainder (modelling postnatal visual experience). Connection fields to V1 neurons were initially random and isotropic (*bottom* of Iteration 0; CFs for 8 sample neurons are shown). Neurons were initially unselective, responding approximately equally to all orientations, and are thus *black* in the orientation map plot (where saturated colors represent orientation-selective neurons whose preference is labeled with the *color* indicated in the key). Over time, neurons develop specific afferent connection fields (*bottom* of remaining iterations) that cause neurons to respond to specific orientations. Nearby neurons respond to similar norientations, as in animal maps, and as a whole they eventually represent the full range of orientations present in the inputs. Reprinted from [29]

to setting $\gamma_S = 0$ and $k = 1$ in Eq. 3 and $\kappa = 0$ in Eq. 5). Without these automatic mechanisms, LISSOM requires the modeller to set the input strength and activation thresholds separately for each dataset and to adjust them as learning progresses. As long as these values have been set appropriately, previous LISSOM results can be treated equivalently to GCAL results, but GCAL is significantly simpler to use and describe, while being more robust to changes in the input distributions [29], so only GCAL is described here.

3 Results

The following sections outline a series of model results that account for anatomical, electrophysiology, imaging, psychophysical, and behavioral results from studies of experimental animals, all arising from the neural architecture and self-organizing mechanisms outlined in the previous section.

3.1 Maps and Connection Patterns

Figure 3 shows how orientation selectivity emerges in the basic GCAL model from Fig. 1, whose subcortical pathway consists of a single set of non-lagged monochromatic ON and OFF LGN inputs for a single eye. Over the course of development, initially unspecific connections become selective for specific patterns of LGN activity, including particular orientations. Hebbian learning ensures that each afferent

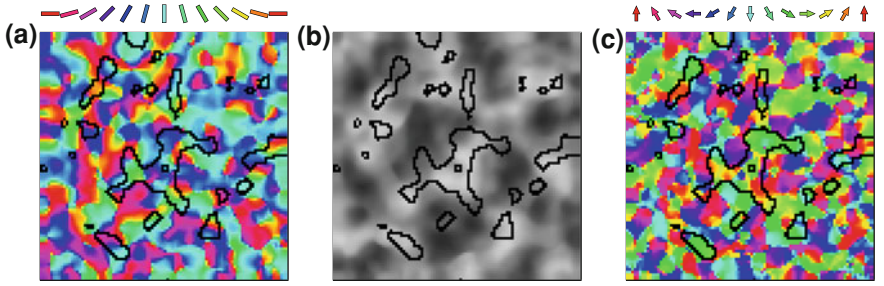

Fig. 4 *Lateral connections across maps*. GCAL and LISSOM model neurons each participate in multiple functional maps, but have only a single set of lateral connections. Connections are strongest from other neurons with similar properties, respecting each of the maps to the degree to which that map affects correlation between neurons. Maps for a combined orientation (OR), ocular dominance (OD), direction (DR) simulation LISSOM model are shown above, with the *black* outlines indicating the connections to the central neuron (marked with a small *square black* outline) that remain after weak connections have been pruned. Model neurons connect to other model neurons with similar orientation preference (**a**) (as in tree shrew, [18]) but even more strongly respect the direction map (**c**). This highly monocular unit also connects strongly to the same eye (**b**), but the more typical binocular cells have wider connection distributions. Reprinted from Ref. [13]

connection field shown represents the average pattern of LGN activity that has driven that neuron to a strong response; each neuron prefers a different pattern at a specific location on the retinal surface. Preferences from the set of all V1 neurons form a smooth topographic map covering the range of orientations present in the input patterns, yielding an orientation map similar to those from monkeys [16]. For instance, the map shows iso-feature domains, pinwheel centers, fractures, saddle points, and linear zones, with a ring-shaped Fourier transform. As in animals [46], orientation selectivity is preserved over a very wide range of contrasts, due to the effect of lateral inhibitory connections in the LGN and in V1 that normalize responses to be relative to activation of neighboring neurons rather than absolute levels of contrast [29].

Similar results are found for models including each of the other low-level features of images, with specific map patterns that match those found in animals. Figure 4 shows results from the larger orientation, ocular dominance, and motion direction simulation from Fig. 2; each neuron becomes selective for some portion of this multidimensional feature space, and together they account for the variation across this space that was seen during self-organization [13]. Other simulations not included here show how color, spatial frequency, and disparity preferences and maps can develop when appropriate information is made available to V1 through additional RGC/LGN sheets [5, 7, 38, 40]. As described in the original source for each model, the model results for each dimension have been evaluated against the available animal data, and capture the main aspects of the feature value coverage and the spatial organization of the maps [31, 38]. The maps simulated together (e.g. orientation and ocular dominance) also tend to intersect at right angles, such that high-gradient regions in one map avoid high-gradient regions in others [13].

These patterns primarily emerge from geometric constraints on smoothly mapping the range of values for the indicated feature, within a two-dimensional retinotopic map [31]. They are also affected by the relative amount by which each feature varies in the input dataset, how often each feature appears, and other aspects of the input image statistics [12]. For instance, orientation maps trained on natural image inputs develop a preponderance of neurons with horizontal and vertical orientation preferences, which is also seen seen in ferret maps and reflects the statistics of contours found in natural images [11, 20].

While the feature maps and afferent connections of neurons primarily represent a decomposition of the image into commonly recurring local features, lateral connections between these neurons store patterns of correlation between each neuron that represent larger-scale structure and correlations. Figure 4 shows the pattern of lateral connectivity for a neuron embedded in an orientation, ocular dominance, and motion direction map. Because the lateral connections are also modified by Hebbian learning, they represent correlations between neurons, and are thus strong for short-range connections (due to the shared retinotopic preference of those neurons) and between other neurons often coactivated during self-organization (e.g. those sharing orientation, direction, and eye preferences). The lateral connections are thus patchy and orientation and direction specific, as found in animals [18, 43, 50]. Neurons with low levels of selectivity for any of those dimensions (e.g. binocular neurons) receive connections from a wide range of feature preferences, while highly selective neurons receive more specific connections, reflecting the different patterns of correlation in those cases. These connection patterns represent predictions, as only a few of these relationships have been tested so far in animals. The model strongly predicts that lateral connection patterns will respect all maps that account for a significant fraction of the response variance of the neurons, because each of those features will affect the correlation between neurons.

Overall, where it has been possible to make comparisons, these models have been shown to reproduce the main features of the experimental data, using a small set of assumptions. In each case, the model demonstrates how the experimentally measured map can emerge from Hebbian learning of corresponding patterns of subcortical and cortical activity. The models thus illustrate how the same basic, general-purpose adaptive mechanism will lead to very different organizations, depending on the geometrical and statistical properties of that feature. Future work will focus on showing how all the results so far could emerge simultaneously in a single model (as outlined in Ref. [7]).

3.2 Surround Modulation

Given a model with realistically patchy, specific lateral connectivity and realistic single-neuron properties, as described above, the patterns of interaction between neurons can be compared with neurophysiological evidence for surround modulation—influences on neural responses from distant patterns in the visual field. These studies

can help validate the underlying model circuit, while helping understand how the visual cortex will respond to complicated patterns such as natural images.

For instance, as the size of a patch of grating is increased, the response of a V1 neuron typically increases at first, reaches a peak, and then decreases [45, 47, 55]. Similar patterns can be observed in a GCAL-based model orientation map with complex cells and separate inhibitory and excitatory subpopulations (figure from Ref. [3]; not shown). Small patterns initially activate neurons weakly, due to low overlap with the afferent receptive fields of layer 4 cells, but the response increases with larger patterns. For large enough patterns, lateral interactions are strong and in most locations net inhibitory, causing many neurons to be suppressed (leading to a subsequent dip in response). The model demonstrates that the lateral interactions are sufficient to account for typical size tuning effects, and also accounts for less commonly reported effects that result from neurons with different specific self-organized patterns of lateral connectivity. The model thus accounts both for the typical pattern of size tuning, and explains why such a diversity of patterns is observed in animals. The results from these studies and related studies of orientation-dependent effects [3] suggest both that lateral interactions may underlie many of the observed surround modulation effects, and also that the diversity of observed effects can at least in part be traced to the diversity of lateral connection patterns, which in turn is a result of the various sequences of activations of the neurons during development.

3.3 Aftereffects

The previous sections have focused on the network organization and operation after Hebbian learning can be considered to be completed. However, the visual system is continually adapting to the visual input even during normal visual experience, resulting in phenomena such as visual aftereffects [53]. To investigate whether and how this adaptation differs from long-term self-organization, we tested LISSOM and GCAL-based models with stimuli used in visual aftereffect experiments [9, 19]. Surprisingly, the same Hebbian equations that allow neurons and maps to develop selectivity also lead to realistic aftereffects, such as for orientation and color (Fig. 5). In the model, we assume that connections adapt during normal visual experience just as they do in simulated long-term development, albeit with a lower learning rate appropriate for adult vision. If so, neurons that are coactive during a particular visual stimulus (such as a vertical grating) will become slightly more strongly laterally connected as they adapt to that pattern. Subsequently, the response to that pattern will be reduced, due to increased lateral excitation that leads to net (disynaptic) lateral inhibition for high contrast patterns like those in the aftereffect studies. Assuming a population decoding model such as the vector sum [9], there will be no change in the perceived orientation of the adaptation pattern, but the perceived value of a nearby orientation will be repelled *away* from the adapting stimulus, because the neurons activated during adaptation now inhibit each other more strongly, shifting the population response. These changes are the direct result of Hebbian learning

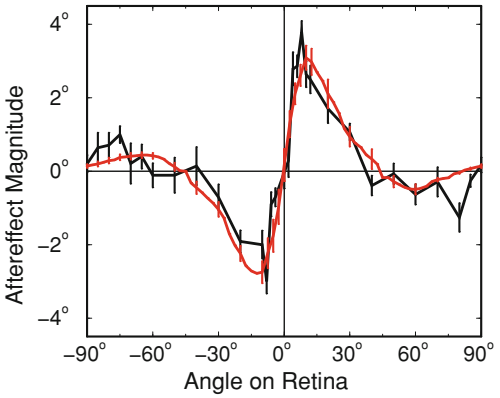

Fig. 5 *Tilt aftereffects from short-term self-organization*. If the fully organized network is repeatedly presented patterns with the same orientation, connection strengths are updated by Hebbian learning (as during development, but at a lower learning rate). The net effect is increased inhibition, which causes the neurons that responded during adaptation to respond less afterwards. When the overall response is summarized as a "perceived value" using a vector average, the result is systematic shifts in perception, such that a previously similar orientation will now seem very different in orientation, while more distant orientations will be unchanged or go in the opposite direction (*red line*, with error bars showing the standard error of measurement). These patterns are a close match to results from humans [34] (e.g. the subject from [34] plotted here as a black line, with error bars showing the standard error of measurement), suggesting that short-term and long-term adaptation share similar rules. Reprinted from Ref. [9] and replotting data from Ref. [34].

of intracortical connections, as can be shown by disabling learning for all other connections and observing no change in the overall behavior.

Interestingly, for distant orientations, the human data suggests an attractive effect, with a perceived orientation shifted *towards* the adaptation orientation [34]. The model reproduces this feature as well, and provides the novel explanation that this indirect effect is due to the divisive normalization term in the Hebbian learning equation (Eq. 7). Specifically, when the neurons activated during adaptation increase their mutual inhibition, the normalization term forces this increase to come at the expense of connections to other neurons *not* (or only weakly) activated during adaptation. Those neurons are thus disinhibited, and can respond more strongly than before, shifting the response towards the adaptation stimulus.

Similar patterns occur for the McCollough Effect [30]; see Ref. [19]. Here the adaptation stimulus coactivates neurons selective for orientation, color, or both, and again the lateral interactions between all these neurons are strengthened. Subsequent stimuli then appear different in both color and orientation, in patterns similar to the human data. Interestingly, the McCollough effect can last for months, which suggests that the modelled changes in lateral connectivity can become essentially permanent, though the effects of short-term exposure typically fade in darkness or in subsequent visual experience.

Overall, the model suggests that the same process of Hebbian learning could explain both long-term development and short-term adaptation, unifying phenomena previously considered distinct. Of course, the biophysical mechanisms may indeed be distinct, potentially operating at different time scales and this short-term adaptation being largely temporary rather than the permanent changes found early in development. Even so, the results here suggest that both early development and adult adaptation may operate using similar mathematical principles. How mechanisms for long- and short-term plasticity may interact, including possible transitions from long- to short term plasticity during so-called "critical periods", is an important area for future modelling and experimental studies.

3.4 Time Course of Neural Responses

Visual aftereffects reflect changes in responses over the course of seconds and minutes, but with a sufficiently short time step (i.e., neural connection delay), the detailed subsecond time course of GCAL LGN and V1 neurons can also be investigated, before adaptation takes effect. Due to the recurrent nature of the GCAL architecture, responses to inputs go through a stereotypical time course that serves to highlight temporal differences in input patterns, just as the mechanisms outlined above serve to highlight spatial differences (e.g. contrast edges). As part of an ongoing project to understand temporal aspects of cortical processing [51], the temporal response properties of the GCAL orientation map were adjusted to match experimental data from [24] for the LGN, and to a fit of experimental data from [1], using a time step size and projection delay of $\delta t = 0.002$ (roughly corresponding to 0.5 ms) instead of the previous $\delta t = 0.05$. Remarkably, even though the model was originally built only to study spatial processing, we were able to match the time course at both the LGN and V1 levels by adjusting only a single parameter in the model LGN and V1: λ, which controls temporal smoothing of activity values in Eq. 3. Figure 6 compares the time course of GCAL responses to a step change in the visual input to the experimental data.

At the LGN level, λ controls only how fast the neural response can change; the underlying trends in the time course reflect the recurrent processing in the LGN, i.e., there is initially a strong response due to the afferent connectivity, which is then reduced by the divisive lateral inhibition in the LGN, with some ringing for this particular λ value. Higher levels of damping (larger λ) would eliminate this ringing, as suggested by some other experimental studies [51], but it has been retained here to show the match to this study. These results are intriguing, because they show how detailed and realistic temporal properties can arise from a circuit with elements originally added for contrast gain control (i.e., the lateral inhibition in the LGN); transient responses emerge as a natural consequence and will serve to highlight temporally varying input.

The time courses of response at the V1 level are similar, and reflect both the time course of its LGN input, and also that due to its own recurrent lateral connections, again smoothed by a λ parameter (Eq. 2). The same conclusions also apply at the

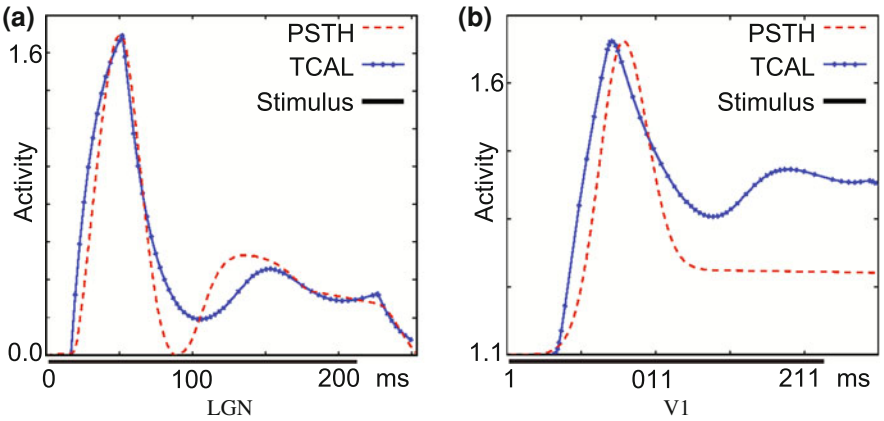

Fig. 6 *Model LGN and V1 temporal responses*. **a** The *dashed red line* shows experimental measurements of a cat LGN peristimulus time histogram (PSTH) in response to a step input [24], with a characteristic large onset response, some ringing, smaller sustained response, and eventual offset. The *blue line* shows the best fit from adjusting the activity smoothing in a GCAL orientation map; the fit is remarkably good considering that only a single GCAL parameter was varied (λ for the LGN). **b** The *dashed red line* shows results from a simple mathematical model of experimental measurements from monkey V1 [1], compared to the best fit response from GCAL from varying λ for V1 (and using the above fit at the LGN level). Again, the fit is remarkably good given the limited control provided by λ, indicating that the underlying dynamics of the model network are already a good match to neural circuits. Reprinted from Ref. [51]

V1 level, with V1 responses higher for changing stimuli than for sustained inputs, and reflecting the structure of the recurrent network in which neurons are embedded, rather than complex single-neuron temporal properties.

4 Discussion and Future Work

The results reviewed above illustrate a general approach to understanding the large-scale development, organization, and function of cortical areas, as a way of understanding processing for real-world data in general. The models show that a relatively small number of basic and largely uncontroversial assumptions and principles may be sufficient to explain a very wide range of experimental results from the visual cortex. Even very simple neural units, i.e., firing-rate point neurons, generically connected into topographic maps with initially random or isotropic weights, can form a wide range of specific feature preferences and maps via unsupervised normalized Hebbian learning of natural images and spontaneous activity patterns. The resulting maps consist of neurons with realistic spatial and temporal response properties, with variability due to visual context and recent history that explains significant aspects of surround modulation and visual aftereffects. The simulator and

example simulations are freely downloadable from www.topographica.org, allowing any interested researcher to build on this work.

Although all the results listed above were from V1, the cortical architecture contained no vision-specific elements at the start of the simulation, and is thus general purpose. Similar models have already been used for other cortical regions, such as rodent barrel cortex [57]. Combining the existing models into a single, runnable visual system is very much a work in progress, but the results so far suggest that doing so will be both feasible and valuable.

As previously emphasized, many of the individual results found with GCAL can also be obtained using other modelling approaches, which can be complementary to the processes modeled by GCAL. For instance, it is possible to generate orientation maps without any activity-dependent plasticity, through the initial wiring pattern between the retina and the cortex [37, 42] or within the cortex itself [25]. Such an approach cannot explain subsequent experience-dependent development, whereas the Hebbian approach of GCAL can explain both the initial map and later plasticity, but it is of course possible that the initial map and the subsequent plasticity occur via different mechanisms. Other models are based on abstractions of some of the mechanisms in GCAL [22, 35, 58, 61], operating similarly but at a higher level. GCAL is not meant as a competitor to such models, but as a concrete, physically realizable implementation of those ideas, forming a prototype of both the biological system and potential future artificial vision systems.

5 GCAL as a Starting Point for Higher-Level Mechanisms

At present, all of the models reviewed contain feedforward and lateral connections, but no feedback from higher cortical areas to V1 or from V1 to the LGN, because such feedback has not been found necessary to replicate the features surveyed. However, note that nearly all of the physiological data considered was from anesthetized animals not engaged in any visually mediated behaviors. Under those conditions, it is not surprising that feedback would have relatively little effect. Corticocortical and corticothalamic feedback is likely to be crucial to explain how these circuits operate during natural vision [49, 52], and determining the form and function of this feedback is an important aspect of developing a general-purpose cortical model. By clearly establishing which V1 properties do not require such feedback, GCAL represents an excellent starting point for building and understanding models of these phenomena.

Eventually, such models will need to be trained using input that reflects the complete context in which an animal develops, rather than just the fixed and arbitrary stream of training images used so far. Ideally, a model of visual system development in primates would be driven by color, stereo, foveated video streams replicating typical patterns of eye movements, movements of an animal in its environment, and responses to visual patterns. Collecting data of this sort is difficult, and moreover cannot capture any causal or contingent relationships between the current visual input

and the current neural organization that can affect future eye and organism movements that will then change the visual input. In the long run, to account for more complex aspects of visual system development such as visual object recognition and optic flow processing, it will be necessary to implement the models as embodied, situated agents [39, 56] embedded in the real world or in realistic 3D virtual environments. Building such robotic or virtual agents will add significant additional complexity, however, so it is important first to see how much of the behavior of V1 neurons can be addressed by the present "open-loop", non-situated approach.

As discussed throughout, the main focus of this modelling work has been on replicating experimental data using a small number of computational primitives and mechanisms, with a goal of providing a concise, concrete, and relatively simple explanation for a wide and complex range of experimental findings. A complete explanation of visual cortex development and function would go even further, demonstrating more clearly *why* the cortex should be built in this way, and precisely what information-processing purpose this circuit performs. For instance, realistic receptive fields can be obtained from "normative" models embodying the idea that the cortex is developing a set of basis functions to represent input patterns faithfully, with only a few active neurons [14, 26, 36, 41], maps can emerge by minimizing connection lengths in the cortex [28], and lateral connections can be modelled as decorrelating the input patterns [6, 21]. The GCAL model can be seen as a concrete, mechanistic implementation of these ideas, showing how a physically realizable local circuit could develop receptive fields with good coverage of the input space, via lateral interactions that also implement sparsification via decorrelation [31]. Making more explicit links between mechanistic models like GCAL and normative theories is an important goal for future work. Meanwhile, there are many aspects of cortical function not explained by current normative models. The focus of the current line of research is on first capturing those phenomena in a general-purpose mechanistic model, so that researchers can then build deeper explanations for why these computations are useful for the organism. The following section outlines the beginning of such an explanation, in the context of data processing in general.

6 Building Complex Systems

If one steps back from cortical modelling to consider what the underlying circuits in GCAL are doing, the simulations reported here suggest a relatively straightforward process for building a circuit or device to process real-world input data:

1. Make sure your input data is organized into a meaningful two-dimensional spatial arrangement. Such a representation comes for free with many types of input data, reflecting the spatiotemporal ordering of the physical world, but for other types of data (as in the olfactory system) the data must first be organized into a two-dimensional pattern in some space (as for the odorant maps in the olfactory bulb [27]). GCAL can perform such mapping itself for small-scale networks, but large

enough networks would require a very extensive set of connections, and thus establishing some initial mapping (as for retinotopy in animals and in GCAL) can be considered a prerequisite.
2. Given this spatial representation, decompose your input data to be processed by a large array of local processing units by mapping it to a set of simulated neurons with topographically local afferent connection fields, so that each can compute independently.
3. Connect your processing units laterally with a pattern that ensures that local patches of neurons respond to similar inputs, and that more distant neurons respond to different inputs. This type of network will generate "activity bubbles" in response to a strong input pattern, with bubbles in different locations depending on the input pattern, to achieve coverage of the input pattern features.
4. Allow neural excitability to vary via homeostatic plasticity, so that neurons adapt to the patterns of input strength over time.
5. Adjust all connections using Hebbian learning, to ensure that neurons become selective for particular local patterns, while others become selective for other patterns.

The resulting network will thus remap the inputs into a sparse representation that covers the ranges of variability in the input data, with lateral connectivity that makes neurons compete to represent a given input, while filling in expected patterns in cases of weak inputs. Short-term adaptation following similar rules as self-organization will make neurons respond most strongly to changes in the input statistics, again highlighting important events. At an even faster time scale, the temporal responses of these recurrently connected neurons will again highlight moment-to-moment changes in input patterns.

The resulting spatially and temporally decorrelated representation can then be available as a substrate for higher-level processing such as reinforcement or supervised learning, acting on sparse patterns that reflect the underlying sources of variability in the environment rather than the initial dense and highly redundant pattern of inputs that reflect the structure of the input receptors. In principle, this approach can be used for any type of input data, potentially offering a starting point for building complex systems for data processing in general.

7 Conclusions

The GCAL model results suggest that it will soon be feasible to build a single model visual system that will account for a very large fraction of the visual response properties, at the firing rate level, of V1 neurons in a particular species. Such a model will help researchers make testable predictions to drive future experiments to understand cortical processing, as well as determine which properties require more complex approaches, such as feedback, attention, and detailed neural geometry and dynamics. The model suggests that cortical neurons develop to cover the typical range

of variation in their thalamic inputs, within the context of a smooth, multidimensional topographic map, and that lateral connections store pairwise correlations and use this information to modulate responses to natural scenes, dynamically adapting to both long-term and short-term visual input statistics.

Because the model cortex starts without any specialization for vision, it represents a general model for any cortical region, and is also an implementation for a generic information processing device that could have important applications outside of neuroscience. By integrating and unifying a wide range of experimental results, the model should thus help advance our understanding of cortical processing and real-world information processing in general.

Acknowledgments Thanks to all of the collaborators whose modelling work is reviewed here, and to the members of the Developmental Computational Neuroscience research group, the Institute for Adaptive and Neural Computation, and the Doctoral Training Centre in Neuroinformatics, at the University of Edinburgh, for discussions and feedback on many of the models. This work was supported in part by the UK EPSRC and BBSRC Doctoral Training Centre in Neuroinformatics, under grants EP/F500385/1 and BB/F529254/1, and by the US NIMH grant R01-MH66991. Computational resources were provided by the Edinburgh Compute and Data Facility (ECDF).

References

1. D.G. Albrecht, W.S. Geisler, R.A. Frazor, A.M. Crane, Visual cortex neurons of monkeys and cats: temporal dynamics of the contrast response function. J. Neurophysiol. **88**(2), 888–913 (2002)
2. H.J. Alitto, W.M. Usrey, Origin and dynamics of extraclassical suppression in the lateral geniculate nucleus of the macaque monkey. Neuron **57**(1), 135–146 (2008)
3. J. Antolik, Unified developmental model of maps, complex cells and surround modulation in the primary visual cortex. Ph.D. thesis, School of Informatics, The University of Edinburgh, Edinburgh, UK, 2010
4. J. Antolik, J.A. Bednar, Development of maps of simple and complex cells in the primary visual cortex. Frontiers Comput. Neurosci. **5**, 17 (2011)
5. C.E. Ball, J.A. Bednar, A self-organizing model of color, ocular dominance, and orientation selectivity in the primary visual cortex. in Society for Neuroscience Abstracts. Society for Neuroscience, www.sfn.org, Program No. 756.9 (2009)
6. H.B. Barlow, P. Földiák, Adaptation and decorrelation in the cortex, in *The Computing Neuron*, ed. by R. Durbin, C. Miall, G. Mitchison (Addison-Wesley, Reading, 1989), pp. 54–72
7. J.A. Bednar, Building a mechanistic model of the development and function of the primary visual cortex. J. Physiol. (Paris, 2012 in press)
8. J.A. Bednar, A. Kelkar, R. Miikkulainen, Scaling self-organizing maps to model large cortical networks. Neuroinformatics **2**, 275–302 (2004)
9. J.A. Bednar, R. Miikkulainen, Tilt aftereffects in a self-organizing model of the primary visual cortex. Neural Comput. **12**(7), 1721–1740 (2000)
10. J.A. Bednar, R. Miikkulainen, Self-organization of spatiotemporal receptive fields and laterally connected direction and orientation maps. Neurocomputing **52–54**, 473–480 (2003)
11. J.A. Bednar, R. Miikkulainen, Prenatal and postnatal development of laterally connected orientation maps. in Computational Neuroscience: Trends in Research, (2004), p. 985–992
12. J.A. Bednar, R. Miikkulainen, Prenatal and postnatal development of laterally connected orientation maps. Neurocomputing **58–60**, 985–992 (2004)

13. J.A. Bednar, R. Miikkulainen, Joint maps for orientation, eye, and direction preference in a self-organizing model of V1. Neurocomputing **69**(10–12), 1272–1276 (2006)
14. A.J. Bell, T.J. Sejnowski, The independent components of natural scenes are edge filters. Vision. Res. **37**, 3327 (1997)
15. E.L. Bienenstock, L.N. Cooper, P.W. Munro, Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. J. Neurosci. **2**, 32–48 (1982)
16. G.G. Blasdel, Orientation selectivity, preference, and continuity in monkey striate cortex. J. Neurosci. **12**, 3139–3161 (1992)
17. V. Bonin, V. Mante, M. Carandini, The suppressive field of neurons in lateral geniculate nucleus. J. Neurosci. **25**, 10844–10856 (2005)
18. W.H. Bosking, Y. Zhang, B.R. Schofield, D. Fitzpatrick, Orientation selectivity and the arrangement of horizontal connections in tree shrew striate cortex. J. Neurosci. **17**(6), 2112–2127 (1997)
19. J. Ciroux, Simulating the McCollough effect in a self-organizing model of the primary visual cortex. Master's thesis, The University of Edinburgh, Scotland, UK, 2005
20. D.M. Coppola, L.E. White, D. Fitzpatrick, D. Purves, Unequal representation of cardinal and oblique contours in ferret visual cortex. Proc. Nat. Acad. Sci. U.S.A. **95**(5), 2621–2623 (1998)
21. D.W. Dong, Associative decorrelation dynamics: A theory of self-organization and optimization in feedback networks, in *Advances in Neural Information Processing Systems 7*, ed. by G. Tesauro, D.S. Touretzky, T.K. Leen (MIT Press, Cambridge, 1995), pp. 925–932
22. B.J. Farley, H. Yu, D.Z. Jin, M. Sur, Alteration of visual input results in a coordinated reorganization of multiple visual cortex maps. J. Neurosci. **27**(38), 10299–10310 (2007)
23. F. Felisberti, A.M. Derrington, Long-range interactions modulate the contrast gain in the lateral geniculate nucleus of cats. Vis. Neurosci. **16**, 943–956 (1999)
24. K. Funke, F. Wörgötter, On the significance of temporally structured activity in the dorsal lateral geniculate nucleus (LGN). Prog. Neurobiol. **53**(1), 67–119 (1997)
25. A. Grabska-Barwinska, C. von der Malsburg, Establishment of a scaffold for orientation maps in primary visual cortex of higher mammals. J. Neurosci. **28**(1), 249–257 (2008)
26. A. Hyvärinen, P.O. Hoyer, A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. Vision. Res. **41**(18), 2413–2423 (2001)
27. T. Imai, H. Sakano, L.B. Vosshall, Topographic mapping-the olfactory system. Cold Spring Harb. Perspect. Biol. Med. 2(8), (2010)
28. A.A. Koulakov, D.B. Chklovskii, Orientation preference patterns in mammalian visual cortex: a wire length minimization approach. Neuron **29**, 519–527 (2001)
29. J.S. Law, J. Antolik, and J.A. Bednar. Mechanisms for stable and robust development of orientation maps and receptive fields. Technical report, School of Informatics, The University of Edinburgh, 2011. EDI-INF-RR-1404
30. C. McCollough, Color adaptation of edge-detectors in the human visual system. Science **149**(3688), 1115–1116 (1965)
31. R. Miikkulainen, J.A. Bednar, Y. Choe, J. Sirosh, *Computational Maps in the Visual Cortex* (Springer, Berlin, 2005)
32. K.D. Miller, A model for the development of simple cell receptive fields and the ordered arrangement of orientation columns through activity-dependent competition between ON- and OFF-center inputs. J. Neurosci. **14**, 409–441 (1994)
33. K.D. Miller, D.J.C. MacKay, The role of constraints in Hebbian learning. Neural Comput. **6**, 100–126 (1994)
34. D.E. Mitchell, D.W. Muir, Does the tilt after effect occur in the oblique meridian? Vision. Res. **16**, 609–613 (1976)
35. K. Obermayer, H. Ritter, K.J. Schulten, A principle for the formation of the spatial structure of cortical feature maps. Proc. Nat. Acad. Sci. U.S.A. **87**, 8345–8349 (1990)
36. B.A. Olshausen, D.J. Field, Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature **381**, 607–609 (1996)
37. S.-B. Paik, D.L. Ringach, Retinal origin of orientation maps in visual cortex. Nat. Neurosci. **14**(7), 919–925 (2011)

38. C.M. Palmer, Topographic and laminar models for the development and organisation of spatial frequency and orientation in V1. Ph.D. thesis, School of Informatics, The University of Edinburgh, Edinburgh, UK, 2009
39. Z.W. Pylyshyn, Situating vision in the world. Trends Cogn. Sci. **4**(5), 197–207 (2000)
40. T. Ramtohul, A self-organizing model of disparity maps in the primary visual cortex. Master's thesis, The University of Edinburgh, Scotland, UK, 2006
41. M. Rehn, F.T. Sommer, A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields. J. Comput. Neurosci. **22**(2), 135–146 (2007)
42. D.L. Ringach, On the origin of the functional architecture of the cortex. PLoS One **2**(2), e251 (2007)
43. B. Roerig, J.P. Kao, Organization of intracortical circuits in relation to direction preference maps in ferret visual cortex. J. Neurosci. 19(24), RC44 (1999)
44. A.B. Saul, A.L. Humphrey, Evidence of input from lagged cells in the lateral geniculate nucleus to simple cells in cortical area 17 of the cat. J. Neurophysiol. **68**(4), 1190–1208 (1992)
45. M.P. Sceniak, D.L. Ringach, M.J. Hawken, R. Shapley, Contrast's effect on spatial summation by macaque V1 neurons. Nat. Neurosci. **2**, 733–739 (1999)
46. G. Sclar, R.D. Freeman, Orientation selectivity in the cat's striate cortex is invariant with stimulus contrast. Exp. Brain Res. **46**, 457–461 (1982)
47. F. Sengpiel, A. Sen, C. Blakemore, Characteristics of surround inhibition in cat area 17. Exp. Brain Res. **116**(2), 216–228 (1997)
48. H.T. Siegelmann, E.D. Sontag, Turing computability with neural nets. Appl. Math. Lett. **4**, 77–80 (1991)
49. A.M. Sillito, J. Cudeiro, H.E. Jones, Always returning: feedback and sensory processing in visual cortex and thalamus. Trends Neurosci. **29**(6), 307–316 (2006)
50. L.C. Sincich, G.G. Blasdel, Oriented axon projections in primary visual cortex of the monkey. J. Neurosci. **21**, 4416–4426 (2001)
51. J.-L. Stevens, A temporal model of neural activity and VSD response in the primary visual cortex. Master's thesis, The University of Edinburgh, Scotland, UK, 2011
52. A. Thiele, A. Pooresmaeili, L.S. Delicato, J.L. Herrero, P.R. Roelfsema, Additive effects of attention and stimulus contrast in primary visual cortex. Cereb. Cortex **19**(12), 2970–2981 (2009)
53. P. Thompson, D. Burr, Visual aftereffects. Curr. Biol. **19**(1), R11–14 (2009)
54. G.G. Turrigiano, Homeostatic plasticity in neuronal networks: the more things change, the more they stay the same. Trends Neurosci. **22**(5), 221–227 (1999)
55. C. Wang, C. Bardy, J.Y. Huang, T. FitzGibbon, B. Dreher, Contrast dependence of center and surround integration in primary visual cortex of the cat. J. Vis. 9(1), 20.1–15, (2009)
56. J. Weng, J.L. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, E. Thelen, Autonomous mental development by robots and animals. Science **291**(5504), 599–600 (2001)
57. S.P. Wilson, J.S. Law, B. Mitchinson, T.J. Prescott, J.A. Bednar, Modeling the emergence of whisker direction maps in rat barrel cortex. PLoS One, 5(1), (2010)
58. F. Wolf, T. Geisel, Universality in visual cortical pattern formation. J. Physiol. Paris **97**(2–3), 253–264 (2003)
59. J. Wolfe, L.A. Palmer, Temporal diversity in the lateral geniculate nucleus of cat. Vis. Neurosci. **15**(4), 653–675 (1998)
60. R.O.L. Wong, Retinal waves and visual system development. Annu. Rev. Neurosci. **22**, 29–47 (1999)
61. H. Yu, B.J. Farley, D.Z. Jin, M. Sur, The coordinated mapping of visual space and response features in visual cortex. Neuron **47**(2), 267–280 (2005)

Chapter 8
Neuro-Centric and Holocentric Approaches to the Evolution of Developmental Neural Networks

Julian F. Miller

Abstract In nature, brains are built through a process of biological development in which many aspects of the network of neurons and connections change and are shaped by external information received through sensory organs. From numerous studies in neuroscience, it has been demonstrated that developmental aspects of the brain are intimately involved in learning. Despite this, most artificial neural network (ANN) models do not include developmental mechanisms and regard learning as the adjustment of connection weights. Incorporating development into ANNs raises fundamental questions. What level of biological plausibility should be employed? In this chapter, we discuss two artificial developmental neural network models with differing degrees of biological plausibility. One takes the view that the neuron is fundamental (neuro-centric) so that all evolved programs are based at the level of the neuron, the other carries out development at an entire network level and evolves rules that change the network (holocentric). In the process, we hope to reveal some important issues and questions that are relevant to researchers wishing to create other such models.

1 Introduction

Although artificial neural networks (ANNs) are over 60 years old [1] few would argue that they even approach the learning capabilities of relatively simple organisms. Yet, over this period our understanding of neuroscience has increased enormously [2] and computer systems have gained enormous improvements in speed. We suggest that a major weakness of many ANNs models is that they follow the "synaptic dogma" (SD) which encodes learned knowledge solely in the form of connection strengths

J. F. Miller (✉)
Department of Electronics, University of York, York, UK
e-mail: julian.miller@york.ac.uk

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_8,

(i.e. weights). SD gives rise to "catastrophic forgetting" (CF) [3–5]. This is where trained ANNs when re-trained on a new problem, forget how to solve the original problem. Interestingly, although Judd showed that learning the weights in a fixed neural network is an NP-complete problem [6], Baum proved that if one allows the addition of neurons and weighted connections, ANNs can solve learning problems in polynomial time [7]. Shortly after Baum's paper "constructive ANNs" were devised [8, 9]. These use supervised learning, in which neurons and connections are added gradually and weights adjusted incrementally until training errors are reduced. Quinlan discussed "dynamic networks" which he defined as "any artificial neural network that automatically changes its structure through exposure to input stimuli" [10, 11]. He observed that "although there has been some work on the removal of weighted connections, these schemes have not explored a general framework where the number of connections is both increased and decreased independently of the number of processing units" [10]. Combining evolution with development is a way of exploring the more dynamic networks that Quinlan discussed. However, despite the fact that the original inspiration for ANNs came from knowledge about the brain, it still remains that very few ANN models use both evolution and development, both of which are fundamental to the construction of the brain [12].

Apart from the problem of catastrophic forgetting, there is much research that undermines the notion that memory in brains is principally related to synaptic strengths. Firstly, it is now known that most synapses are not static but are constantly pruned away and replaced by new synapses and learning is related strongly to this process [13]. Secondly, synaptic plasticity does not merely involve the increase or decrease of the number of synapses, but is related also to the exact location of the synapses on the dendritic tree and the actual geometry of the dendritic branches. Thirdly, much research indicates that learning and environmental interaction are strongly related to *structural* changes in neurons. Dark-reared mice when placed in the light develop new dendrites in the visual cortex within days [14]. Animals reared in complex environments involving active learning have an increased density of dendrites and synapses [15, 16].[1] Within the brains of songbirds in the breeding season, it is has been found that the number, size and spacing of neurons increases [18]. In a well-known study it was found that the hippocampi of London taxi drivers who must remember large parts of central London, are significantly larger relative to those of control subjects [19]. Rose is quite emphatic about the role of structural change in memory and argues that after a few hours of learning the brain is permanently altered "if only by shifting the number and position of a few dendritic spines on a few neurons in particular brain regions" [20]. Another, almost obvious, aspect supporting the view that structural changes in the brain are strongly associated with learning, is simply that the most significant period of learning in animals happens in infancy, when the brain is developing [21].

[1] However, a recent study showed that environmental enrichment alone does not significantly increase hippocampal neurogenesis or bestow spatial learning benefits in mice [17].

Our view is that structural changes in a computational network will enhance the learning capability of artificial neural networks and development appears to be a promising way of acheiving this. The idea is that evolution will arrive at useful developmental rules that result in enhanced learning ability. In this chapter we discuss two models that attempt to do this. The first is neuro-centric and uses evolution to build a collection of programs that represent many aspects of a neuron (i.e. dendritic and axonal branches, soma, synapses and neuron firing) [22–28]. The advantage this has is that many aspects of neuroscience can enrich and inform such an approach. The second approach is holocentric, it uses evolution and development, however evolution works at the whole system level. Most conventional ANNs are also holocentric as learning mechanisms are employed at a whole network level. Holocentric models tend to use a much higher level of abstraction of neural systems and consequently they are simpler and computationally more efficient. The holocentric model we discuss here uses a genotype which unfolds through self-modification and iteration to produce an entire computational network. At each iteration a complete functioning network is obtained and embedded self-modification operators dictate changes to the network so that a new network is produced. In both models the computational network develops at run time, so that if executed for sufficiently long periods of time, the networks can become arbitrarily large and complex. This contrasts with other well known developmental computational approaches such as HyperNEAT [29] and Cellular Encoding [30] where the phenotype is fixed and defined by artificial evolution.

The neuro-centric method uses a form of genetic programming (GP) [31] called Cartesian Genetic Programming (CGP) [32, 33] to represent the computational components of the neuron. It is referred to as the CGP developmental network (CGPDN). The other approach (holocentric) is based on a form of CGP called Self-Modifying CGP (SMCGP)[34–40]. CGP is a method for evolving graph-based computational structures so it naturally lends itself to representing ANNs. Indeed, ANNs can be encoded directly by CGP and recent work suggests this is in itself a very promising encoding [41–43]. CGP has been shown to offer advantages over other forms of GP, in that it can evolve solutions in less genotype evaluations and solutions are more compact than many other methods [44]. Like GP it is a general method for evolving programs so that it provides a method for evolving many kinds of computational structures (e.g. Boolean circuits, computer programs, sets of equations) including artificial neural networks.

The plan of the chapter is as follows. Since CGP underlies the models discussed here, we give an overview in Sect. 2. In Sect. 3 we outline a neuro-centric CGP developmental network (CGPDN). In Sect. 4 we discuss a developmental form of CGP called self-modifying CGP. In the process we compare and contrast the developmental CGP approach with the non-developmental approach. In Sect. 7 we discuss a developmental holocentric approach to ANNs that uses SMCGP to develop ANNs. We discuss and contrast the neuro-centric and holocentric approaches to ANNs in Sect. 8. We end the chapter with some observations on requirements for acheiving general learning in ANNs.

2 Cartesian Genetic Programming

For completeness we give a brief overview of CGP. A more detailed account is available in the recently published book [45]. In CGP, programs are represented in the form of directed acyclic graphs. These graphs are represented as a two-dimensional grid of computational nodes (which may, or may not, be sigmoidal neurons). The genes that make up the genotype in CGP are integers that represent where a node gets its data, what operations the node performs on the data and where the output data required by the user is to be obtained. When the genotype is decoded, some nodes may be ignored. This happens when node outputs are not used in the calculation of output data. When this happens, we refer to the nodes and their genes as 'non-coding'. We call the program that results from the decoding of a genotype a phenotype. The genotype in CGP has a fixed length. However, the size of the phenotype (in terms of the number of computational nodes) can be anything from zero nodes to the number of nodes defined in the genotype. The types of computational node functions used in CGP are decided by the user and are listed in a function look-up table. For instance, if conventional ANNs are required there might be only one function, a sigmoid. In this case function genes are not required as by default they are all sigmoid. However, in general CGP can use any combination of functions desired. In [41–43] two functions are used, sigmoid and hyperbolic tangent.

In CGP, each node in the directed graph represents a particular function and is encoded by a number of genes. One gene is the address of the computational node function in the function look-up table. We call this a *function gene*. The remaining node genes say where the node gets its data from. These genes represent addresses in a data structure (typically an array). We call these *connection genes*. Nodes take their inputs in a feed-forward manner from either the output of nodes in a previous column or from a program input. It should be noted that recurrent networks can also be represented as in [41].

The number of connection genes a node has is chosen to be the maximum number of inputs (often called the arity) that any function in the function look-up table has. The program data inputs are given the absolute data addresses 0 to $n_i - 1$ where n_i is the number of program inputs. The data outputs of nodes in the genotype are given addresses sequentially, column by column, starting from n_i to $n_i + L_n - 1$, where L_n is the user-determined upper bound of the number of nodes. The general form of a Cartesian genetic program is shown in Fig. 1. If the problem requires n_o program outputs, then n_o integers are generally added to the end of the genotype. In general, there may be a number of output genes (O_i) which specify where the program outputs are taken from. Each of these is an address of a node where the program output data is taken from. Nodes in columns cannot be connected to each other. In many cases graphs encoded are directed and feed-forward; this means that a node may only have its inputs connected to either input data or the output of a node in a previous column. The structure of the genotype is seen below the schematic in Fig. 1. All node function genes f_i are integer addresses in a look-up table of functions. All connection genes

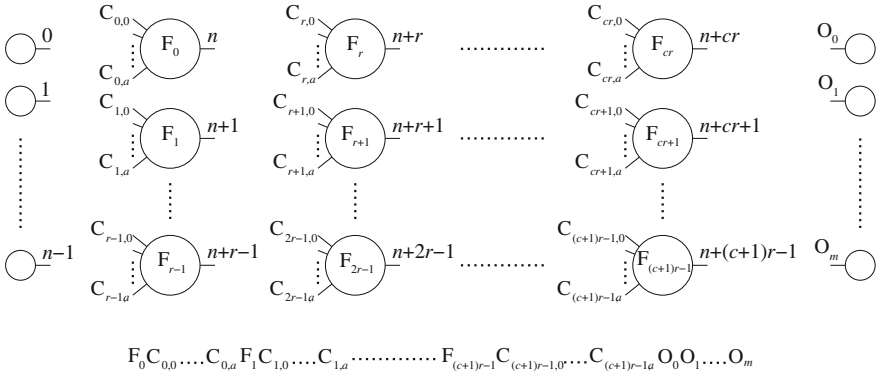

Fig. 1 General form of CGP. It is a grid of nodes whose functions are chosen from a set of primitive functions. The grid has n_c columns and n_r rows. The number of program inputs is n_i and the number of program outputs is n_o. Each node is assumed to take as many inputs as the maximum function arity a. Every data input and node output is labeled consecutively (starting at 0), which gives it a unique data address which specifies where the input data or node output value can be accessed (shown in the *figure* on the outputs of inputs and nodes)

C_{ij} are data addresses and are integers taking values between 0 and the address of the node at the bottom of the previous column of nodes.

CGP programs which solve computational problems are found using a search algorithm. The algorithm typically used is a simple kind of probabilistic hill-climber, known as a $1 + \lambda$ evolutionary algorithm [46]. Usually λ is chosen to be 4. This has the form shown in Algorithm 1.

Algorithm 1 The $(1 + 4)$ evolutionary strategy

1: **for all** i such that $0 \leq i < 5$ **do**
2: Randomly generate individual i
3: **end for**
4: Select the fittest individual, which is promoted as the parent
5: **while** a solution is not found **or** the generation limit is not reached **do**
6: **for all** i such that $0 \leq i < 4$ **do**
7: Mutate the parent to generate offspring i
8: **end for**
9: Generate the fittest individual using the following rules:
10: **if** an offspring genotype has a better or *equal* fitness than the parent **then**
11: Offspring genotype is chosen as fittest
12: **else**
13: The parent chromosome remains the fittest
14: **end if**
15: **end while**

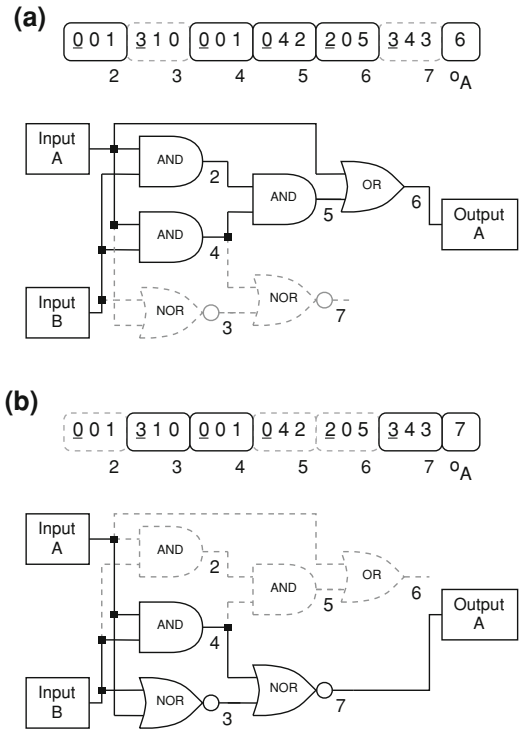

Fig. 2 An example of the point mutation operator before and after it is applied to a CGP genotype, and the corresponding phenotypes. A single point mutation occurs in the program output gene (o_A), changing the value from 6 to 7. This causes nodes 3 and 7 to become active, whilst making nodes 2, 5 and 6 inactive. The inactive areas are shown in *grey dashes*. **a** Before mutation, **b** After mutation

The mutation operator used in CGP is a point mutation operator. In a point mutation, an allele at a randomly chosen gene location is changed to another valid random value (see [45] for details). If a function gene is chosen for mutation, then a valid value is the address of any function in the function set. In cases where are is only one function allowed (e.g. all functions are sigmoidal neuron) then function genes are not required. If an input gene is chosen for mutation, then a valid value is the address of the output of any previous node in the genotype or of any program input. Also, a valid value for a program output gene is the address of the output of any node in the genotype or the address of a program input. The number of genes in the genotype that can be mutated in a single application of the mutation operator is defined by the user, and is normally a percentage of the total number of genes in the genotype.

When CGP programs are evolved, the connectivity, functionality and topology of the encoded graphical structures can change dramatically from generation to generation. The evolutionary algorithm is optimizing all these aspects simultaneously.

An example showing the application of a point mutation operator is shown in Fig. 2. In this example, the function nodes are all Boolean logic functions (with two inputs). The example also highlights how a small change in the genotype can sometimes produce a large change in the phenotype.

On line 10 of the procedure there is an extra condition: that when an offspring genotype in the population has the same fitness as the parent and there is no other offspring that is better than the parent, in that case the *offspring* is chosen as the new parent. This is a very important feature of the algorithm, which allows genotypes to change even when the phenotype does not. Such genotypes have mutations in genetic code that is inactive. Such inactive genes therefore have a neutral effect on genotype fitness. CGP genotypes are dominated by redundant genes. For instance, Miller and Smith showed that in genotypes having 4,000 nodes, the percentage of inactive nodes is approximately 95 %! [47]. The influence of neutrality in CGP has been investigated in detail [33, 47–49] and has been shown to be extremely beneficial to the efficiency of the evolutionary process on a range of test problems. The neutral drift of genotypes allows mutation to create many innovative variants of the current best genotype, some of which will occasionally contain phenotypes which are fitter than the parent.

3 A Neurocentric Model: The CGP Developmental Network

The CGPDN model has idealized the behaviour of a neuron in terms of seven main processes. The reasons for this have been discussed in more detail in [12, 28].

1. Local interaction among neighbouring branches of the same dendrite.
2. Processing of signals received from dendrites at the soma and deciding whether to fire an action potential.
3. Synaptic connections which transfer potential through axon branches to the neighbouring dendrite branches.
4. Dendrite branch growth and shrinkage. Production of new dendrite branches, removal of old branches.
5. Axon branch growth and shrinkage. Production of new axon branches, removal of old branches.
6. Creation or destruction of neurons.
7. Updating the synaptic weights (and consequently the capability to make synaptic connections) between axon branches and neighbouring dendrite branches.

Each aspect is incorporated with a separate chromosome (CGP program). The advantage of having a compartmentalized model is that different aspects of the model can be examined separately. Their utility to the whole can be assessed and if necessary the different compartments of the model can be refined.

In the CGPDN, neurons are placed randomly in a two dimensional grid so that they are only aware of their spatial neighbours (see Fig. 3). Each neuron is initially allocated a random number of dendrites, dendrite branches, one axon and a random number of axon branches. An integer variable that mimics electrical potential is used for internal computation in neurons and communication between neurons. Neurons receive information through dendrite branches, which is processed by the evolved

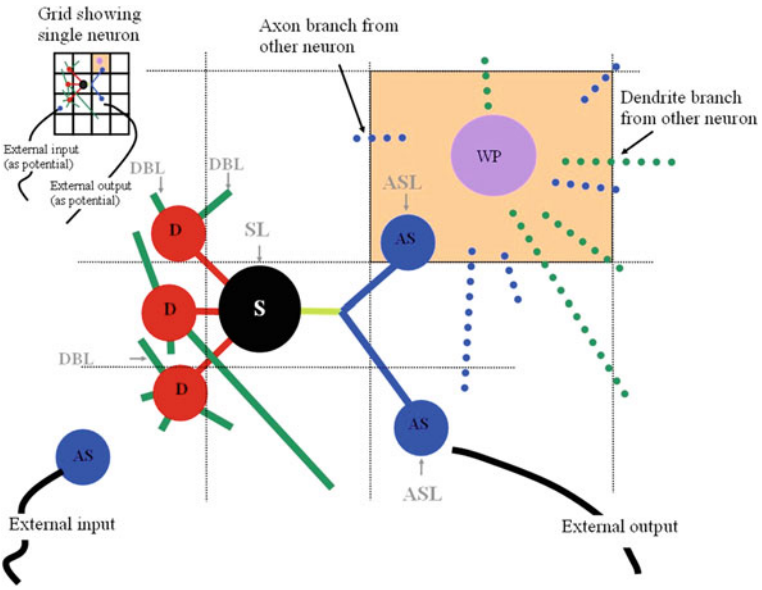

Fig. 3 On the *top left* a grid is shown containing a single neuron. The rest of the *figure* is an exploded view of the neuron. The neuron consists of seven evolved computational functions. Three are 'electrical' and process a simulated potential in the dendrite (D), soma (S) and axo-synapse branch (AS). Three more are developmental in nature and are responsible for the 'life cycle' of neural components (shown in *grey*). They decide whether dendrite branches (DBL), soma (SL) and axo-synaptic branches (ASL) should die, change or replicate. The remaining evolved computational function (WP) adjusts synaptic and dendritic weights and is used to decide the transfer of potential from a firing neuron to a neighbouring neuron

dendrite program (D) and transferred to the evolved soma program (S). S determines the final potential in the soma, which is compared to a threshold to determine whether it should fire or not. Axon branches transfer information only to dendrite branches in their proximity by passing the signals from all the neighbouring branches through a CGP program (AS), acting as an electrochemical synapse, which in turn updates the values of potential only in neighbouring branches. The weight processing chromosome (WP) adjusts the weights of potential connections to the synapse. The signal is transferred to the postsynaptic neuron having the largest weight. External inputs and outputs are also converted into potentials before being applied to the network.

3.1 Internal Neuron Variables

Four integer variables are incorporated into the CGPDN, representing either the fundamental properties of the neurons (*health*, *resistance*, and *weight*) or as an aid to computational efficiency (*state-factor*).

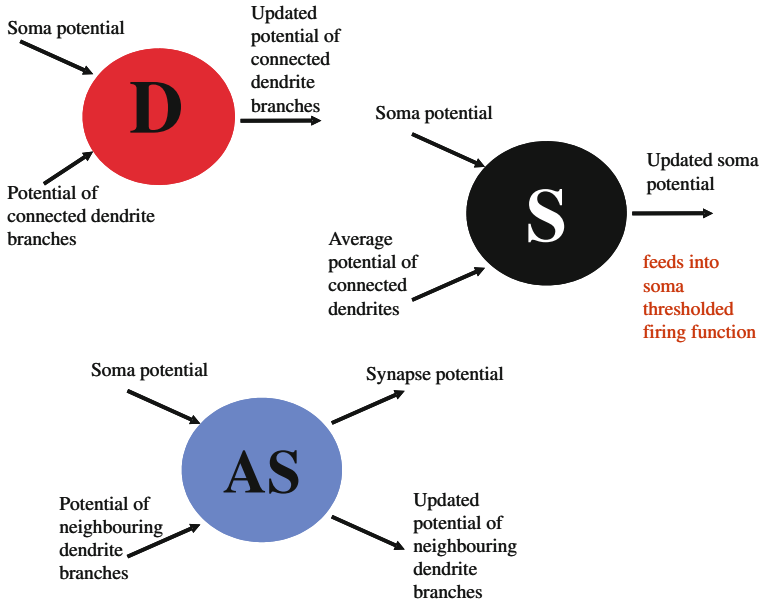

Fig. 4 Electrical processing in a neuron, showing the CGP programs for a dendrite branch, the soma and an axo-synaptic branch with their corresponding inputs and outputs

Associated with each dendrite branch and axo-synaptic connection are the variables *health*, *resistance* and *weight*. The values of these variables are adjusted by the CGP programs (see below). The *health* variable is used to govern the replication and/or death of dendrites and axon branches. A large value implicates replication a low value implicates removal (death). The *resistance* variable controls the growth and/or shrinkage of dendrites and axon branches. The weight variable is used in calculating the potentials in the network. Each soma has only two variables: *health* and *weight*.

The variable *state-factor* is used as a parameter to reduce the computational burden by keeping some of the neurons and branches inactive for a number of cycles. When the *state-factor* is zero, the neurons and branches are considered to be active and their corresponding program is run. The value of *state-factor* is affected by the CGP programs, as it is dependent on the outputs of the CGP electrical-processing chromosomes.

3.2 Electrical Processing

The electrical-processing chromosomes (D, S and AS) are responsible for signal processing inside neurons and communication between neurons. The inputs supplied to the CGP programs are shown in Fig. 4.

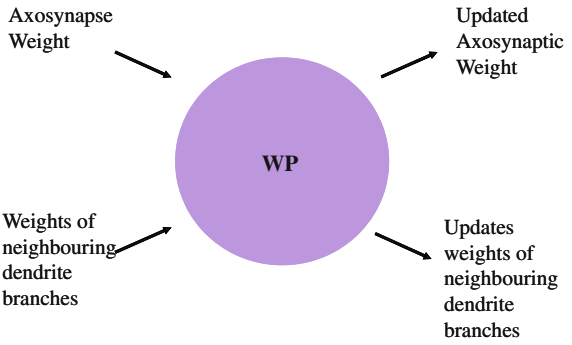

Fig. 5 Weight processing in an axo-synaptic branch, with its corresponding inputs and outputs

3.3 Weight Processing

The weight processing program (WP) is responsible for updating the *weights* of branches. The *weights* of axon and dendrite branches are also used to modulate and transfer the simulated potential [28].

Figure 5 shows the inputs and outputs to the weight-processing chromosome. The CGP program encoded in this chromosome takes as input the *weight* of the axo-synapse and the *neighbouring* dendrite branches of other neurons and produces their updated values as output. The synaptic potential produced at the axo-synapse is transferred to the dendrite branch having the highest weight after weight processing.

3.4 Developmental Aspects of Neurons

The DBL, ABL and SL CGP chromosomes (see Fig. 3) are responsible for increases or decreases in the numbers of neurons and neurite branches and also the growth and migration of neurite branches. The inputs and outputs of the programs encoded in these chromosomes are shown in Fig. 6.

3.5 Inputs and Outputs

The inputs are applied to the CGPDN through axon branches by using axo-synaptic electrical-processing chromosomes. The axon branches are distributed across the network in a similar way to the axon branches of neurons as shown in Fig. 7. These branches can be regarded as 'input neurons'. They take an input from the environment and transfer it directly to the axo-synapse input. When inputs are applied to the system, the program encoded in the axo-synaptic electrical branch chromosome is executed and the resulting signal is transferred to its neighbouring active dendrite branches.

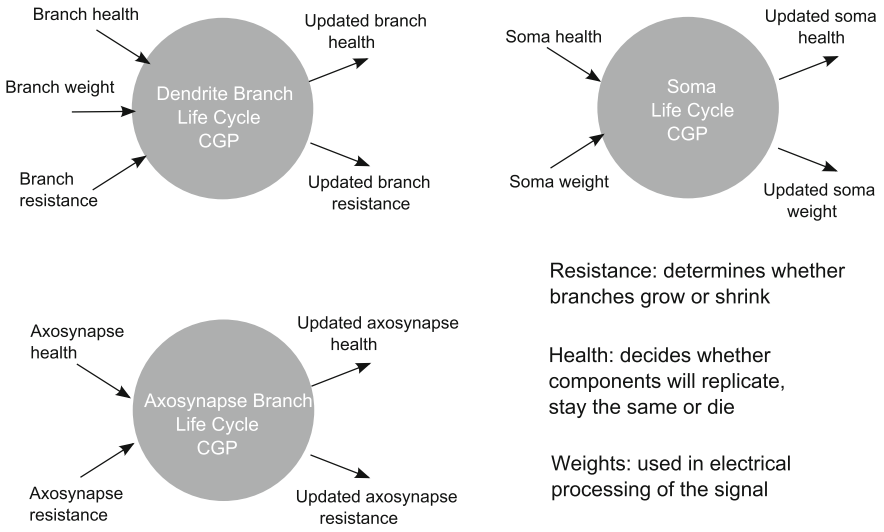

Fig. 6 Life cycle of neuron, showing CGP programs for life cycles in a dendrite branch, the soma and an axo-synapse branch, with their corresponding inputs and outputs

Similarly, there are output neurons which read the signal from the network through output dendrite branches. These output dendrite branches are distributed across the network as shown in Fig. 7. The branches are updated by the axo-synaptic chromosomes of the neurons in the same way as for other dendrite branches. The output from the output neuron is taken without further processing after every five cycles. The number of inputs and outputs can change at run time (during development), a new input or output branch can be introduced into the network, or an existing branch can be removed. This allows CGPDN to handle arbitrary numbers of inputs and outputs at run time.

The number of programs, that are run and transfer the potential from all active neurons to other active neurons is dependent on the number of active neural electrical components. Developmental programs determine the morphology of the neural network (i.e. the number of dendrite branches, axo-synapes and somae and how they are connected). The number of dendrites on each neuron is fixed, however the number of dendrite branches on each dendrite is variable and is determined by whether the developmental dendrite branch programs (DBL) in the past decided to eliminate or grow new branches. Every neuron is invested with a single axon, however, the number of axo-synapses attached to each axon is determined by whether the axo-synaptic branch program (ASL) in the past decided to grow or eliminate new axo-synapses. The number of neurons (initialized as a small randomly chosen number) is determined over time by whether soma developmental programs (SL) decided to replicate neurons or not (Fig. 8).

Whatever the number of programs that are run in the developing neural network, the size of the genotype is fixed and depends only on the sizes of the seven

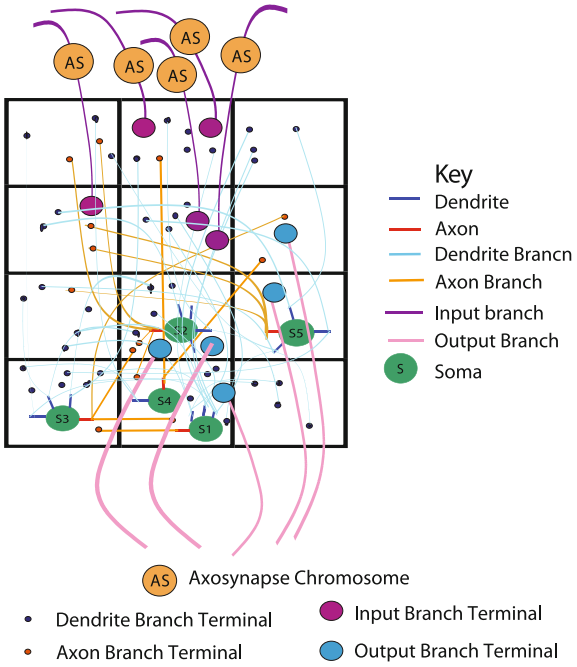

Fig. 7 Schematic illustration of a CGPDN defined over a 3×4 grid. The grid contains five neurons; each neuron has a number of dendrites with dendrite branches, and an axon with axon branches. Inputs are applied at five random locations in the grid using input axo-synapse branches by running axo-synaptic CGP programs. Outputs are taken from five random locations through output dendrite branches. Each *grid square* represents one location; the branches and soma are shown *spaced* for clarity. Each branch location is represented by where it terminates. Every location can have an arbitrary number of neurons and branches; there is no upper limit

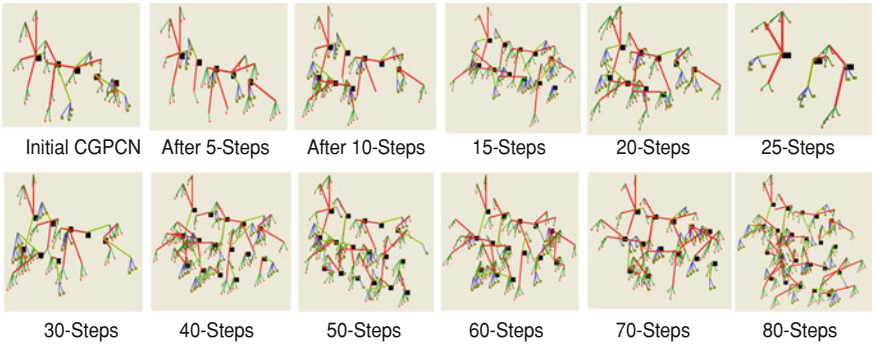

Fig. 8 Structural changes in a CGPDN network of a Wumpus World agent at different time steps. The network has five neurons at the start, and 21 neurons after completing 80 steps. *Black squares* are somae, *thick lines* (*red*) are dendrites, *yellowish green lines* are axons, *green lines* are dendrite branches, and *blue lines* show axon branches. Inputs and outputs are not shown

chromosomes that give rise to a network. This is one of the advantages of the developmental approach. A relatively simple collection of evolved programs can define an entire network of arbitrary complexity. The CGPDN model has been evaluated on a number of problems in artificial intelligence. Wumpus world [23], checkers [22, 25, 26] and maze solving [50]. Results show that the CGPDN produces networks that learn with experience (without further evolution). With structurally *different* networks they can recognize situations that have occurred before and cause the same actions. For instance, we observed that in a series of games of checkers, CGPDN players make appropriate, and often the same move, when a new game starts even though the neural network is different from the network that existed at the start of a previous game.

4 Self-Modifying CGP

Self-modifying Cartesian Genetic Programming (SMCGP) is a form of Genetic Programming founded on Cartesian Genetic Programming that is developmental in nature. In addition to the usual computational functions, it includes functions that can modify the program encoded in the genotype. This means that programs can be iterated to produce an infinite sequence of programs (phenotypes) from a single evolved genotype. It also allows programs to acquire more inputs and produce more outputs during this iteration.

Algorithm 2 gives a high-level overview of the process of mapping a genotype to a phenotype in SMCGP. The first stage of the mapping is the modification of the genotype. This happens through the use of evolutionary operators acting on the genotype. The developmental steps in the mapping are outlined in lines 3–8 of the algorithm. The first step is to make an exact copy of the genotype and call it the phenotype at iteration 0. After this, the self-modification operators are applied to produce the phenotype at the next iteration. Development stops when either a predefined iteration limit is achieved or it turns out that the phenotype has no self-modification operations that are active.

At each increment, the phenotype is evaluated and its fitness calculated. The underlying assumption here is that one is trying to solve a series of computational problems, rather than a single instance as is usual in GP. For instance, this might be a series of parity functions, ever-closer approximations to pi, or the natural numbers of the Fibonacci sequence. If the problem, however, has only a single instance (i.e. a classification problem), we can take a fixed number of iterations (either a user-defined parameter or evolved) and evaluate the single phenotype. Another possibility would be to iterate until no self-modification rules are active.

It is important to note that there are various ways in which there may be no active self-modification operations. Firstly, no self-modification operations may exist in the phenotype. Secondly, self-modification operations may be present but be non-coding. Thirdly, the self-modification operations may not be ‘activated’ when the

instructions encoded in the phenotype are executed. These various conditions will be discussed in the detailed description in the following sections.

Algorithm 2 Overview of genotype, phenotype and development

1: Generate genotype
2: Copy genotype to phenotype. Iteration, $i = 0$
3: **repeat**
4: Apply self-modification operations to phenotype i
5: increment i
6: Calculate fitness increment, f_i
7: **until** ((i equals number of iterations required) **OR** (No self-modification functions to do))
8: Evaluate phenotype fitness F from fitness increments, f_i

5 The Relation of SMCGP to CGP

The genetic representation in SMCGP has much in common with the representation used in CGP. The genotype encodes a graph (usually acyclic) that includes a list of node functions used and their connections. The arity of all functions is chosen to be equal to that of the largest-arity function in the function set. So, as in CGP, functions of lower arity ignore extraneous inputs. Although the form of SMCGP described here represents genotypes using a linear string of nodes, a two-dimensional form has been recently proposed [51].

5.1 Self-Modification Functions

The most significant difference between SMCGP and CGP is the addition of self-modification (SM) functions. These functions can be used in the genotype in the same manner as the more conventional computational operators, but at run time they provide different functionality. When the SMCGP phenotype is run, the nodes in the graph are parsed in a similar way to CGP. The graph is executed recursively by following nodes from the output nodes to the terminals (inputs). When computational functions are called, then—as usual—they operate on the data coming into the node.

When an SM node is called, the process is as follows. If an SM node is 'activated'; then its self-modification instructions are added to a list of pending manipulations which is called the *To-Do* list. The modifications in this list are then performed between iterations. In some implementations of SMCGP SM nodes are 'activated' if some numerical condition of its input data is obeyed (i.e. the first input is larger than the second). This makes the genotype-phenotype mapping data (and therefore context) dependent. Such a concept could be useful in ANNs (see later) as SM

Table 1 Examples of self-modification functions

Function name	Description
Delete (DEL)	Delete the nodes between $(P_0 + x)$ and $(P_1 + x)$
Add (ADD)	Add P_1 new random nodes after $(P_0 + x)$.
Move (MOV)	Move the nodes between $(P_0 + x)$ and $(P_1 + x)$ and insert after $(P_2 + x)$
Overwrite (OVR)	Copy the nodes between $(P_0 + x)$ and $(P_1 + x)$ to position $(P_2 + x)$, replacing existing nodes
Duplication (DUP)	Copy the nodes between $(P_0 + x)$ and $(P_1 + x)$ and insert after $(P_2 + x)$
Change connection (CHC)	Change the $(P_1 \bmod 3)$th connection of node P_0 to P_2
Change function (CHF)	Change the function of node P_0 to the function associated with P_1
Change argument (CHA)	Change the $(P_1 \bmod 3)$th argument of node P_0 to P_2

P_i are the evolved arguments of the self-modification functions; x represents the absolute position of the node in the graph, where the leftmost node has position 0. All additions are taken modulo (the number of nodes in the phenotype), this ensures they are always valid

operations could be activated, say, when the signal supplied to it was above a certain threshold (rather like the firing behaviour).

Many SM operators are imaginable, and Table 1 lists a few examples. In the table, we can see that the operators also require arguments. These come from the genotype and are described in Sect. 5.3. It is also worth noting that the indices for SM operations are defined relative to the current node.

5.2 Computational Functions

The computational functions used in SMCGP are typical of such functions in GP in general. They may be arithmetic operations such as addition, subtraction, multiplication, and division, or they may be mathematical functions such as *sin*, *exp* etc. In neural models one might choose sigmoid or hyperbolic tangent functions.

5.3 Arguments

Each node in the SMCGP genotype contains three integers defined in the range 0 to the maximum number of nodes in the genotype.[2] These numbers are evolved and are used in several ways by the SMCGP phenotype. The SM functions require several arguments to specify how graph modifications are to be carried out (see Table 1). These arguments are integers, and their values are calculated from the node's arguments.

During iteration of the genotype, the arguments can be altered by the SM function CHP ('change parameter'). This, in principle, allows storing of the state (i.e. a memory), since a phenotype could pass information to the phenotype at the next iteration through a collection of constant values.

[2] other reported implementations of SMCGP use floating point numbers.

5.4 Relative Addressing

The SM operators' ability to move, delete and duplicate sections of the graph means that the classical CGP approach of labelling nodes becomes cumbersome. Classical CGP uses *absolute addressing*, so that each node has an address and nodes reference each other using these addresses (this is what connection genes are—see Sect. 2).

To simplify the representation, absolute addresses were replaced with relative addresses. Now, instead of a node containing an absolute address of another node, it specifies how many nodes back from its position are required to make a connection. The connections in the genotype are now defined as positive integers that are greater than 0 (which prevents cycles).

When the graph is run, the interpreter can calculate where a node gets its input values from by just subtracting the connection value from the current address of the node. If the node addresses a value that is not in the graph (i.e. connects too far back), then a default value is returned (in the case of numeric applications this is 0).

The arguments of SM operators are also defined relative to the current node (see Table 1). The relative addressing allows subgraphs to be placed or duplicated in the graph whilst retaining their semantic validity. This means that subgraphs could represent the same subfunction, but act on different inputs. This can be done without recalculating any node addresses, thus maintaining validity of the whole graph. So subgraphs can be used as functions in the sense of the ADFs of standard GP.

5.5 Input and Output Nodes

Most, if not all, genetic programming implementations have a fixed number of inputs. This certainly makes sense when there is a constant or bounded number of inputs over the lifetime of a program. However, it prevents the program from scaling to larger problem sizes by increasing the number of inputs it uses—and this in turn may prevent general solutions from being found. Similarly, most GP systems have a fixed number of outputs. In tree-based GP, there is typically a single output. In classical CGP, a number of input nodes are placed at one end of the graph, and these are used as the starting point for the recursive interpretation of the program. To allow an arbitrary number of inputs and outputs, SMCGP introduces several new functions into the basic function set. These are shown in Table 2.

The interpreter now keeps track of an input pointer, which points to a particular input in the array of input values. Calling the function INPI returns the value that the pointer is currently on, and then moves the pointer to the next value. When the pointer runs out of inputs, it resets to the first input. Similarly, the DECI function returns an input but then moves the pointer to the previous value. Sometimes it may not be convenient or useful to move by only one input at a time, hence the SKPI is included. It moves the pointer a number of places. The number is arrived at by adding (modulo the number of program inputs) P_0 to the input pointer and then

Table 2 Input and output functions

Function	Operation
INCI	Return input pointed to by current_input, increment current_input
DECI	Return input pointed to by current_input, decrement current_input
SKPI	Return input pointed to by current_input, current_input = current_input + P_0
INCO	write data to current_output element of output_register, increment current_output
DECO	write data to current_output element of output_register, decrement current_output
SKPO	write data to current_output element of output_register, current_output = current_output + P_0

P_0 is the first argument gene; current_input wraps around so that when current_input equals the number of program inputs, current_input is set to zero; current_output also wraps around and writes outputs to an output register that has a number of elements equal to the number of program outputs. The output register is initialized with zeros

returning that input. The output functions work in a similar manner except that they write to an output register which has the same number of elements as the desired program outputs at that iteration. The register is filled with defaults values of zero. By duplicating any input and output functions the SMCGP phenotypes can acquire more inputs and outputs when they are iterated.

6 SMCGP Performance and Applications

The performance of SMCGP, in terms of the average number of genotype evaluations required to solve a problem, has been evaluated on a number of problems.[3] It has also been compared with other published results of other approaches [39] and also compared with standard CGP and a form of CGP with ADFS [44]. In all cases SMCGP solves easy instances of problems in a larger number of evaluations than other approaches, however it scales much better so that it solves harder instances in considerably less evaluations. Indeed it has also been shown to be able to produce mathematically provable general solutions to some problems. This has been done for parity, binary addition, and computing pi and e. Clearly SMCGP is a powerful and flexible evolutionary technique. It is natural, therefore, to investigate its utility in the evolution of developmental neural networks. We discuss this idea in the next section.

[3] This work has not involved ANNs.

Table 3 Suggested self-modification functions for ANNs

Function name	Description
Add connection (ADDC)	Add a random connection to nodes $(P_0 + x)$ to $(P_1 + x)$
Remove connection (REMC)	Remove a connection at random for all nodes $(P_0 + x)$ to $(P_1 + x)$
Increase weight (INCW)	Increase weight by a fixed percentage for all nodes $(P_0 + x)$ to $(P_1 + x)$
Derease weight (DECW)	Decrease weight by a fixed percentage for all nodes $(P_0 + x)$ to $(P_1 + x)$

P_i are the evolved arguments of the self-modification functions; x represents the absolute position of the node in the graph, where the leftmost node has position 0. All additions are taken modulo (the number of nodes in the phenotype), this ensures they are always valid

7 A Holocentric Model: SMCGP Artificial Neural Networks

It is relatively straightforward to adapt the SMCGP approach so that it constructs developmental neural networks. Here the function set can be a collection of conventionally used neuron functions (i.e. sigmoidal, hyperbolic tangent). New SM functions, in addition to those described in Table 1 need to be designed that are specific to ANNs. Suggested SM functions relevant to ANNs are shown in Table 3. With the complete set of functions (and assuming say a sigmoid computational node function) shown in Tables 1 and 3 one would have genotypes which were valid ANNs, that when iterated could produce ANNs with arbitrary topology and weight adjustment. The topology of the network would change through the action of embedded self-modification instructions (see Tables 1 and 2). Also weights at each iteration could be changed in the network through the action of embedded weight altering self-modification functions (see Table 3). This combined with the incremental fitness function described in Algorithm 2 could allow conventional ANNs to develop their own topology while performing a task. An additional attractive feature of the SMCGP approach to ANNs is that it should be relatively easy to adopt other models of artificial neurons, particularly spiking neural networks [52].

7.1 Process of Learning in SMCGP Artificial Neural Networks

The process of learning in SMCGP artificial neural networks would happen through a combination of topological changes (numbers of neurons and connections) and synaptic changes. It is important to note that evolved genotype would be simultanenously both (a) a valid ANN and (b) a set of rules that would produce an arbitrary sequence of ANNs. It would be a valid ANN because all computational nodes would be a standard neuron function (e.g. sigmoid) and all connections would have explicit weights in the genotype. However, on iteration (i.e. application of the embedded self-modification instructions) a new network would be produced. It may

turn out to be necessary to apply weight adjustment self-modification instructions more frequently than topological change inducing instructions. This could be done via a weight-adjustment algorithm (i.e. backpropagation or a search algorithm) operating within a single iteration (in the sense of Algorithm 2). However, by supplying a series of learning problems of different types, it should be possible to evolve an algorithm that when iterated, adjusts weights and makes topological changes in such a way that effective learning takes place. For instance, a number of the fitness increments referred to in Algorithm 2 could be acrued during training on a particular problem, before iterating the SMCGP phenotype on another learning problem. The number of iterations that are devoted to a particular learning problem would be a choice for the experimenter. In principle the manner in which fitness increments are calculated on a particular learning problem could be similar to the way fitness is evaluated in neuroevolutionary approaches to artificial neural networks. There are many issues here that remain for future research.

8 Neuro-Centric Versus Holocentric ANNs

Perhaps the greatest problem with neuro-centric ANNs is the complexity of the neurons. In the approach outlined above the neurons contained seven evolved chromosomes. The action of these chromosomes was to alter internal variables that resulted in electrical and morphological changes in the neurons. The values of these variables were used to determine increases in neural sub-component weights, neurite topology, transfer of potential, neuron and neurite replication and death and so on. To do this required a large number of additional rules many of which could be important for the success of the technique. The scientific study of such complex models is consequently difficult. If the model proves to be successful, what rules and assumptions are essential to that success? Though it is true that neuro-centric models can benefit from improvements in our understanding of neuroscience, this inevitably compounds the computational complexity problem. Science has not hitherto attempted to build models of complex systems using extremely complex primitives (i.e. a neuron), however it is clear that natural evolution has built a fundamental building block of life that is of staggering complexity. This is the cell. Only time will tell as to whether it is possible to extract from neuroscience a sufficiently simple and computationally efficient model of a neuron.

Holocentric models are therefore very attractive as they do not simulate many low level details of neurons. For instance, SMCGP can build a new computational network by carrying out simple graph altering operations on the genotype. HyperNEAT can generate whole neural networks by running a program that is a function of four variables. This makes them computationally more efficient. The drawback that these models have is that informing them from neuroscience is much more difficult. Holocentric models are much more abstract than neuro-centric models and are founded on high level assumptions. One of these was historically, the idea that synaptic weights were sufficient for general learning. The danger with making such

high-level assumptions is that they may produce models that have hitherto unknown limitations. Despite this it appears that abstracting developmental aspects of neuroscience and informing holocentric models with this appears to be a very promising direction.

9 Conclusions and Future Outlook

In this chapter we have outlined two approaches to developmental neural networks. One is neuro-centric and evolves complex computational models of neurons, the other is holocentric and use evolution and development at a whole network level. Both approaches have merits and potential drawbacks. The main issue with neuro-centric models is related to keeping the model complexity to be as small as possible and making the model computationally efficient. Holocentric models are simpler and more efficient but make high level assumptions that may ultimately restrict their power and generality. Both approaches are worthy of continued attention.

It is our view that one of the main aims of the neural networks should be to produce networks that are capable of general learning. General learning refers to an ability to learn in multiple task domains without the occurrence of interference. A fundamental problem in creating general learning systems is the *encoding problem*. This is where the data that is fed into the neural networks has to be specifically encoded for each problem. Biological brains avoid this by using sensors to acquire information about the world and actuators to change it. We suggest that such universal representations will be required in order for developmental artificial neural networks to show general learning. Thus we feel that general learning systems can only be arrived at through systems that utilize sensory data from the world. Essentially this means that such systems need to be implemented on physical robots. This gives us an even greater incentive to construct highly efficient models of neural networks.

References

1. W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophy. **5**, 115–133 (1943)
2. E.R. Kandel, J.H. Schwartz, T.M. Jessell, *Principles of Neural Science*, 4th edn. (McGraw-Hill, New York, 2000)
3. R.M. French, Catastrophic forgetting in connectionist networks: causes, consequences and solutions. Trends Cogn. Sci. **3**(4), 128–135 (1999)
4. M. McCloskey, N.J. Cohen, Catastrophic interference in connectionist networks: the sequential learning problem. Psychol. Learn. Motiv. **24**, 109–165 (1989)
5. R. Ratcliff, Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychol. Rev. **97**, 285–308 (1990)
6. S. Judd, On the complexity of loading shallow neural networks. J. Complex. **4**, 177–192 (1988)
7. E.B. Baum, A proposal for more powerful learning algorithms. Neural Comput. **1**, 201–207 (1989)

8. S.E. Fahlman, C. Lebiere, *The cascade-correlation architecture*, ed. by D.S. Touretzky. Advances in Neural Information Processing Systems (Morgan Kaufmann, San Mateo, 1990)
9. M. Frean, The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Comput. **2**, 198–209 (1990)
10. P.T. Quinlan, Structural change and development in real and artificial networks. Neural Netw. **11**, 577–599 (1998)
11. P.T. Quinlan (ed.), *Connectionist Models of Development* (Psychology Press, New York, 2003)
12. J.F. Miller, G.M. Khan, Where is the brain inside the brain? on why artificial neural networks should be developmental. Memet. Comput. **3**(3), 217–228 (2011)
13. J.R. Smythies, *The Dynamic Neuron* (MIT Press, Cambridge, 2002)
14. F. Valverde, Rate and extent of recovery from dark rearing in the visual cortex of the mouse. Brain Res. **33**, 1–11 (1971)
15. J.A. Kleim, E. Lussnig, E.R. Schwartz, T.A. Comery, W.T. Greenough, Synaptogenesis and fos expression in the motor cortex of the adult rat after motor skill learning. J. Neurosci **16**, 4529–4535 (1996)
16. J.A. Kleim, K. Vij, D.H. Ballard, W.T. Greenough, Learning-dependent synaptic modifications in the cerebellar cortex of the adult rat persist for at least four weeks. J. Neurosci **17**, 717–721 (1997)
17. M.L. Mustroph, S. Chen, S.C. Desai, E.B. Cay, E.K. Deyoung, J.S. Rhodes. Aerobic exercise is the critical variable in an enriched environment that increases hippocampal neurogenesis and water maze learning in male C57BL/6J mice. Neuroscience (2012), Epub ahead of print
18. A.D. Tramontin, E. Brenowitz, Seasonal plasticity in the adult brain. Trends Neurosci. **23**, 251–258 (2000)
19. E.A. Maguire, D.G. Gadian, I.S. Johnsrude, C.D. Good, J. Ashburner, R.S.J. Frackowiak, C.D. Frith, Navigation-related structural change in the hippocampi of taxi drivers. PNAS **97**, 4398–4403 (2000)
20. S. Rose, *The Making of Memory: From Molecules to Mind* (Vintage, London, 2003)
21. A.S. Dekaban, D. Sadowsky, Changes in brain weights during the span of human life. Ann. Neurol. **4**, 345–356 (1978)
22. G.M. Khan, J.F. Miller, *Evolution of cartesian genetic programs capable of learning*, ed. by F. Rothlauf. Conference on Genetic and Evolutionary Computation (GECCO) (ACM, 2009) pp. 707–714
23. G.M. Khan, J.F. Miller, D.M. Halliday, *Coevolution of intelligent agents using Cartesian genetic programming*. Conference on Genetic and Evolutionary Computation (GECCO) (2007), pp. 269–276
24. G.M. Khan, J.F. Miller, D.M. Halliday, *Breaking the synaptic dogma: Evolving a neuro-inspired developmental network*, ed. by X. Li, M. Kirley, M. Zhang, D.G. Green, V. Ciesielski, H.A. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K.C. Tan, J. Branke, Y. Shi. Simulated Evolution and Learning, 7th International Conference, SEAL 2008, Melbourne, Australia, December 7–10, 2008. Proceedings, volume 5361 of Lecture Notes in Computer Science (Springer, 2008), pp. 11–20
25. G.M. Khan, J.F. Miller, D.M. Halliday, *Coevolution of neuro-developmental programs that play checkers*, ed. by G. Hornby, L. Sekanina, P.C. Haddow. Evolvable Systems: From Biology to Hardware, 8th International Conference, ICES 2008, Prague, Czech Republic, September 21–24, 2008. Proceedings, volume 5216 of Lecture Notes in Computer Science (Springer, 2008), pp. 352–361
26. G.M. Khan, J.F. Miller, D.M. Halliday, *Developing neural structure of two agents that play checkers using cartesian genetic programming*, ed. by C. Ryan, M. Keijzer. Conference on Genetic and Evolutionary Computation (GECCO) Companion Material (ACM, 2008), pp. 2169–2174
27. G.M. Khan, J.F. Miller, D.M. Halliday, In search of intelligent genes: the cartesian genetic programming computational neuron (cgpcn), in *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2009, Trondheim, Norway, 18–21 May, 2009 (IEEE, 2009), pp. 574–581

28. G.M. Khan, J.F. Miller, D.M. Halliday, Evolution of cartesian genetic programs for development of learning neural architecture. Evol. Comput. **19**(3), 469–523 (2011)
29. K.O. Stanley, D.B. D'Ambrosio, J. Gauci, A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**, 185–212 (2009)
30. F. Gruau, Automatic definition of modular neural networks. Adapt. Behav. **3**, 151–183 (1994)
31. J.R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection* (MIT Press, Cambridge, 1992)
32. J.F. Miller, An *Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach*. Conference on Genetic and Evolutionary Computation (GECCO) (Morgan Kaufmann, 1999), pp. 1135–1142
33. J.F. Miller, P. Thomson, Cartesian Genetic Programming, in *Proceedings of the European Conference on Genetic Programming*, vol. 1802 of LNCS (Springer, 2000), pp. 121–132
34. S. Harding, J.F. Miller, W. Banzhaf, *A survey of self modifying CGP*, ed. by R. Riolo, T. McConaghy, E. Vladislavleda. Genetic Programming Theory and Practice VIII, 2010 (Springer, 2010) pp. 91–107
35. S. Harding, J.F. Miller, W. Banzhaf, Self-modifying Cartesian Genetic Programming, in *Proceedings of the Genetic and Evolutionary Computation Conference* (2007), pp. 1021–1028
36. S. Harding, J.F. Miller, W. Banzhaf, *Evolution, development and learning using self-modifying cartesian genetic programming*, ed. by F. Rothlauf. Conference on Genetic and Evolutionary Computation (GECCO) (ACM, 2009), pp. 699–706
37. S. Harding, J.F. Miller, W. Banzhaf, *Self modifying cartesian genetic programming: Fibonacci, squares, regression and summing*, ed. by L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, M. Ebner. Genetic Programming, 12th European Conference, EuroGP 2009, Tübingen, Germany, April 15–17, 2009, Proceedings, volume 5481 of Lecture Notes in Computer Science (Springer, 2009), pp. 133–144
38. S. Harding, J.F. Miller, W. Banzhaf, Self modifying cartesian genetic programming: Parity, in *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2009, Trondheim, Norway, 18–21 May, IEEE, 2009 (2009), pp. 285–292
39. S. Harding, J.F. Miller, W. Banzhaf, Developments in cartesian genetic programming: self-modifying cgp. Genet. Program. Evolvable Mach. **11**(3–4), 397–439 (2010)
40. S. Harding, J.F. Miller, W. Banzhaf, *Self modifying cartesian genetic programming: finding algorithms that calculate pi and e to arbitrary precision*, ed. by M. Pelikan, J. Branke. Conference on Genetic and Evolutionary Computation (GECCO) (ACM, 2010), pp. 579–586
41. M.M. Khan, G.M. Khan, J.F. Miller, Efficient representation of recurrent neural networks for Markovian/Non-Markovian non-linear control problems, ed. by A.E. Hassanien, A. Abraham, F. Marcelloni, H. Hagras, M. Antonelli, T.-P. Hong, in *Proceedings of the International Conference on Intelligent Systems Design and Applications* (IEEE, 2010), pp. 615–620
42. M.M. Khan, G.M. Khan, J.F. Miller, Evolution of neural networks using cartesian genetic programming, in *Proceedings of the IEEE Congress on Evolutionary Computation*, CEC 2010, Barcelona, Spain, 18–23 July 2010 (IEEE, 2010)
43. M.M Khan, G.M. Khan, J.F. Miller, Evolution of optimal anns for non-linear control problems using cartesian genetic programming, ed. by H.R. Arabnia, D. de la Fuente, E.B. Kozerenko, J.A. Olivas, R. Chang, P.M. LaMonica, R.A. Liuzzi, A.M.G. Solo, in *Proceedings of the 2010 International Conference on Artificial Intelligence*, ICAI 2010, July 12–15, 2010, Las Vegas Nevada, USA, vol. 2 (CSREA Press, 2010), pp. 339–346
44. J.A. Walker, J.F. Miller, The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. IEEE Trans. Evolut. Comput. **12**(4), 397–417 (2008)
45. J.F. Miller (ed.), *Cartesian Genetic Programming*. Natural Computing Series (Springer, Berlin, 2011)
46. I. Rechenberg, Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Ph.D. thesis, Technical University of Berlin, Germany, (1971)
47. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evolut. Comput. **10**(2), 167–174 (2006)

48. V.K. Vassilev, J.F. Miller, *The Advantages of Landscape Neutrality in Digital Circuit Evolution*. International Conference on Evolvable Systems, vol. 1801 of LNCS (Springer, 2000), pp. 252–263
49. T. Yu, J.F. Miller, Neutrality and the evolvability of Boolean function landscape, in *Proceedings of the European Conference on Genetic Programming*, vol. 2038 of LNCS (Springer, 2001), pp. 204–217
50. G.M. Khan, J.F Miller, Solving mazes using an artificial developmental neuron, in *Proceedings of the Conference on Artificial Life (ALIFE) XII* (MIT Press, 2010), pp. 241–248
51. S. Harding, J.F. Miller, W. Banzhaf, *SMCGP2: Self-modifying Cartesian Genetic Programming in Two Dimensions*. Conference on Genetic and Evolutionary Computation (GECCO) (ACM, 2011), pp. 1491–1498
52. W. Gerstner, W.M. Kistler, *Spiking Neuron Models* (Cambridge University Press, Cambridge, 2002)

Chapter 9
Artificial Evolution of Plastic Neural Networks: A Few Key Concepts

Jean-Baptiste Mouret and Paul Tonelli

Abstract This chapter introduces a hierarchy of concepts to classify the goals and the methods used in articles that mix neuro-evolution and synaptic plasticity. We propose definitions of "behavioral robustness" and oppose it to "reward-based behavioral changes"; we then distinguish the switch between behaviors and the acquisition of new behaviors. Last, we formalize the concept of "synaptic General Learning Abilities" (sGLA) and that of "synaptic Transitive learning Abilities (sTLA)". For each concept, we review the literature to identify the main experimental setups and the typical studies.

1 Introduction

The abilities of animals to adapt to new environments is one of the most fascinating aspects of nature and it may be what most clearly separates animals from current machines. Natural adaptive processes are classically divided into three main categories, each of them having been a continuous source of inspiration in artificial intelligence and robotics [8]: evolution, development and learning. While studying each of these processes independently have been widely successful, there is a growing interest in understanding how they benefit from each other.

In particular, a large amount of work has been devoted to understanding both the biology of learning (e.g. [19, 29]) and the design of learning algorithms for artificial neural networks (e.g. [9]); concurrently, evolution-inspired algorithms have been

J.-B. Mouret (✉)
Institut des Systèmes Intelligents et de Robotique (ISIR), UMR 7222,
Sorbonne Universités, UPMC Univ Paris, 06, F-75005 Paris, France
e-mail: mouret@isir.upmc.fr

P. Tonelli
UMR 7222, ISIR, F-75005 Paris, France
e-mail: tonelli@isir.upmc.fr

T. Kowaliw et al. (eds.), *Growing Adaptive Machines*,
Studies in Computational Intelligence 557, DOI: 10.1007/978-3-642-55337-0_9,

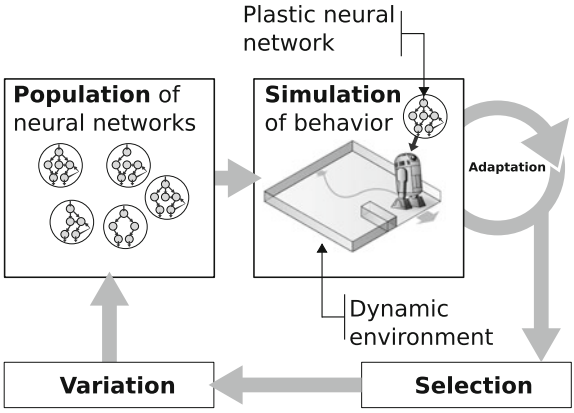

Fig. 1 The artificial evolution of plastic neural networks relies on the classic evolutionary loop used in neuro-evolution. The algorithm starts with a population of genotypes that are thereafter developed into plastic neural networks. The topology of the neural network is sometimes evolved [17, 18, 21–24, 28]. Typical plastic neural networks use a variant of the Hebb's rule to adapt the weight during the "lifetime" of the agent. The fitness of the agent is most of the time evaluated in a dynamic environment that requires the agent to adapt its behavior. The agent is therefore usually directly selected for its adaptive abilities

successfully employed to automatically design small "nervous systems" for robots [7, 10, 12, 13, 25, 26], sometimes by taking inspiration from development processes [10, 13, 16, 25]. A comparatively few articles proposed to combine the artificial evolution of neural networks with synaptic plasticity to evolve artificial agents that can adapt their "artificial nervous system" during their "lifetime" [7, 15–18, 21–23, 28, 30] (Fig.1). However, the analysis of these articles shows that they often address different challenges in very different situations, while using the same terminology (e.g. "learning", "robustness" or "generalization").

The goal of the present article is to provide a set of definitions to make as clear as possible current and future work that involve the evolution of such plastic artificial neural networks (ANNs) to control agents (simulated or real robots). While some definitions and some distinctions are novel, the main contribution of the present chapter is to isolate each concept and to present them in a coherent framework. For each definition, we will provide examples of typical setups and current results. Figure 2 displays the hierarchy of the concepts that will be introduced; it can serve as a guide to the chapter.

2 Synaptic Plasticity

In neuroscience, plasticity (or neuroplasticity) is the ability of the brain and nervous systems to change structurally and functionally as a result of their interaction with

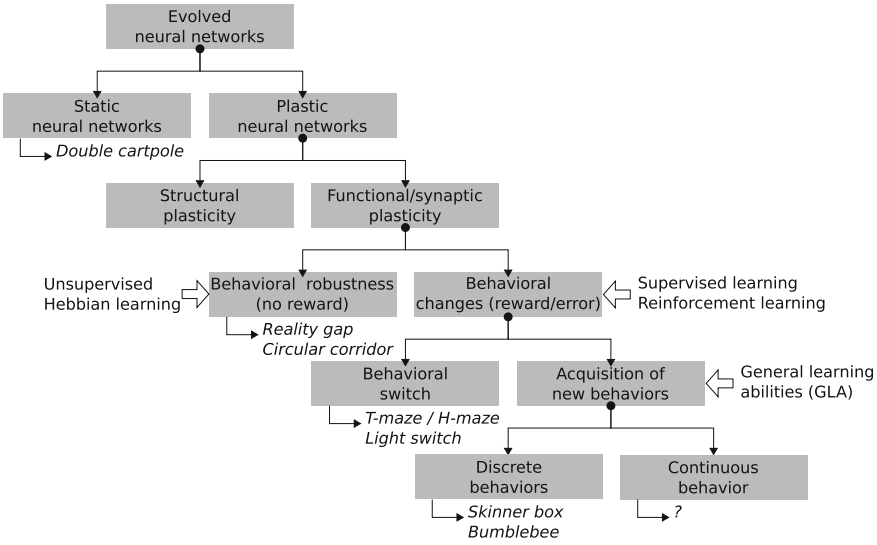

Fig. 2 Hierarchy of concepts described the present chapter. See text for a definition of each *grey box*

the environment. Plasticity is typically observed during phases of development and learning. Trappenberg [29] defines two kinds of plasticity: structural plasticity and synaptic (or functional) plasticity.

Definition 1 (*Structural plasticity*) Structural plasticity is the mechanism describing generation of new connections and thereby redefining the topology of the network.

Definition 2 (*Synaptic plasticity*) Synaptic plasticity is the mechanism of changing strength values of existing connections. It is sometimes termed "functional plasticity" [29].

Nolfi et al. [16] investigated *structural plasticity* in a system in which the genotype contained developmental instructions for the construction of a neural network. Genes specified (1) the position of each neuron and (2) instructions that described how axons and branching segments grew. These instructions were executed when a neuron was sufficiently stimulated by its surrounding neurons and by the agent's environment. The authors observed different phenotypes when the same genotype was used in two different environments and concluded that their approach increased the adaptive capabilities of their organisms. Several other authors evolved neural networks while letting them grow axons depending on their location (e.g. [10, 25]) but the environment was not taken into account.

Most research on the evolution of plastic neural networks instead focused on *synaptic plasticity* [2, 6, 14, 30], maybe because of the prominence of learning algorithms that only adapt weights in the machine learning literature. Most of the works that do not rely on machine learning algorithms (e.g. the backpropagation

algorithm) [3, 15] use variants of the "Hebb's rule" [2, 6, 17, 28, 30], which posits that the simultaneous activation of two neurons strengthens the synapse that link them.

Definition 3 (*Hebb's rule*) Let us denote by i and j two neurons[1], a_i and a_j their respective activation level, w_{ij} the synaptic weight of the connection from i to j and η a learning rate that describes how fast the change occurs. According to Hebb's rule, w_{ij} should be modified as follows:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \tag{1}$$

$$\Delta w_{ij} = \eta \cdot a_i \cdot a_j \tag{2}$$

Hebb's rule is often extended to include more complex combinations of pre- and post-synaptic activities [2, 6, 17, 29, 30].

Definition 4 (*Extended Hebbian rule*)

$$\Delta w_{ij} = f(a_i, a_j, w_{ij}) \tag{3}$$

Many different $f()$ have been investigated; one of the simplest extended Hebbian rule consists in linearly combining pre- and post-synaptic activities [14, 17, 21]:

$$\Delta w_{ij} = A \cdot a_i \cdot a_j + B \cdot a_i + C \cdot a_j + D \tag{4}$$

where A,B, C and D are four real numbers. Several rules can be mixed in the same neural networks, as Urzelai and Floreano did it when let evolve the kind of rules for each synapse in a fully connected, fully plastic neural networks [30].

A synapse can also be strengthened or weakened as a result of the firing of a third, modulatory inter-neuron (e.g. dopaminergic neurons) [1, 14, 22]. To reflect this phenomenon, two kinds of neurons can be distinguished: modulatory neurons and modulated neurons. Inputs of each neuron are divided into modulatory inputs and signal inputs; the sum of the modulatory inputs of j governs the modulation of the all non-modulatory connections to j:

Definition 5 (*Modulated Hebbian rule*) Let us denote by $I_j^{(m)}$ the set of modulatory inputs of neuron j and by $I_s^{(j)}$ the set of non-modulatory inputs. Each incoming connection of neuron j is modified as follows:

[1] We focus our discussion on classic neurons (as used in classic machine learning) and population-based models of neurons (e.g. leaky integrators) because they are the neuron models that are used by most of the community. Spiking neuron models can make use of other plasticity mechanisms (e.g. STDP) that will not be described here.

$$m_j = \tanh\left(\sum_{k \in I_j^{(m)}} w_{kj} a_k\right) \tag{5}$$

$$\forall\ i \in I_s^{(j)}, \Delta w_{ij} = m_j \cdot f(a_i, a_j, w_{ij}) \tag{6}$$

In addition to its biological realism, this weight adaptation rule makes easier to use rewards signals (for instance, plasticity could be enabled only when a reward signal is on). It also leads to networks in which only a part of the synapses are changed during the day-to-day life of the agent. These two features make such networks match more closely some of the current actor-critic models of reinforcement learning used in computational neuroscience [19].

Modulated Hebbian plasticity has been used several times when evolving plastic neural networks [11, 14, 18, 21, 22]. In these simulations, experiments in reward-based scenarios where modulatory neurons were enabled achieved better learning in comparison to those where modulatory neurons were disabled [21].

3 Robustness and Reward-Based Scenarios

A major goal when evolving neuro-controllers is to evolve neural networks that keep performing the same optimal (or pseudo-optimal) behavior when their morphology or their environment change. For instance, a robot can be damaged, gears can wear out over time or the light conditions can change: in all these situations, it is desirable for an evolved controller to compensate these changes by adapting itself; we will call this ability *behavioral robustness*.

Definition 6 (*Behavioral robustness*) An agent displays behavioral robustness when it keeps the same qualitative behavior, notwithstanding environmental and morphological changes. Behavioral robustness does not usually involve a reward/ punishment system.

In a typical work that combines synaptic plasticity, evolution and behavioral robustness, Urzelai and Floreano [30] evolved neuro-controllers with plastic synapses to solve a light-switching task *in which there was no reward*; they then investigated whether these controllers were able to cope with four types of environmental changes: new sensory appearances, transfer from simulations to physical robots, transfer across different robotic platforms and re-arrangement of environmental layout. The plastic ANNs were able to overcome these four kinds of change, contrary to a classic ANN with fixed weights.

However, as highlighted by Urzelai and Floreano, "these behaviors were not learned in the classic meaning of the term because they were not necessarily retained forever". Actually, synaptic weights were continuously changing such that the robot performed several sub-behaviors in sequence; the evolutionary algorithm therefore opportunistically used plasticity to enhance the dynamic power of the ANN. These

high-frequency changes of synaptic weights appear different from what we observe in natural system (in particular in the basal ganglia), in which synaptic weights tend to hold the same value for a long period, once stabilized [5, 27].

Besides robustness, an even more desirable property for an evolved agent is the ability to change its behavior according to external stimuli and, in particular, according to rewards and punishments. For instance, one can imagine a robot in a T-maze that must go to the end of the maze where a reward has been put [17, 18, 21]. The robot should first randomly try different trajectories. Then, once the reward has been found a few times, the robot should go directly to the reward. Nonetheless, if the reward is moved somewhere else, the robot should change its behavior to match the new position of the reward. Once the robot has found the optimal behavior (the behavior that maximizes the reward), the synaptic weights of its controller should not change anymore. This ability to adapt in a reward-based scenario can be more formally defined as follows:

Definition 7 (*Behavioral change*) A plastic agent is capable of behavioral changes in a reward-based scenario if and only if:

- a change of reward makes it adopt a qualitatively new behavior;
- the synaptic weights do not significantly change once an optimal behavior has been reached.

Notable setups in which authors evolved plastic neuro-controllers for behavioral changes are the T-maze [17, 18, 21], the bumblebee foraging task [14], the "dangerous foraging task" [24] and the Skinner box [28, 29].

4 Learning Abilities in Discrete Environment

The main challenge when evolving plastic agents for behavioral change is to make them able to learn *new behaviors* in *unknown situations* and, in particular, in situations that have never been encountered during the evolutionary process. Put differently, selecting agents for their abilities to switch between alternatives is not sufficient; the evolved agent must also be placed in completely new situations to assess its ability to find an optimal behavior in a situation for which it has never been selected.

We previously introduced a theoretical framework to characterize and analyze the learning abilities of evolved plastic neural networks [28, 29]; we will rely on this framework in the remainder of this chapter. For the sake of simplicity, we focus on a discrete world, with discrete stimuli and discrete actions. The canonical setup, inspired by experiments in operant conditioning, is the Skinner Box [20]: an agent is placed in a cage with n stimuli (lights), m actions (levers), positive rewards (food) and punishments (electric shocks). The goal of the agent is to learn the right associations between each stimulus and each action. This task encompasses most discrete reward-based scenarios (Fig. 3). For instance, the discrete T-maze experiment [17, 18, 21–23] can be described as a special case of a Skinner box.

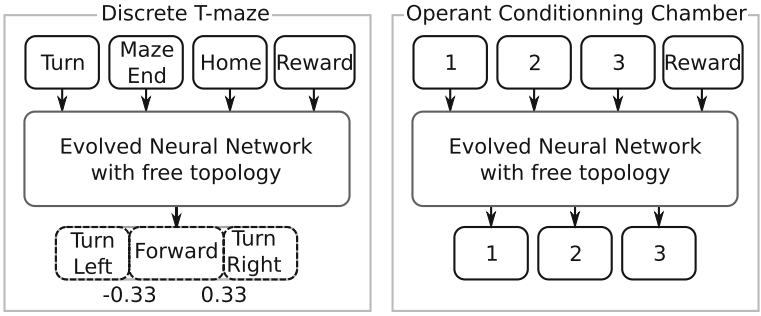

Fig. 3 Learning the best-rewarding behavior in a discrete T-maze is equivalent to a Skinner box (Operant Conditioning Chamber, *left*): in both cases, the challenge is to associate the right stimulus to the right action

More formally, an evolved neural network $N(I, \lambda)$ must adapt several synaptic weights $\lambda \in \mathbb{R}^z$ such that each input pattern $I \in [0, 1]^n$ is associated to the best rewarded output vector $K \in [0, 1]^m$. The adaptation is performed by a learning function such that $\lambda = g(\lambda_r, I, R_{I,K})$, where λ_r is a random vector in $\mathbb{R}^z$ and $R_{I,K}$ the reward function. These notations lead to the following definitions:

Definition 8 (*Association set*) An association set $A = \{(I_1, K_1), \ldots, (I_n, K_n)\}$ is a list of associations that covers all the possible input patterns. The set of all association sets is denoted $\mathbb{A}$.

Definition 9 (*Fitness association set*) The fitness association set $\mathbb{F}_{\mathbb{A}} = \{A_1 \cdots A_k\}$ is the set of the association sets that are used during the fitness evaluation.

For a given topology, some association sets may not be learnable by only changing synaptic weights. This case occurs in particular when the topology of neural networks are evolved: if there is no selective pressure to maintain a connection, it can easily disappear; but this connection may be required to learn a similar but different association set. Some association sets may also be not learnable because they require specific topologies. For instance, the XOR function requires a hidden layer of neurons to be computed.

Definition 10 (*Learnable set*) Given a suitable reward function $R_{I,K}$, an association set $A \in \mathbb{A}$ is said to be learnable by the neural network N, if and only if $\forall\ \lambda_r \in \mathbb{R}^z$ and $\forall\ (I, K) \in A, \exists\ \lambda = g(\lambda_r, I, R_{I,K})$ such that $N(I, \lambda) = K$. The set of all learnable sets for N is denoted $\mathbb{L}_N$.

Definition 11 (*sGLA*) A plastic ANN is said to possess synaptic General Learning Abilities (sGLA) if and only if $\forall\ A \in \mathbb{A}, A \in \mathbb{L}_N$.

Although it does not use Hebbian learning, the multi-layer perceptron with the back-propagation algorithm is an example of a neural network with synaptic General

Learning Abilities. At the opposite end, a neural network in which each input is connected to only one output can learn only one association set.

To evolve a plastic ANN with sGLA, the simplest method is to check the learnability of each association set during the fitness evaluation; that is, to set the fitness association set equal to the set of all the association sets ($\mathbb{F}_{\mathbb{A}} = \mathbb{A}$). This approach has often been followed by authors who evolved agents to solve the T-maze task [17, 18, 21–23]. We propose to call such approaches the *evolution of behavioral switches* to distinguish it from the evolution of more general learning abilities.

Definition 12 (*Evolution of behavioral switches*) $\mathbb{F}_{\mathbb{A}} = \mathbb{A}$.

However, a plastic ANN that can cope with unknown situations must have sGLA while only a subset of the possible association sets (i.e. a subset of problems from the same problem class) has been used during the evolutionary process.

Definition 13 (*Evolution of sGLA for unknown situations*) $|\mathbb{F}_{\mathbb{A}}| < |\mathbb{A}|$ and $\forall\, A \in \mathbb{A}$, $A \in \mathbb{L}_N$.

At first sight, nature relies on the long lifetime of animals (compared to the "lifetime" of artificial agents) and on the large size of the populations to obtain a stochastic evaluation of virtually every possible scenarios. This probably explains why most authors tried to obtain agents with sGLA by using a large, often randomized subset of the association sets in the fitness association set. In supervised learning, Chalmers [3] assessed how well an evolved plastic ANN can cope with situations never encountered during the evolution. In his experiments, he evolved the learning rule for a small single-layer ANN (five inputs, one output) and his analysis showed that at least 10 sets of input/output patterns (among 30 possible sets) were required to evolve an algorithm that correctly learns on 10 unknown sets. In reinforcement learning, Niv et al. [14] evolved plastic ANNs to solve a bumblebee-inspired foraging task in which simulated bees must select flowers by recognizing their color. To promote general learning abilities, they randomly assigned rewards to colors at each generation and they showed that the resulting ANNs successfully learned unknown color/reward associations. In the "dangerous foraging task", Stanley et al. [24] similarly randomized the parameters of the fitness function to avoid overspecialized behaviors.

However, the encoding and the development process may also play a key role in allowing the adaptation to situations which have never been encountered before [28, 29]. Intuitively, a very regular network may repeat the same adaptation structure many times whereas it was only required once by the fitness; it could therefore "propagate" the adaptation structure. Since most developmental encoding are designed to generate very regular structures [4, 13, 28, 29], using such encodings could substantially reduce the number of evaluations required to obtain general learning abilities. In the ideal case, we should be able to show that the developmental process implies that if a few association sets have been successfully learned, then all the other sets have a high probability of being learnable. Such networks will be said to possess "synaptic Transitive Learning Abilities".

Definition 14 (*sTLA*) Let us denote by $\mathbb{T}_N$ a subset of the learnable association set $\mathbb{A}$. A plastic ANN is said to possess synaptic Transitive Learning Abilities (sTLA) if and only if $\exists\, \mathbb{T}_N \subset \mathbb{A}$ such that the following implication is true:

$$\mathbb{T}_N \subset \mathbb{L}_N \Rightarrow \mathbb{L}_N = \mathbb{A}$$

$p = |\mathbb{T}_N|$ will be called the "sTLA-level".

Definition 15 (*Optimal-sTLA*) A plastic ANN is said to possess Optimal synaptic Transitive Learning Abilities (optimal-sTLA) if and only if it possesses sTLA and $|\mathbb{T}_N| = 1$.

The sTLA-level of certain families of topologies (i.e. topologies generated by a specific genetic encoding) can possibly be computed theoretically. It can also be easily evaluated by a succession of evolutionary experiments: (1) select p association sets; (2) evolve ANNs that successfully learns the p association sets; (3) check the sGLA of optimal ANNs; (4) if optimal ANNs do not possess sGLA, then increase p and start again.

Using this method, Tonelli and Mouret [28, 29] showed that a very regular map-based encoding proposed in [13] have a TLA-level or 1 or 2. Preliminary experiments suggest that other generative encodings such as HyperNEAT [4, 25] could also possess a low TLA-level [29]. Overall, the concept of sTLA highlights how evolution, learning and development are interwoven.

Last, the authors are not aware of any definition that would of an equivalent of the concept of GLA for continuous world and behaviors.

5 Concluding Remarks

With the rise of computing power, it is now easy to simulate artificial agents for enough time for them to learn and to evolve; this allows the study of well-defined scientific questions with modern experimental and statistical techniques. Nevertheless, future work in this direction will have to precisely define what they work on: Do they aim at behavioral robustness or at behavioral change? How do they evaluate the general learning abilities of the evolved agents? Do the evolved neural network manage to learn in unknown situations? What is the role of the encoding in the final result? The definitions proposed in the present chapter will hopefully help to design a methodology to answer such questions.

The present work also highlights open questions and avenues for future research:

- Should future work focus more on structural plasticity? This approach to plasticity may be more complex but it may also allow agents to learn new skills without forgetting the old ones (because the previous structure is not deleted).
- How to evaluate learning abilities in continuous world and with continuous behaviors?
- What are the links between encodings, plasticity and learnability? [28, 29] provides first answers but only for simple and discrete scenarios.

Acknowledgments This work was funded by the EvoNeuro project (ANR-09-EMER-005-01) and the Creadapt project (ANR-12-JS03-0009).

References

1. C.H. Bailey, M. Giustetto, Y.Y. Huang, R.D. Hawkins, E.R. Kandel, Is heterosynaptic modulation essential for stabilizing Hebbian plasticity and memory? Nature Rev. Neurosci. **1**(1), 11–20 (2000)
2. J. Blynel, D. Floreano, Levels of dynamics and adaptive behavior in evolutionary neural controllers, in *Conference on Simulation of Adaptive Behavior (SAB)* (2002), pp. 272–281
3. D.J. Chalmers, *The Evolution of Learning: An Experiment in Genetic Connectionism* (Connectionist Models Summer School, 1990)
4. J. Clune, K.O. Stanley, R.T. Pennock, C. Ofria, On the performance of indirect encoding across the continuum of regularity. IEEE Trans. Evol. Comput. **15**(3), 346–367 (2011)
5. N.D. Daw, Y. Niv, P. Dayan, Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. Nature Neurosci. **8**(12), 11–1704 (2005)
6. D. Floreano, Evolution of plastic neurocontrollers for situated agents, in *Conference on Simulation of Adaptive Behavior (SAB)* (1996)
7. D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning. Evol. Intell. **1**(1), 47–62 (2008)
8. D. Floreano, C. Mattiussi, *Bio-inspired Artificial Intelligence: Theories, Methods, and Technologies* (The MIT Press, 2008)
9. S. Haykin, *Neural networks: a comprehensive foundation* (Prentice Hall, Upper Saddle River, 1999)
10. J. Kodjabachian, J.-A. Meyer, Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. IEEE Trans. Neural Networks **9**(5), 796–812 (1998)
11. T. Kondo, Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control. Appl. Soft Comput. **7**(1), 189–202 (2007)
12. J.-B. Mouret, S. Doncieux, Using behavioral exploration objectives to solve deceptive problems in neuro-evolution, in *Conference on genetic and evolutionary computation (GECCO)* (2009)
13. J.-B. Mouret, S. Doncieux, B. Girard, Importing the computational neuroscience toolbox into neuro-evolution—application to basal ganglia, in *Conference on genetic and evolutionary computation (GECCO)* (2010)
14. Y. Niv, D. Joel, I. Meilijson, E. Ruppin, Evolution of reinforcement learning in uncertain environments: a simple explanation for complex foraging behaviors. Adapt. Behav. **10**(1), 5–24 (2002)
15. S. Nolfi, How learning and evolution interact: the case of a learning task which differs from the evolutionary task. Adapt. Behav. **4**(1), 81–84 (1999)
16. S. Nolfi, O. Miglino, D. Parisi, Phenotypic plasticity in evolving neural networks. in *From Perception to Action Conference* (IEEE, 1994), pp. 146–157
17. S. Risi, K.O. Stanley, Indirectly encoding neural plasticity as a pattern of local rules, in *Conference on Simulation of Adaptive Behavior (SAB)* (2010)
18. S. Risi, S.D. Vanderbleek, C.E. Hughes, K.O. Stanley, How novelty search escapes the deceptive trap of learning to learn, in *Conference on genetic and evolutionary computation (GECCO)* (2009)
19. W. Schultz, P. Dayan, P.R. Montague, A neural substrate of prediction and reward. Science **275**(5306), 1593–1599 (1997)
20. B.F. Skinner, Operant behavior. Am. Psychol. **18**(8), 503 (1963)
21. A. Soltoggio, J.A. Bullinaria, C. Mattiussi, P. Dürr, D. Floreano, Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. Artif. Life **11**, 569 (2008)

22. A. Soltoggio, P. Dürr, C. Mattiussi, D. Floreano, Evolving neuromodulatory topologies for reinforcement learning-like problems, in *IEEE Congress on Evolutionary Computation (CEC)* (2007)
23. A. Soltoggio, B. Jones, Novelty of behaviour as a basis for the neuro-evolution of operant reward learning, in *Conference on genetic and evolutionary computation (GECCO)* (2009)
24. K.O. Stanley, B.D. Bryant, R. Miikkulainen, Evolving adaptive neural networks with and without adaptive synapses, in *IEEE Congress on Evolutionary Computation (CEC)* (2003)
25. K.O. Stanley, D. D'Ambrosio, J. Gauci, A hypercube-based indirect encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
26. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
27. R.S. Sutton, A.G. Barto, *Reinforcement learning: An introduction* (The MIT press, 1998)
28. P. Tonelli, J.-B. Mouret, On the relationships between synaptic plasticity and generative systems, in *Conference on genetic and evolutionary computation (GECCO)* (2011)
29. P. Tonelli, J.-B. Mouret, On the relationships between generative encodings, regularity, and learning abilities when evolving plastic artificial neural networks. PLoS One. **8**(11), e79138 (2013)
30. J. Urzelai, D. Floreano, Evolution of adaptive synapses: robots with fast adaptive behavior in new environments. Evol. Comput. **9**(4), 495–524 (2001)