Chapter 23
On Reservoir Computing: From Mathematical Foundations to Unconventional Applications

Zoran Konkoli

Abstract In a typical unconventional computation setup the goal is to exploit a given dynamical system, which cannot be easily adjusted or programmed, for information processing applications. While one has some intuition of how to use the system, it is often the case that it is not entirely clear how to achieve this in practice. Reservoir computing represents a set of approaches that could be useful in such situations. As a paradigm, reservoir computing harbours enormous technological potential which can be naturally released in the context of unconventional computation. In this chapter several key concepts of reservoir computing are reviewed, re-interpreted, and synthesized to aid in realizing the unconventional computation agenda, and to illustrate what unconventional computation might be. Some philosophical approaches are discussed too, e.g. the strongly related implementation problem. The focus is on understanding reservoir computing in the classical setup, where a single fixed dynamical system is used: To this end, mathematical foundations of reservoir computing are presented, in particular the Stone-Weierstrass approximation theorem, with a mixture of rigor, and intuitive explanations. To make the synthesis it was crucial to thoroughly analyze both the differences and similarities between Liquid State Machines and Echo State Networks, and find a common context insensitive base. The result of the synthesis is the suggested Reservoir Machine model. The model could be used to analyze how to build unconventional information processing devices and to understand their computing capacity.

23.1 Introduction

In a typical information processing setup a *reservoir computing device* implements a mapping $\mathcal{F}$ from the space of time series input data Ω to the space of time series output data Ω',

Z. Konkoli (✉)
Department of Microtechnology and Nanoscience - MC2,
Chalmers University of Technology, SE-412 Gothenburg, Sweden
e-mail: zorank@chalmers.se

© Springer International Publishing Switzerland 2017

A. Adamatzky (ed.), *Advances in Unconventional Computing*,
Emergence, Complexity and Computation 22,
DOI 10.1007/978-3-319-33924-5_23

$$\mathcal{F} : \Omega \rightarrow \Omega' \tag{23.1}$$

The computation is performed in the on-line mode: At each time instance the previously "seen" time series is "inspected" by the device and an output is assigned. In signal engineering such an object is referred to as a *filter*. As the time instance moves, an input data series is converted into the output data series.

The term "reservoir computing" describes a related set of ideas for performing computation with non-linear dynamical systems as filters. Surprisingly, despite the fact that the idea originates from a solid mathematical background, the theory of reservoir computing is still phenomenological in many ways. There are essentially two perspectives that emphasize the use of either a class of systems or a single system. These perspectives are strongly related, and in the literature are often treated as one concept. This can cause a great deal of confusion since implicit context dependent assumptions are frequently made. For example, when envisioning applications, one often assumes the one-system perspective and, yet, when discussing the expressive power of the same system the other perspective is assumed (without rigorous justification). This is one of the reasons why the theory of reservoir computing is still phenomenological.

In the following, the distinction between the two perspectives will be kept explicit. The one-system setup will be referred as the classical setup of reservoir computing, since this perspective is often (implicitly) assumed in the literature when applications are discussed:

> A reservoir computer consists of a dynamical system, a reservoir, and an interface that extracts the information stored in the internal states of the system, a readout layer. If the configuration space of the system is complex enough, various inputs to the system can drive the system to different regions of the configuration space, which represents computation. The result of the computation is extracted by the readout layer.

As a paradigm, reservoir computing addresses the natural question whether and under which conditions an arbitrary system can be used for advanced information processing applications. The outcome depends on which choices are available, e.g. whether it is possible to choose among many systems or, if only one system is available and how adjustable it actually is. The key foundation of the reservoir computing field is an insight that this indeed can be done, regardless of the perspective assumed. If the following conditions are met the reservoir computer could be used to compute in principle anything[1]:

[1] While such machines are expected to be able to implement a broad class of information processing tasks, they are not Turing universal. There is no obvious way of establishing that such machines have infinite expressive power, at least not in the strict sense of the word. For example, one would have to provide a construct to realize a universal Turing machine using reservoir computing. The abstract concepts such as "the tape" or "the reading head" are not easily realized in this context.

The properties that guarantee that the system can be used for reservoir computing are usually referred to as (i) *the separability*, (ii) *the echo state*, and (iii) *the fading memory* property. Such system are referred to as *good reservoirs*. Further, the readout layer should possess the (iv) *the approximation* property.

The goal will be to re-interpret the above requirements by adopting a practical point of view assuming that the aim is to actually construct a reservoir device from a given dynamical system. *This classical scenario is ubiquitous in unconventional computation*.

The Promise of Reservoir Computing

As a paradigm, reservoir computing might have an enormous technological potential. Property (iv), the approximation property, might be hard to realize for readout layer implementations that are not done in-silico. However, properties (i-iii) appear generic. Many systems might exhibit such behaviors.[2] Should this be really true, the idea of reservoir computing could have enormous practical implications for a range of information processing technologies.

It is possible that there are many systems that can be used for advanced information processing in the reservoir computing context, but are simply overlooked since they have never been studied that way.

However, it is not yet clear whether the field of reservoir computing can live up to its technological promise. There are two pertaining issues. First, to verify rigorously whether the system of interest (a reservoir) has properties (i–iii) is a highly non-trivial task. This is a serious obstacle towards understanding which systems might be used for reservoir computing. Second, the theory of reservoir computing has not been developed enough.

The requirements (i–iii) cannot be expressed in clear engineering terms since they are a direct translation of the related mathematical formulations which are simply impractical for engineering applications. This is another reason why the theory of reservoir computing appears phenomenological. It is necessary to understand and re-interpret the existing reservoir computing mathematical background to bridge towards the engineering side.

A brute-force strategy for finding suitable reservoir systems is to simply start checking for every conceivable system whether the properties exist. Eventually, by

[2]For example, systems with the dynamics that is (a) chaotic-like or input sensitive (to ensure the separation property), or (b) with some sort of friction or energy dissipation (to ensure the echo state or equivalently the fading memory property).

studying many such systems one should be able to generalize and gain an understanding of which systems might have such properties. The main problem with the strategy is that it is in general very hard to judge whether an arbitrary system has indeed the required properties. There are several reasons for that. From the strict mathematical point it is possible to rigorously define when the system possess the separation property. However, despite the presence of the mathematical rigor, the validity check is not straight forward since there is simply no universal procedure for performing the check in practice. Further, when engineering reservoir computing systems, one can ask: how much should the system separate inputs? Is there a tolerance range (a resolution) which is acceptable? In general, the issues related to accuracy, tolerance to damage or noise, have not been extensively addressed in the reservoir computing literature.

The goal of this chapter is to provide the necessary overview of key mathematical concepts, and take them as a starting point to develop a suitable theory and the related strategies for building reservoir computing systems, which could provide some generic guidance for the related engineering efforts. Several practical principles will be discussed throughout the text. The text is organized as follows.

- A brief history of reservoir computing is presented in Sect. 23.2, together with some key mathematical concepts. The section contains a discussion about the two most common up to date implementations of the idea: Liquid State Machines and Echo State Networks.
- Section 23.3 contains the definition of the Reservoir Machine concept. The definition is a mathematical formalisation of what reservoir computing in the classical setup is. The related mathematical concepts serve as the foundation for the analysis in the sections that follow.
- The technological promise of reservoir computing is discussed in two subsequent Sects. 23.4 and 23.5, each inspired by two different perspectives:
  - Section 23.4 contains a discussion on the relationship between reservoir computing and philosophy of computation, and in particular the implementation problem.
  - Section 23.5 analyzes the mathematical foundation of reservoir computing, the Stone-Weierstrass approximation theorem. This section contains a formal mathematical background that is necessary for understanding reservoir computing on one hand, and understanding how to build reservoir computers on the other. The section also contains some examples of how the theorem can be used.
- Section 23.6 contains a discussion of how the technological potential of reservoir computing could be realized in practice. It contains a set of practical guidelines that, if adhered to, could enable us to build powerful reservoir computers.
- The concluding Sect. 23.7 contains a brief discussion of several key theoretical concepts that one should learn to command (understanding, implementation, and exploitation). If we are able to do that, the "prophecy" of reservoir computing might be fulfilled.

23.2 A Brief History of Reservoir Computing

Reservoir computing was independently suggested in [1, 2] and [3–5]: These studies introduced the concepts of Liquid State Machines (LSMs) and Echo State Networks (ESNs) respectively. Both focused on explaining some specific features of neural network dynamics. The LSM and ESN concepts were further developed and exploited thereafter, e.g. as discussed in the reviews [6, 7].

After being initially suggested, the concept has been exploited in a range of applications. The web-site [8] dedicated to reservoir computing contains an excessive list of references. Since this book chapter is not a review paper the reader is directed to these sources for additional information. To illustrate the flexibility of the concept, a few more recent examples that exploit different reservoirs can be mentioned: reservoirs made of memristor networks have been considered in for pattern recognition [9, 10] or harmonics generation [11], photonic systems as reservoirs have been discussed in [12–14], etc. For a more detailed discussion on various reservoirs that have been considered please see [15] and references therein. The field seems to be exploding and reviewing all possibilities that have been investigated in the literature is out of the scope of this book chapter.

Surprisingly, since the original publications, three are very few studies that address the fundamental (conceptual) side of the problem. For example, one might wonder, what are the limits of reservoir computing? or what is the computing capacity of such devices? and ultimately, how to increase it? A few examples of such studies can be found in [12, 16–20] but there seems to be a gap in the literature, as this fundamental side of the problem is not attracting extensive attention. In particular, there seem to be a lack of interest in the mathematical foundations of reservoir computing, which is strange given that these play a prominent role in formulating the concept. One of the goals of this book chapter is to remedy this situation by re-visiting the mathematical foundations of reservoir computing.

For historical reasons, in the literature, the reservoir computing concept is assumed to be synonymous with Liquid State Machines and Echo State Networks. Despite being related, these two models emphasize slightly different perspectives. The LSM model is formulated for a class of systems, while the ESN model emphasizes the classical (one system) perspective. Both approaches assume a strict separation between the dynamical system (the reservoir) and the readout layer (the interface), where the reservoir carries the full burden of computation.

Liquid State Machine(s)

Liquid State Machine is a model of computation with, in principle, the expressive power of the Turing machine. The expressive power follows directly from the use of the Stone-Weierstrass approximation theorem and the assumption that there is whole class of devices (machines) to choose from that implement the assumptions of the theorem. The Stone-Weierstrass approximation theorem strongly emphasizes the concept of the algebra of functions. In some sense, the LSM model is the most direct implementation of the Stone-Weierstrass approximation theorem in the context of

time series information processing. This is exactly the reason why one assumes the existence of a class of machines (the base filters) that realize an algebra of continuous filters.

Echo State Network(s)

The echo state network approach strongly adopts the classical perspective of reservoir computing with a focus on a chosen dynamical system. The concept was suggested to be able to explain the observation that random networks can be made to perform computation by using a somewhat "lighter" training procedure. This insight is a result of numerous numerical investigations of recurrent neural networks. The background to the echo state network idea is the realization that it is not necessary to train the full network, but only a smaller part consisting of the outer (interface) layer of neurons. To train the interface part it is sufficient to use very simple methods, e.g. the linear regression or similar.

Are These Two Approaches Similar?

In the literature no formal distinction is made between the two approaches, which might seem rather confusing: Strictly speaking the two perspectives are not identical. However, there is a deep connection between the two perspectives due to the fact that a single complex dynamical system can harbor many smaller subsystems. If these small subsystems are coupled, their dynamics will be also complex. In this way a single complex system can implicitly represent a class of smaller "mini/micro-reservoirs". A typical example is an artificial neural network with many neurons or groups of neurons.

23.3 One-System Reservoir Computing: Reservoir Machine

The classical reservoir computing setup, where the goal is exploit a *single* dynamical system for computation, can be mathematically formalized as follows. The dynamical system is normally referred to as a reservoir $\mathcal{R}$. The reservoir needs to be equipped with a readout layer ψ, which is used to probe the states of the reservoir. The reservoir and the readout layer define a reservoir device.

To be able to describe information processing features in exact mathematical terms the phrase *reservoir (computing) machine* will be used to indicate that a reservoir has been assigned a readout layer (possibly optimized for a specific information processing task). Mathematically, a reservoir machine $\mathcal{M}$ is an ordered pair that consists of the reservoir and a readout layer

$$\mathcal{M} = (\mathcal{R}, \psi) \tag{23.2}$$

In this construction the only variable (adjustable) part of the machine is the readout layer ψ. From the engineering perspective a key challenge is to actually construct a suitable readout layer ψ and train (adjust) it. If the readout layer is simple in some sense, it should be easily trainable.[3]

23.3.1 Reservoir $\mathcal{R}$

A reservoir $\mathcal{R}$ is a dynamical system that evolves in time and can respond to external inputs. The input is assumed to exert a direct influence on the internal (microscopic, mesoscopic, or macroscopic) degrees of freedom of the reservoir. For theoretical modeling, the time variable t can be both discrete, $t \in \mathbb{Z}$, or continuous, $t \in \mathbb{R}$, depending on which type of dynamics seems more appropriate.

The input consists of time data series. Formally, a data series u is a mapping

$$u : \mathbb{R} \to \mathbb{U} \text{ or } u : \mathbb{Z} \to \mathbb{U}$$

that assigns to each time instance t a value taken from a set $\mathbb{U}$. Typically, the set $\mathbb{U}$ is taken to be a bounded subset of $\mathbb{R}^n$, e.g. a point in the set is given by $u \equiv (u_1, u_2, \ldots, u_n)$ where each $u_i \in [u_{\min}, u_{\max}]$ is taken from an interval on the real line; $u_{\min}, u_{\max} \in \mathbb{R}$. The time variable can be used to "index" all values in the series by evaluating $u(t)$. Further, let

$$x \in \mathbb{S}$$

denote an internal degree of freedom (a state) of the reservoir, where $\mathbb{S}$ denotes the space of all possible states of the system. The variable x could be treated as an observable: It does not necessarily have to represent a microscopic degree of freedom, like a position of a molecule. It could be also a variable like the temperature, or the concentration profile.

The dynamics of the system is governed by an evolution operator that describes how the internal state of the system $x(t)$ changes in time under the influence of the external input $u(t)$. A typical continuous model is stated as

$$\frac{dx(t)}{dt} = H(x(t), u(t)) ; \ t \in \mathbb{R} \tag{23.3}$$

and a typical discrete model is given by

$$x_t = H(x_{t-1}, u_t) ; \ t \in \mathbb{Z} \tag{23.4}$$

[3]Naturally, this cannot be taken for granted as the computational simplicity does not necessarily imply that the system is easily adjusted in engineering terms.

where $H(x, u)$ denotes a multi-variable function that governs the dynamics of the system. To make the notation simpler the same symbol H has been used in both equations.

Note the difference in notation for the continuous and the discrete time dynamics, e.g. $u(t)$ versus u_t, and $x(t)$ versus x_t. In the following the symbols $q(t)$ with $q \in \{x, u\}$ will be used when discussing both continuous and discrete systems. The symbols q_t with $q \in \{x, u\}$ will be used exclusively for discrete systems.

The machine performs computation by changing its internal state under the influence of the input. The internal state of the system at each time instance depends on the whole history of previous inputs. In signal engineering, such a system is referred to as a *filter*. It maps input time series $u(t)$ into another times series $x(t)$. Thus the reservoir can be seen as an implementation of the filter

$$u \rightarrow x = \mathcal{R}(u) \tag{23.5}$$

where the individual sequence values can be inspected as $x(t) = \mathcal{R}(u)(t)$.[4] The same symbol is used to denote the physical reservoir and the mapping (the filter) it realizes.

23.3.2 Readout-Layer ψ

Every reservoir is expected to be equipped with a readout layer ψ. The readout layer should not process information in any substantial way. It should be only used to assess the information stored in the dynamical system.

What is the complexity of ψ that we can allow and still be able to claim that ψ is not doing any substantial computation? This is of course a complicated mathematical and philosophical problem. Please see [21] for a possible answer to this question. In the reservoir computing context it is naturally resolved by claiming that the computation should be done instantaneously. The readout function should not be a filter in the strict engineering sense. The function per se should not accumulate any memory. It should only "see" history through x.[5] Mathematically, these principles are expressed as follows. The readout layer is used to extract the output of computation $o(t)$, i.e. by "inspecting" the value of $x(t)$ at each time instance t:

$$o(t) = \psi(x(t)) \tag{23.6}$$

The symbol $\psi(x)$ denotes a multi-variable function that provides a mathematical description of the readout layer.

[4] Note that it would be wrong to use $\mathcal{R}(u(t))$.

[5] For complex systems, $x(t)$ is expected to contain a very long list of values and, in principle, a lot of history could be stored in x.

23.3.3 Reservoir Machine as a Model of Computation

The Filter Implemented by the Machine $\mathcal{M}$

Taken together, each machine $\mathcal{M}$ realizes the related function (filter) $\mathcal{F}$ that maps an input sequence u into the output sequence o:

$$u \to o = \mathcal{F}(u) \tag{23.7}$$

The values of these sequences can be inspected at each time instance t as $o(t) = \mathcal{F}(u)(t) \equiv \psi(\mathcal{R}(u)(t))$.[6] In what follows no formal distinction will be made between the machine and the filter it implements, and $\mathcal{M}$ will be used in place of $\mathcal{F}$ except in situations when a confusion might arise.

Configurability of the Readout Layer Generates a Class of Machines

It will be useful to considers a collection of all machines that are obtained from a fixed but otherwise arbitrary reservoir $\mathcal{R}$ and all possible readout layers:

$$\mathfrak{M} = \{\mathcal{M} \equiv (\mathcal{R}, \psi) : \psi \in \Psi\} \tag{23.8}$$

The collection of machines $\mathfrak{M}$ is generated by the reservoir and will be referred to as a *programmable reservoir machine*. Once a particular readout layer has been chosen (engineered) it will be referred to as a *programmed reservoir machine*, or just a *reservoir machine*. Thus Reservoir Machine can be viewed as a model of computation. It is clearly not a universal model of computation. Its computing power comes mostly from the reservoir, which is "frozen": In the LSM construct it is always allowed to choose among different reservoirs. In the ESN model, this is sometimes allowed, and sometimes forbidden. In the Reservoir Machine model it is *always* forbidden.

23.4 The Technological Potential of Reservoir Computing: Lessons from Philosophy of Computation

Hilary Putnam suggested a construction of how to use *any* object to implement *any* finite state automaton [22]. The statement is a paradox: even a rock should be able to compute anything. The recent advances in understanding the relatively novel approach to information processing, the reservoir computing paradigm, seem to indicate that this seemingly absurd idea is not without merits.

[6] Note that here we do not use $\psi(\mathcal{R}(u))(t)$. That would be incorrect, given the assumption that the readout layer ψ should not accumulate any information from the past. The notation $\Psi(x)$ implies that Ψ acts as a filter, which is clearly not the case according to the definition.

23.4.1 Putnam's Construction

Understanding which dynamic properties guarantee information processing ability is a highly complex problem. As an example, to appreciate the difficulties associate with the problem, consider the following example. Can a rock compute? An engineer's answer would very likely be "no", which is also the common intuitive expectation. Interestingly, a philosopher's answer is "yes". This begs a question: Which one is correct? Probably both.

One of the key issues that the philosophy of mind tries to address is whether the human brain implements an automaton. If yes, then different states of mind are just different states of the automaton, e.g. as summarized in [23]. As a response to this thesis Hilary Putnam [22] provided a construction through which it can be shown that even a simple object as a rock can be made to implement any finite state automaton. This rather absurd conclusion was the very reason why Putnam considered the construction. He used it to show that the notion of computation needs to be restrained when discussing philosophy of mind. Clearly, just the ability to compute cannot be used to define what a mind is. In this context, the notion of computation is simply too broad, as any object seems to be able of performing it. Nevertheless, the statement is a paradox that illustrates an important principle, and a way of thinking about computation, and especially unconventional computation.

23.4.2 A Thought Experiment

Putnam's construction and the reservoir computing idea are strongly related. What would happen if Putnam's idea were taken out of context and applied in the context of reservoir computing? Assume that the goal is to use the rock as a reservoir. Note that Putnam's rock is taken grossly out of context, but such a possibility can be envisioned, and servers the purpose of illustrating an important point.

The Rock can Compute but an Auxiliary Interface must be Used

Putnam's construction has been criticized from many angles (cf. [24–26] and references therein). The most common argument against the construction states that to implement a complex automaton on a rock, large auxiliary equipment would be needed. At the end, the actual computing would be done by the auxiliary equipment and not by the rock. This is certainly a valid argument and extremely useful for understanding the reservoir computing idea.

Rock can be used for computation in principle, but this is hard to achieve in practice. This particular line of criticism against Putnam's construction indicates that when discussing computing ability of a dynamical system, it is very important to distinguish between the system per se and the interface that the system might be equipped with to achieve information processing.

The Interface is Equally Important as the Reservoir

This further implies that it is very important to understand how to interface the system properly, and how to mathematically describe the interface and gauge its computing power versus the computing power of the system. This line of reasoning has been explored in [21] which contains an example of how these ideas can be formalized. Interpreted in this vein, the thesis of reservoir computing states implicitly that the interface cannot be too complex. This is exactly the reason why either a simpler linear readout algorithm or an easily trained perceptron networks is used as the interface in the context of reservoir computing. Clearly, by using an auxiliary equipment the rock could be turned into the system that has the desired properties, but this equipment, very likely, will never be classified as a "simple readout layer".

How to Judge Whether a Given System has the Key Properties?

The properties seem so generic that it seems reasonable to assume that many physical systems can perform computation, even the systems which were not specifically designed by humans for that purpose. In that sense the original construction by Putnam has some practical relevance. This connection between Putnam's paradox and the idea of reservoir computing has been already pointed out in [27]. The rock is certainly not one of these systems. From the reservoir computing perspective, a rock cannot compute, as it lacks these key properties.[7]

23.5 The Technological Potential of Reservoir Computing: Lessons from Mathematics

The problem of a suitable approximation is probably as old as mathematics itself. In the particular case that deals with the problem of approximating an arbitrary function with a fixed class of functions, the approximation problem has a solution with a surprisingly elegant formulation. The theorem is one of the foundations of reservoir computing, and the structure of the theorem will be discussed in the following. An attempt will be made to present a deeply mathematical subject by emphasizing intuitive reasoning.

The theorem is normally referred to as the Stone-Weierstrass approximation (SWA) theorem (SWAT). Behind the simple formulation of the theorem hides an enormous application potential. The theorem has been used frequently in both mathematics and engineering to analyse approximation properties (the computing capacity) for plethora of systems. It is somewhat surprising that is has not been used excessively in unconventional computation too.

[7]Note that this statement might be an oversimplification. For example, imagine that there is simple readout layer that can access the internal states of the rock. Then the rock might still have the desired properties.

First, the theorem will be stated in its most general abstract form. The initial abstract formulation is useful when proving the theorem and, most importantly, it is useful for analysing unconventional computation applications, as will be illustrated in the following sections. Perhaps one of the reasons why the theorem has not found its way into unconventional computation is that it is very likely incomprehensible to somebody not versed in advanced analysis. Accordingly, the abstract formulation will be augmented by discussing the key components of the theorem to provide more intuitive formulations and facilitate its use in the unconventional computation context. Second, a version of the theorem will be provided as used by engineers. Third, while being an extremely useful tool, the theorem has its limitations, which will be discussed too.

A note on notation: In this section various metric spaces are discussed. Naturally, the most important "metric" space of interest is the space of input sequences Ω.[8] The elements of this space should be labeled by the symbol u. However, in this section a temporary change of notation is made, to make the connection with the existing mathematical literature more explicit: It is a custom to label elements of a metric space by using the symbol x.[9]

23.5.1 A Rigorous Mathematical Formulation of the Theorem

Theorem 1 (Stone-Weierstrass—the first version) *Let $\mathcal{A}$ be an algebra of continuous functions that map from a compact metric space Ω to the set of real numbers $\mathbb{R}$. Let elements of $\mathcal{A}$ separate points, and let there be a constant function in the algebra. Let $C(\Omega, \mathbb{R})$ denote the set of all continuous functions that map from Ω to $\mathbb{R}$. Then, $\mathcal{A}$ is dense in $C(\Omega, \mathbb{R})$.*

Some of the concepts used in the above formulation of the theorem are explained below. The explanations that follow are not provided solely to educate the reader regarding the theorem per se, but they are essential for understanding the reservoir computing idea, and for practical applications of the theorem. In particular, from the discussions that follow it should be clear which mathematical properties a physical system must possess that would render it useful for computation. Later on, these abstract mathematical properties will be discussed from a practical point of view (e.g. when building such devices). Several examples will be provided that illustrate how these mathematical concepts might manifest themselves in applications.

Definition 1 When does the set of functions form an algebra? Formally, this is stated as follows. Let $\mathcal{A}$ and Ω be as in the theorem above. The set of functions

[8] Here the quotation marks are used since Ω is not a metric space for any norm. To turn it into a metric space a special norm has to be used, as discussed later.

[9] The reader can substitute u in place of x whenever in doubt regarding how mathematical statements discussed in this section relate to the discussion on reservoir computing contained in other sections.

$\mathcal{A} = \{a|a : \Omega \to \mathbb{R}\}$ forms an algebra when for any pair of elements $a, b \in \mathcal{A}$, the combinations $a + b$, ab, and wa with $w \in \mathbb{R}$ are also in $\mathcal{A}$, where $(a + b)(x) \equiv a(x) + b(x)$, $(wa)(x) \equiv wa(x)$, and $(ab)(x) \equiv a(x)b(x)$ for every $x \in \Omega$.

An algebra of functions is a slight extension of the concept of the vector space of functions. In addition to the usual vector operations (the addition and multiplication by a constant), to form an algebra, the set of functions needs to be closed with regard to the pairwise multiplication operation as well.[10] Further, it is somewhat surprising that the presence of a constant function is explicitly required. An example will be provided later on to illustrate why this property is important.

This closure property is an important property of an algebra that makes it useful for analysing expressive power of the algebra (e.g. it approximation power, in the mathematical sense of the word), and for analysing unconventional computing devices (e.g. their computing capacity).

Definition 2 What does it mean for an algebra to separate points? Let $\mathcal{A}$ and Ω be as in the theorem above. Algebra of functions $\mathcal{A}$ separates points if for every two elements $x_1, x_2 \in \Omega$ that are different, $x_1 \neq x_2$, an element in the algebra g can be found such that $g(x_1) \neq g(x_2)$.

This is another important property of the algebra that is necessary for approximation purposes. Note the particular form of the requirement. For example, it is not required that two distinct elements from the algebra are found such that $g_1(x_1) \neq g_2(x_2)$, or such that $g_1(x_1) \neq g_2(x_1)$ and $g_1(x_2) \neq g_2(x_2)$. The condition used in the definition is a rather mild in the sense that it is not too restrictive on the algebra.

The separation property should be checked for a given pair of points, one at a time. Once the pair has been fixed, the user of the theorem needs only to find an element in the algebra that will satisfy the condition. In principle, if different pairs of points require different elements is of no concern. However, there must be enough elements in the algebra: The real difficulty is in checking that this can be done for every pair of points. Later on, it will be shown that this transforms into an important engineering requirement.

Definition 3 What does it mean that $\mathcal{A}$ is dense in $C(\Omega, \mathbb{R})$? By definition, it means that $\bar{\mathcal{A}} = C(\Omega, \mathbb{R})$, where the symbol $\bar{\mathcal{A}}$ denotes the closure of the algebra $\mathcal{A}$. The closure of $\mathcal{A}$ is defined as the smallest closed set that contains $\mathcal{A}$.

For someone without a background in point-set topology this statement might not make much sense. Thus alternative more intuitive definition will be provided.

[10] Note that we do not require that the set is closed with respect to the composition operation $a \circ b$ with $(a \circ b)(x) = a(b(x))$. In here, ab does not refer to $a \circ b$.

Definition 4 Intuitively, one should think of $\bar{\mathcal{A}}$ as the set of all functions that can be approximated by elements in $\mathcal{A}$. Now it becomes clear that the statement $\bar{\mathcal{A}} = C(\Omega, \mathbb{R})$ implies that under the assumed conditions the algebra can approximate any function.

Thus there are two definitions of the closure, as the smallest closed set containing $\mathcal{A}$ and as a set of functions that can be approximated by elements in $\mathcal{A}$. To see that they are equivalent requires a bit of work.

First, it is necessary to assume that $\mathcal{A}$ forms a metric space. For any two elements a and b in $\mathcal{A}$ one must be able to compute the distance between these two elements $\rho(a, b)$. Any algebra of *continuous* functions supports the natural sup norm metric,

$$\rho(a, b) \equiv \sup_{x \in \Omega} |a(x) - b(x)| \tag{23.9}$$

This is a very sensitive measure of similarity, as the difference is checked at every point. With the ability to measure the distance between any two elements one can define what a convergence is by using the standard arguments: A sequence $a_1, a_2, a_3, \ldots, a_n, \cdots$ of elements in $\mathcal{A}$ converges to a_* if for every $\varepsilon > 0$, an index $n(\varepsilon)$ exists such that $n > n(\varepsilon)$ guarantees that $\rho(a_*, a_n) < \varepsilon$.

Second, if $\mathcal{A}$ is a metric space, its closure $\bar{\mathcal{A}}$ can be defined as the set of all possible limits that can be obtained by considering all converging sequences made by using elements from $\mathcal{A}$,

$$\bar{\mathcal{A}} \equiv \{a_* | \lim_{n \to \infty} a_n = a_*, a_n \in \mathcal{A}\} \tag{23.10}$$

This is a rigorous theorem in mathematics. It is not obvious, and it requires some thinking to show that this is true (regardless of the fact that the proof consists of few lines [28]). The statement above implies that if a function f is in $\bar{\mathcal{A}}$, then for every $\varepsilon > 0$ there is an element a in $\mathcal{A}$ such that

$$|a(x) - f(x)| < \varepsilon \tag{23.11}$$

The approximation is uniform, it holds for every $x \in \Omega$. Equation (23.11) follows from the definition of convergence and (23.10). For any $f \in \bar{\mathcal{A}}$ a convergent sequence $\{a_n\}_{n=0,\infty}$ can be constructed. Take the first element in the sequence for which $\rho(a_n, f) < \varepsilon$. This element defines the a in the equation above.

The above formulation of the SWA theorem does not feature the interface (the readout layer). However, the interface that is used to read internal states is an important part of the device. Alternatively, to emphasize the presence of the interface the following formulation of the theorem might be more useful.

Theorem 2 (Stone-Weierstrass—the second version) *Let $\mathcal{B}$ be an algebra of continuous functions that map from a compact metric space Ω to the set of real numbers $\mathbb{R}$. Let elements of $\mathcal{B}$ separate points. Let $\mathcal{P}$ denote the algebra of multivariate polynomials with d variables. Let $C(\Omega, \mathbb{R})$ denote the set of all continuous functions that map from Ω to $\mathbb{R}$. Then, for every accuracy requirement $\varepsilon > 0$, and every continuous*

function $f \in C(\Omega, \mathbb{R})$ *elements* $b_1, b_2, \ldots, b_d \in \mathcal{B}$ *and the polynomial* $p \in \mathcal{P}$ *can be found such that* $|f(x) - p(b_1, b_2, \ldots, b_d)(x)| < \varepsilon$ *for every* $x \in \Omega$.[11]

This formulation is the one that is often used in the context of reservoir computing, since in some situations this formulation is more practical. Note that in the above statement of the theorem there are less restrictions on the algebra since only separation property is required, no constants are needed. The constants are, of course, included in the algebra of polynomial functions.

Further, the two versions of the theorem point to the issue that was discussed previously. How to balance the computing power of the reservoir, versus the computing power of the interface? For example, one might focus on linear interfaces instead, then the question is whether a linear combination of d elements in $\mathcal{A}$ can be found in place of a in Eq. (23.11), e.g. $a(x) = \sum_{i=1}^{d} w_i a_i(x)$. As ε is made smaller the number of elements d that have to be taken in the weighted sum are expected to increase.

In brief, the Stone-Weierstrass approximation theorem states that if an algebra of continuous functions that maps from a compact metric space Ω to real numbers separates points and contains constant element $\mathbf{1}$, then it can approximate any continuous function on Ω uniformly.

Proof The proof of the SWA theorem is a constructive proof, and can be found in many textbooks, e.g. cf. [29]. It usually extends over several pages of text (if all concepts are defined from scratch). The proof will not be presented in here. However, a few interesting steps, from the reservoir computing perspective, will be commented upon. These will be returned to later in the text, but from a more engineering-like perspective.

The proof of the theorem outlines a procedure (an algorithm essentially) of how to approximate an arbitrary continuous function using the elements in the algebra. As one goes through the procedure it is important to check that every step in the algorithm is valid, which is ensured by the assumed conditions in the theorem. More specifically, the recipe works by showing that any $f \in C(\Omega, \mathbb{R})$ can be approximated at two arbitrary points, and exploiting this fact to show than a uniform approximation can be found everywhere. For this procedure to work, the following requirements must be met:

- The two point approximation procedure discussed above: This is the reason why the presence of the constant function in the algebra is important. It is needed to show that any continuous function can be approximated at any two points by the elements from the algebra by exploiting a linear-like interpolation and requiring that $w_1\mathbf{1}(x) + w_2 g(x) = f(x)$ at $x = x_1, x_2$ where $\mathbf{1}$ denotes the constant function.
- All functions of the algebra must map from a *compact domain*. This is used to claim that various parts of the domain can be covered by a finite number of open

[11] Here naturally, the expression $p(b_1, b_2, \ldots, b_d)(x)$ is interpreted as $p(b_1(x), b_2(x), \ldots, b_d(x))$.

sets. This step is necessary to be able to cover all Ω space by patches where the pairwise approximations work well.
- The algebra property, in particular the closure with regard to multiplications, is used to show that $\bar{A}$ forms a lattice (in the mathematical sense of the word), i.e. that min and max operations are possible on functions. These are used to switch to different functions when the patches are changed to stay ε-close to the target function. □

23.5.2 A Few Application Examples of the Theorem

To illustrate the power of the theorem several applications of the first version of the theorem will be illustrated. Each example was chosen to illustrate a particular aspect of the theorem.

The use of the theorem can appear at odds with its formulation, in the sense that it is hard to build an intuition of what is going on behind the scenes when the conditions of the theorem are being checked. The examples below illustrate the fact that the procedure of checking for the separation property lies completely outside of the theorem, and is a challenging problem on its own.

Example: All Polynomials on a Finite Interval $\Omega = [0, 1]$

Consider all continuous functions on a finite interval $\Omega = [0, 1]$, and an algebra of polynomials on that interval. The goal is to see whether it is possible to approximate any continuous function on the interval by using polynomials. It is a well-known result that this is possible (the Weierstrass approximation theorem). It is straight forward to show this using the SWA theorem. The set of all polynomials clearly forms an algebra. The algebra contains a constant element (in fact infinitely many such elements). The algebra also separates points, since any linear polynomial, e.g. $p(x) = x$, works for every pair of points.

Example: The First Ten Legendre Polynomials (Defined on $\Omega = [-1, 1]$

This set does not form an algebra. For example, the product of the tenth polynomial with itself is not in the algebra. This algebra cannot approximate all functions since it is not closed with respect to the multiplication operation.

Example: Polynomials *without* Constant Term on a Finite Interval $\Omega = [0, 1]$

Note that every such polynomial vanishes at $x = 0$. It can be shown that such polynomials can approximate all continuous functions on Ω that vanish at the origin.

However, such polynomials cannot approximate all continuous functions on Ω. This example illustrates why the presence of the constant function **1** in the algebra is important.

Example: An Algebra of Predicate Based Functions on the Interval $\Omega = [0, 1]$

Assume that all functions in the algebra are defined by using logical predicates. Every predicate is a Boolean formula that features x and some constants. For example, a predicate function could be the Boolean expressions $\pi_1(x) = x > 1$, $\pi_2(x) = \sin^2(x) + \cos(x) == 0$, or $\pi_3 = \pi_1(x) \wedge \pi_2(x)$. Let the elements in the algebra a be defined by considering all possible predicate functions. Each predicate function defines the algebra element as $a(x) = 0$ (1) when $\pi(x) = F$ (T) where F and T stand for false and true logical values. Can this algebra be used to approximate all continuous functions on the interval? Unfortunately, the theorem cannot be used since this algebra of functions is not continuous. Clearly, in the present form, the theorem has its limitations.

Example: Polynomials on $\Omega = (-\infty, t]$ with $t \in \mathbb{R}$

In this case the domain is not compact. The theorem cannot be used. It does not matter which algebra of functions is considered. We shall return later on to this example when the fading memory property will be discussed.

23.6 Realizing the Technological Potential: From Mathematical Concepts to Practical (Engineering) Guidelines

How to build reservoir computers with powerful information processing abilities? The elegance of the LSM formalism is an illustration of how the SWA theorem can be used to understand the expressive power of a class of machines. How can one do the same in the classical reservoir computing setup? when only one system can be used to build a reservoir computer.

Motivated by the LSM and ESN setups it is tempting to consider two options: (i) build a class of dynamical systems that can be combined, or (ii) use a sufficiently complex single dynamical system.[12] Both approaches have its merits. The success of the CMOS technology is strong evidence in favor of the first approach. The key technological feature of the CMOS paradigm is that one can exercise full control of the construction process, down to the tiniest component that contributes to the overall information processing ability of the system. However, this control cannot be often exercised in the context of unconventional (natural) computation, where one can only control some selected parts of the system but not all its components.

[12]Used in this way, the LSM and ESN concepts are taken slightly out of context. They were introduced as models of computation, a tools to study specific features of neural network dynamics.

For example, in molecular computing it is very challenging to control individual molecules. For unconventional computing applications the second option seems more relevant. Accordingly, the following text emphasizes the classical (one-) reservoir computing paradigm.

This section discusses how to bridge from the abstract mathematical context of the SWA theorem towards a more engineering like setup when the goal is to build an actual device, a reservoir machine. This is done by carefully analyzing how the conditions of the SWA theorem can be met for an arbitrary reservoir machine, and how these conditions can be engineered in practice. This section is not meant to be a historical overview of the reservoir computing method (though a part of the text follows the historical development of the field), but rather aims to provide a synthesis of it from a practical point of view.

> If one were to interpret the Stone-Weierstrass theorem in a broader, more engineering like context, it would appear that the following reasoning and the resulting hypothesis seem feasible: If a physical system can realize an algebra of functions that separate points, then it should be possible to use the system to compute in principle anything. It is possible that both requirements are naturally realized by physical systems at microscopic level, and for complex systems at even higher levels (meso-, macro-scales). The hypothesis is that the technological potential of reservoir computing can be indeed "released": Provided there are readout layers that can resolve such microstates, there are no a priori reasons why powerful reservoir computing devices could not be realized.

The possibility that the hypothesis is actually true is too important to be ignored. It might change the way we think about information processing and have profound impact on information processing engineering. It is important to understand whether this agenda can be realized in practice, and if not, where the limitations are. This section is an interpretation of the SWA theorem in the technological context of reservoir computing in the classical setup. The goal is to provide a set of broad guidelines of how reservoir computers could be engineered and which requirements should be met in order to turn them into powerful information processing devices. Some open problems are pointed out too.

23.6.1 Existence of the Filter

The first and the most important question is whether any dynamical system realizes a filter. This cannot be taken for granted since the system has to be started from an initial state, and deciding whether the initial state matters or not is a highly non-trivial issue. The echo state property and the fading memory property have been suggested specifically to address this issue.

23.6.1.1 The Fading Memory Requirement

To be able to use a system as a filter, in the on-line computation context, the present state of the system should be weakly dependent on distant inputs. Many dynamical systems found in nature often equilibrate, and have the potential to act as filters. But there are also systems that do not equilibrate easily, e.g. chaotic systems. It is important to be able to distinguish these two classes of systems. There are several ways of formalizing mathematically the condition that the dynamics is insensitive to the initial condition. The most common definition is as follows.

Definition 5 (*Fading memory*) For reservoirs with the fading memory property, the dynamics of the reservoir should not be influenced by a too distant past. Two input time series that differ in the distant past should lead to roughly the same output: For every u and $\varepsilon > 0$, there exist a $\delta(u, \varepsilon) > 0$ and an interval $[t_0 - T, t_0]$ such that $|(\mathcal{F}u)(t_0) - (\mathcal{F}v)(t_0)| < \varepsilon$ for every input v that is δ-close to u on that interval; $|u(t) - v(t)| < \delta$ for $t \in [t_0 - T, t_0]$.

Note that this resembles the definition of continuity at a point (not uniform continuity). It was shown that the systems with fading memory have unique steady states (that lock-onto the input, for a proof see section *XIII*, Theorem 6, in [30]).

Fading Memory Leads to a Special form of Continuity

Interestingly, the fading memory property ensures some useful mathematical properties of the filter realized by the system. Since Ω is the space of infinite time series it is not automatically compact. The Arzelà-Ascoli theorem states that to make the space of functions compact, one would have to, at least, limit the time frame by considering only a finite time window $t \in [t_0 - \tau, t_0]$. However, there are two immediate problems, what should one choose for the reference (computation) time t_0 (the reference issue) and the interval length τ (the length issue)?

The compactness problem can be solved as follows. If the distance between two time series is defined by using the weighting functions construct, which tend to favor more recent values in time,

$$\rho^{(t)}(u, v) \equiv \sup_{k \le t} w(t - k)|u(k) - v(k)| \tag{23.12}$$

where $w(k) \to 0$ with $k \to \infty$, then the space Ω with this metric is compact.

Interestingly, once the compactness is in place, the fading memory ensures that the mapping realized by the filter is continuous. Any filter with fading memory is continuous in the metric $\rho^{(t)}$:

$$\rho^{(t)}(u, v) < \delta \Rightarrow |(\mathcal{F}u)(t) - (\mathcal{F}v)(t)| < \varepsilon \tag{23.13}$$

This result was stated as a theorem in [31].[13] This has been also used as an alternative definition of fading memory (e.g., see section *III*, the definition 3.1 in [30]).

23.6.1.2 The Echo State Property

The echo state property is a result of a direct attempt to deal with the filter existence issue. In echo state networks the present state of the network is an "echo" of the input history. The initial state of the system can be forgotten if the system has been exposed to the input for a sufficiently long time. The rest is a mathematical formulation of the idea.

Historically, it has been realized that the echo state property is crucial if the network can be trained by adjusting its output weights only. Quoting from [32]:

> For the supervised learning algorithms which are used with Echo State Networks (Edit: citing the original technical report [4], and a later review [7]) it is crucial that the current network state x_k is uniquely determined by any left-infinite input sequence $u_{-\infty}, \ldots, u_{k-1}, u_k$.

Such behavior guarantees that the on-line computation is possible, i.e. that the dynamics of the system does not depend on the initial state of the device. The original definition of the echo state property is as follows [5].

Definition 6 (*Echo state, discrete dynamics*) Assume a fixed time instance t. Let $q[-\infty : t] \equiv (\ldots, q_{t-2}, q_{t-1}, q_t)$ denote a left infinite sequence obtained by truncating $(\ldots, q_{t-1}, q_t, q_{t+1}, \cdots)$ at t. For any input $u[-\infty : t]$ that has been used to drive the system (for an infinitely distant past until the time t), and for any two trajectories $x[-\infty : t]$ and $x'[-\infty : t]$ that are consistent with the dynamic mapping (23.4),[14] it must be that $x_t = x'_t$.

Figure 23.1 is a graphical illustration of this property. The echo state property is equivalent to stating that there *exists* an input echo function $\mathcal{E}$ such that if the system has been exposed to the infinite input sequence, its current state is given by

$$x_t = \mathcal{E}(\ldots, u_{t-2}, u_{t-1}, u_t) \tag{23.14}$$

This notation is somewhat uncomfortable since it involves a function with an infinite list of arguments. However, the statement implied by Eq. (23.14) is useful from an engineering perspective, this particular definition emphasizes the existence of a filter. The original Definition 6 is more suitable for mathematical analysis, e.g. for

[13] Note that the key property is the fading memory. The particular definition of the metric is "for free" (it can be always made). There is a nice alignment with the assumptions of the SWA theorem: the domain of the mapping implemented by the filter Ω should be compact, and the mapping should be continuous. Then the filter realized by the reservoir maps from a compact metric space and is continuous. These properties are useful for establishing the expressive power of the filter.

[14] The consistency is expressed as requiring that for a given sequence x it is true that $x_k = H(x_{k-1}, u_k)$ for every k. The same must hold for the other sequence, i.e. $x'_k = H(x'_{k-1}, u_k)$ for every k.

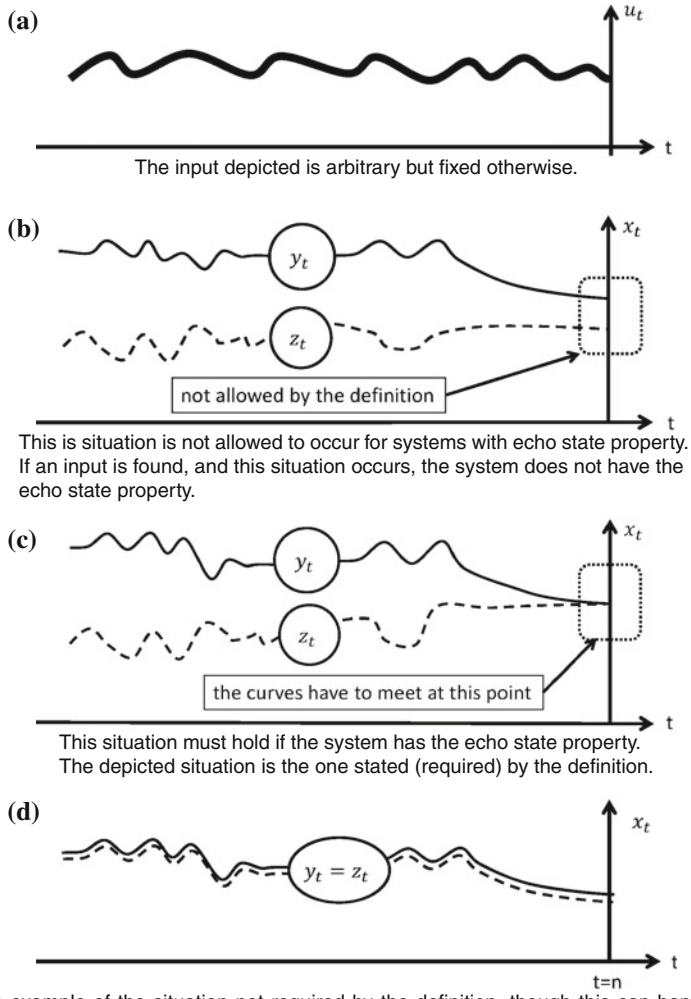

Fig. 23.1 An illustration of the echo state property (backward-oriented perspective). *Panel* **a** Depicts a fixed but otherwise arbitrary input. For a given input, *panels* **b–d** depict various situations. The definition of the echo state specifies exactly which situation is allowed or forbidden. A much more intuitive, the forward-oriented definition, is illustrated in Fig. 23.2

identifying whether the system has such a property. An alternative (and equivalent) definition is listed in the appendix that might be even better suited to that end.

There are systems for which the echo state function $\mathcal{E}$ does not exist, and a few examples will be provided (assuming the discrete dynamics). The examples are chosen to illustrate that instead of (23.14) the following expression should be used

$$x_t = \mathcal{E}(\ldots, u_{t-2}, u_{t-1}, u_t|\mathfrak{x}) \tag{23.15}$$

where $\mathfrak{x}$ denotes the initial state of the system in the infinite past. The initial state can exert an influence on the dynamics for an infinitely long time. Several systems that behave like that are listed below:

Example 1

As the first negative example, consider the system with the discrete dynamics that strongly depends on the initial condition:

$$H_1(x, u) = x \tag{23.16}$$

The system "remembers" the initial condition forever. In fact it is insensitive to the input which is a rather special case. This system does not exhibit the echo state property since it is possible that two separate trajectories exist for a given input u.[15]

Example 2

The following example exhibits a less trivial dynamics, where the system can be influenced by the input:

$$H_2(x, u) = x + \lambda \tanh u \tag{23.17}$$

This mapping resembles a discrete version of the random walk where the spatial increment at each time step depends on the input received by the walker. The input is wrapped by the tanh function just to limit the size of individual steps, which is controlled by λ. Note that for any choice of λ the initial state in the infinite past influences the dynamics.

Example 3

As the last example, consider any system that has at least two quasi[16] stable states with two (or more) basins of attraction. The key feature of the dynamics is that for "weak" inputs the system never crosses the basins of attraction. Transitions from one basin of attraction to the other can only happen under the influence of "strong" inputs. For strong inputs it is not true that (A) for two sequences of states x and x' that are consistent with the dynamics it follows that (B) $x_t = x'_t$. Note that the echo state Definition 6 requires $A \Rightarrow B$. It is possible to find trajectories where A is true but B is false leading to $A \not\Rightarrow B$.[17] Thus the system under consideration does not exhibit the echo state property.

[15]Note that such a pair of trajectories can be found for any input. For example, consider two trajectories that started from x_0 and x'_0 with $x_0 \neq x'_0$ in the infinite past.

[16]Here the term "quasi" indicates that the states are easily "disturbed" by the external input.

[17]It is sufficient to consider two initial conditions that start from different basins of attraction.

23.6.1.3 Summary of the Existence Issue

It is clear that the fading memory and echo state properties are strongly related. It has been proved that if a system has the echo state property then it also has the fading memory property [5]. A version of the (likely[18]) proof that the fading memory implies echo state can be found in [30] (Sect. 8.2, Theorem 6). However, despite being strongly related, both concepts do emphasize slightly different aspects of the problem. For example, the fading memory definition emphasizes the possibility to perform on-line computation, while the echo state property emphasizes the existence of the time invariant filter.

The main problem with both definitions is that it is hard to check whether a given system has either of these. There are some results regarding the existence of the echo state property for a particular class of dynamic functions $H(x, u)$ [3–5]. Clearly, from the above discussion one can see that it is very hard to make generic statements regarding the systems which exhibit these properties. Such analysis has to be done for every system of interest.

23.6.2 The Expressive Power of Reservoir Machine

The notion of the expressive power of the reservoir is important for realizing the advocated technological goals. While it is clear that the echo state mapping exists, provided the specific requirements are met, it is less clear what one can actually do with such devices. If we knew which features of the device influence it expressive power, we would have a theory for building powerful reservoir computers, and we would also understand the limits of the approach.

For discrete reservoirs with the echo state property the input echo function involves an infinite list of arguments. If the system has natural relaxation time (or past forgetting time) τ_* then this limits the number of arguments: Instead of (23.14) the following description of the filter might be closer to the truth,

$$x(t) \approx \mathcal{E}_*(u_{t-\tau_*}, \ldots, u_{t-2}, u_{t-1}, u_t) \tag{23.18}$$

For such systems, how do we realize filters with very short or very long list of arguments? Both situations are problematic. For example, assume that the goal is to realize a filter that takes only the two immediate inputs, $o(t) = (\mathcal{F}u)(t)$, that should be trained to return the sum $(\mathcal{F}u)(t) = u(t-1) + u(t)$. It is not at all clear how to get rid of the dependence on the remaining inputs further away in the past. Filtering longer signals is equally problematic since it is not clear where the distant

[18] "Likely" is emphasized due to the following. First, the proof assumes continuous dynamics. Can it be generalized to discrete dynamics? Second, the proof does not lead straightly to Definition 5, but to a continuous formulation of the echo state that is possibly equivalent to the original echo state definition.

information should be stored. To realize both scenarios would likely require more powerful reservoirs, but there is only one reservoir to choose from.

In the following a brief summary of what is known about the expressive power of liquid state machines and echo state networks is discussed. The expressive power of Reservoir Machine is discussed after that.

23.6.2.1 Lessons from the Past: The Expressive Power of LSM and ESN

A lot of research effort has been spent to understand which dynamic properties of the system can guarantee the echo state property. Much less effort has been spend on understanding how to exploit such a property if it exists. This is still an open problem.

LSM is indeed Turing universal (in the fading memory sense), but only provided that a whole class of machines is considered (the base filters). Not surprisingly, this class should have exactly the same properties as the ones stated in the second version of the SWA theorem. In fact, the LSM model originated from a direct application of the second version of the SWA theorem on filters.

In contrast to LSM, in the literature there seems to be a lack of precision in stating the expressive power of ESN, and possibly some misconceptions. This very likely results from using the work by Maass et al. on LSM in imprecise way without specifying the context properly (e.g. one system, or a class of systems). For example, consider the following quote from [33]:

> Universal computation and approximation properties. ESNs can realize every nonlinear filter with bounded memory arbitrary well. This line of theoretical research has been started and advanced in the file of LSM (Edit: quoting the LSM work by Maass et al. [1] and [34])

In the quote, there are several important assumptions that are implicit. Consider a few examples of what might go wrong if such a claim were made without specifying a proper context.

For example, the first implicit assumption is that one should consider a class of networks, and not a single system (network). The second implicit assumption is that this class should have the properties required by the SWA theorem, be an algebra and separate points (which sould be proven, however). These implicit assumptions are very fragile since their validity is context sensitive. Let us discuss each in turn.

The ESN idea, by construction, emphasizes the use of a fixed network, and in this classical setup the quote is most certainly incorrect (as will be explained later on). But, if the network is complex, then it might implicitly harbor a class of smaller subsystems (see Sect. 3.1 in [5] for an illustrative neural network example). In that sense, the assumption that possibly a large class of echo state mappings is available for a single system is justified. However, while these subsystems might exhibit the echo state property (i.e. realize a set of filters), do these subsystems form an algebra? In fact, they very likely do not form an algebra since the number of elements is finite. But should one worry about it, i.e. is it necessary to require that they do? e.g. as in the

LSM model. Subsystems might exhibit some separation features, but most certainly not the exact separation property.

Another strongly related misconception regards the issue of the expressive power of the classical reservoir computer (since it is often associated with the ESN concept). It is implicitly assumed that RC has an infinite computing power, e.g. as in the following quote from [7]:

> Modeling capacity. RC is computationally universal for continuous-time, continuous-value real-time systems modeled with bounded resources (including time and value resolution (Edit: quoting the work by Maass et al. [35] and [36])

This is simply not true a priori for the same reasons as discussed above. If one-system is used as a reservoir, a great caution should be exercised in making such a claim. The main motivation behind suggesting the Reservoir Machine model was to make these issues explicit so that they can be rigorously addressed. Understanding the expressive power of the model is a highly non-trivial and still an open problem, as discussed below.

The SWA theorem states clearly which properties a collection of filters must possess if used in the information processing context: realize an algebra with constant element that separate points, the algebra must realize mappings from a compact space. In the following, each of the requirements of this generic theorem will be revisited an interpreted in practical (engineering) terms.

23.6.2.2 The Burden of Realizing an Algebra Can Be Taken by the Readout Layer

The key lesson from the SWAT is that machines $\mathcal{M}$ should form an algebra in some sense. Do they, and can we engineer them in such a way in the context of unconventional computation?

At this stage the mathematical concept of an algebra needs to be converted into an engineering one. The requirement that the collection of machines described by Reservoir Machine forms an algebra implies that they can be combined in some way, or adjusted, to obtain new, hopefully more powerful, machines. Any two machines $\mathcal{M}$ and $\mathcal{M}'$ might be combined into a larger machine $\mathcal{M} \odot \mathcal{M}'$ to compute another function (filter). Here the symbol $\odot$ denotes an engineering operation on the machines. The filter being realized this way should be exactly the one that is obtained by applying the algebraic operations on the respective output values $o(t)$ and $o'(t)$, as $o(t)o'(t)$.

The above considerations put some constraints on how the engineering operation should be implemented

$$u(t) \to (\mathcal{M} \odot \mathcal{M}')(u)(t) = \psi(\mathcal{R}(u)(t))\psi'(\mathcal{R}(u)(t)) \tag{23.19}$$

Note that the reservoir is not changing. Only the readout function is allowed to change. *Conceptually, the only freedom we have when designing new machines is to combine their readout layers.* Mathematically, the engineering challenge can be

represented as

$$\mathcal{M}\odot\mathcal{M}' \equiv (\mathcal{R}, \psi)\odot(\mathcal{R}, \psi') = (\mathcal{R}, \psi \odot\psi') \tag{23.20}$$

A direct comparison between (23.19) and (23.20) shows that if we can design a new readout layer ψ'' such that $\psi''(x) = \psi(x)\psi'(x)$ for every $x \in \mathcal{S}$ then it is possible to engineer an algebra of machines. In the similar way it can be shown that engineering the vector space operations, multiplication by scalar $w\odot\mathcal{M}$ or addition $\mathcal{M} + \mathcal{M}'$, can be transferred to the readout layer.

23.6.2.3 Engineering the Readout Layer ψ

The preceding discussion shows that the first technological requirement towards building a reservoir machine is to ensure that the set of readout layers Ψ forms an algebra that can be realized. Of course, for in-silico implementations that is not an issue, but in any other setup it has to be addressed explicitly.

The readout layer, or the interface, is one of the key concepts of the reservoir computing. Intuitively, since the goal is to use the system as is, any auxiliary equipment that has to be added onto the system to turn it into an information processing unit should be as simple as possible. Typically, when performing mathematical analysis the readout function ψ is assumed to be a multivariate polynomial or even a simple linear polynomial. How complex the readout layer should be?

Again, this is an instance where the SWA theorem needs to be reinterpreted in an engineering context. The second version of the SWA theorem indicates that any class of functions which have the universal approximation property will do.[19] However, this might be an unnecessarily strong requirement. In the literature, several implementations of the readout layer have been suggested ranging from a simple linear readout towards a more complicated simple perceptron network. An extensive list of various readout layers can be found in [7] (in Sect. 8).

> The question is why should one engineer more complex readout layers if simpler ones will do? and what is the simplest readout layer that can be used? To answer these questions is an extremely challenging problem that has been grossly overlooked in the literature.

Since the reservoir is frozen, and all the burden of the algebraic properties rests on the readout layer, it is clear that the required complexity of the readout layer is conditioned on the complexity of the dynamics of the reservoir. It is hard to make generic statements without knowing what the reservoir looks like in a bit more detail. Understanding this interplay is still an open problem. There is simply no available theory that could be used to address the issue.

[19] Multivariate polynomials have this universal approximation property. Any function that behaves in the same way can be used instead.

From the practical point of view, it is implicit in the construction that such readout needs to be constructed, and it should be kept be kept as simple as possible. The computing that can be done by the reservoir should not be done by the readout layer. To understand how much of the burden needs to be put on the readout layer one should really start from basics, i.e. the first (abstract) version of the SWA theorem, as discussed below.

23.6.2.4 Separating Points with One Reservoir Is Problematic but Still Possible

The algebra of machines does not automatically satisfy the requirements of SWAT for one obvious reason. There is only one reservoir to choose from. However, it is possible that complex reservoirs can be used to implement such an algebra.

A Problem

Let us see whether the algebra of reservoir machines separates points. For every pair of time series u and v we must find a reservoir machine $\mathcal{M}$ such that $\mathcal{M}(u)(t) \neq \mathcal{M}(v)(t)$ where t stands for the observation (reference) time. This is the place where the construct with a single reservoir breaks: in contrast to the LSM model, there are simply no reservoirs to choose from, only one is available. What choices can one actually make? In principle, there are two. First, while a single system cannot exhibit infinite "separation" power, it might exhibit some if it is complex enough, i.e. if it contains many components that are "addressable". Second, by assumption, there is a class of readout layers to choose from. However, by construction, these cannot contribute to realizing the separation property. Thus, strictly speaking, there is only one choice to be made, one has to "dig" into the system.

A Possibility

If complex enough, the reservoir could be divided into smaller parts: A truly complex reservoir with many components realizes many filters, where the number of filters is equal to the number of microscopic components N:

$$x \equiv (x_1, x_2, \ldots, x_i, \ldots, x_N) \tag{23.21}$$

where each component realizes one filter

$$u \to x_i(u) \tag{23.22}$$

This possibility has been already pointed out in [5] (Sect. 3.1). For example, for a discrete system it implies that input echo functions (filters) exist, such that

$$\begin{aligned} x_t^{(1)} &= \mathcal{E}^{(1)}(\ldots, u_{t-2}, u_{t-1}, u_t) \\ x_t^{(2)} &= \mathcal{E}^{(2)}(\ldots, u_{t-2}, u_{t-1}, u_t) \\ &\ldots \\ x_t^{(i)} &= \mathcal{E}^{(i)}(\ldots, u_{t-2}, u_{t-1}, u_t) \\ &\ldots \\ x_t^{(N)} &= \mathcal{E}^{(N)}(\ldots, u_{t-2}, u_{t-1}, u_t) \end{aligned} \tag{23.23}$$

Thus, possibly, there is a very large class of mappings to choose from.

The filters realized by such sub-systems might or might not form an algebra. However, this feature is *not* an engineering requirement: The readout layer carries the burden of realizing an algebra! What is important is that all these filters separate points: In some sense all these filters should be "different".

The output of the computation is given by

$$o(t) = \psi(x_1(u)(t), x_2(u)(t), \ldots, x_i(u)(t), \ldots, x_N(u)(t)) \tag{23.24}$$

At this stage it is important to formalize what a sub-system might be, i.e. *what the variables x_i actually represent*. In this context, the concept of observable in statistical physics is extremely useful, and in particular the observables that are relevant for information processing as discussed in [21]. Such observables could be referred to as "information processing observables". While formalizing (23.24) in terms of information processing observables would make the discussion more complete it would also make it a bit more technical. We just leave it at claiming, as in the reservoir dynamics Sect. 23.3.1, that x_i does not necessarily describe microscopic degrees of freedom but can also refer to features on larger scales.

In practice, it might be hard to access all subsystems (even if they are not microscopic). From the engineering point of view, a more reasonable assumption is that the readout layer will have only access to a limited number of components. Thus the equation for the output should read instead

$$o(t) = \psi(x_{i_1}(u)(t), x_{i_2}(u)(t), \ldots, x_{i_d}(u)(t)) \tag{23.25}$$

where it has been assumed that the readout layer can only access d components. The multiple $(i_1, i_2, \ldots, i_d)$ denotes a particular choice of component filters being indexed. In practice the number of accessible filters will be such that $d \ll N$.

One Should Resist the Temptation to Divide into too Many Parts

It is tempting to increase N by considering smaller and smaller systems. This would make the collection of the filters generated by the subsystems very large and increase the resolution of the algebra. Note that this is somewhat equivalent to the assumption that x describes microscopic states. It also strongly parallels Putnam's construction (the assumption that every micro-state is accessible). However, there is a fundamental problem with realizing such an agenda. The dynamics becomes noisy.

For observables that refer to very small sub-systems the dynamic laws expressed in (23.3) and (23.4) do not longer apply. For such systems a noise term should be added in the equations of motion. For example, it is a well-known fact that the dynamics of a single molecule in the sea of solvent molecules kept at a finite temperature should be modelled by using a stochastic differential equation. For observables that are microscopic in nature one would have to assume the following dynamical law

$$\frac{dx(t)}{dt} = \tilde{H}(x(t), u(t)) + \eta(t) \tag{23.26}$$

where η is a stochastic variable with a given mean and a finite variance and $\tilde{H}$ indicates that H needs to be modified for the effects of friction. For example, the form of the noise that describes experiments that involve diffusion is $\langle \eta(t) \rangle = 0$ and $\langle \eta(t)\eta(t') \rangle = \gamma \delta(t - t')$ where $\delta(t)$ denotes the Dirac delta function and γ is a temperature dependent parameter.

The Expressive Power of a Reservoir Machine is Limited

Without knowing more details about the component filters $x_i(t); i = 1, 2, \ldots, N$, it is impossible to make further progress. This is exactly the reason why one-reservoir construct is hard to analyze. However, it is clear that if the component filters separate inputs, and if the readout functions can approximate sufficiently well, then it should be possible to tune the reservoir machine towards arbitrary information processing task, but there are clearly limits to what can be computed. The expressive power of a programmable reservoir machine is not infinite.

23.7 Conclusions

Perhaps it is fair to say that reservoir computing is more an insight about computing than an approach to computing. The insight is about the possibility to use an arbitrary dynamical system for computation without elaborate re-configuring (training) procedures. In this chapter the existing problems with realizing this classical setup were discussed, together with what could be done to address these problems, and what could be gained by doing so. The three big questions that were addressed were:

Which features of the construct are hard to engineer? How to engineer such features in principle? What is the expected technological impact of such devices?

In this classical setup, reservoir computing is also a "call" to study dynamical systems, and in particular unconventional computation, in a new way. The Reservoir Machine concept was suggested to make this point of view explicit.

In the literature Echo State Networks and Liquid State Machines are often treated as one concept. Admittedly, they are strongly related but they are not identical. Any interpretation of these concepts is strongly context dependent, as illustrated in the text. This lack of clarity might obscure further progress, and the goal was to, first, point out the key differences between the concepts and then, second, provide a synthesis of the ideas they represent. This was done in the context of the classical setup of reservoir computing. The *Reservoir Machine* concept is the result of the attempted synthesis.

Reservoir Machine has been introduced to emphasize the fact that, ultimately, in the worst case scenario, one is facing the problem of using a fixed dynamical system for computation with a very little freedom of tuning the system. This worst case scenario is ubiquitous in situations when in-silico solutions are not possible, and in particular in the context of unconventional computation.

The explicit formulation of Reservoir Machine established a clear starting point for the analysis of several aspects of the problem: The mathematical foundation of reservoir computing, the SWA theorem, has been re-interpreted to provide strategic guidelines for building powerful unconventional reservoir machines and to aid in understanding their computing power. The most important observations are as follows.

1. There is a strong connection between Putnam's construction and the reservoir computing idea in the classical setup. Putnam's construction addresses nearly the same problem.[20] The notion of the microscopic degree of freedom and the ability to access such states is crucial to establish the connection. This line of thinking puts a clear emphasis on the need to understand how to build good readout layers. This aspect of the problem should be given a serious consideration. While being extremely "passive" in mathematical terms, in the engineering sense the readout layer is as important as the reservoir.
2. It has been shown that, in the Reservoir Machine setup, the readout layer takes the burden of realizing the algebra, while the reservoir takes the burden of separating

[20] It is tempting to argue that the separation property being emphasized in reservoir computing is, in some sense, related to Putnam's requirement of non-periodic behavior and the idea of the internal dial [24].

points. This is in contrast with Liquid State Machines where base filters realize an algebra and the readout layer plays a somewhat passive role. The interplay between the formal requirement to realize an algebra and the requirement that the algebra separates points should be understood better.[21]

3. Given that one can build powerful readout layers, it is important to be aware of the fact that making the sub-systems too small results in noisy dynamics. It is possible that for some applications the presence of noise is not an issue, and might even be desirable, but in majority of cases noise is probably a nuisance. In general, the effects of noise in the reservoir computing setup still need to be understood.
4. There seems to be no theory of reservoir computing that might be directly relevant for engineering applications. The Reservoir Machine construct provides an illustration of how to approach the problem of constructing such a theory.[22]

Perhaps the suggested reservoir machine model could be taken as a starting point for constructing a theory that is relevant from a practical point of view, as it clearly points out the relevant set of issues. There are several options for constructing a better theory of reservoir computing.

1. The SWA theorem might not be particularly useful in the engineering context, after all. The theorem is simply an existence statement. It states that an approximation can be found under given conditions. It does not address the accuracy or tolerance issues, or gives any bounds. One might try to constructs and prove a completely different version of the SWA theorem, in a form that is more useful for engineers.
2. The second option is to re-work some ingredients of the existing theorem towards a more engineering like setup. For example, the separation property seems to be crucial. The separation property is a mathematical formulation of an intuitive understanding that the system must have some "resolution power" (of inputs). It could be useful to re-phrase the separation criterion in less absolute terms by, e.g., requiring not strict separation of inputs, but separation up to a certain accuracy (resolution).[23]

---

[21] Intuitively, one expects that the expressive power of the algebra is strongly related to the "richness" of the algebra, provided such a concept can be defined, e.g. by using the number of sub-filters as a measure. The readout layer is not the source of that richness and, yet, takes the burden of realizing the algebra. This is a result of the systematic analysis that has been undertaken, and should be taken as such.

[22] A lot has been achieved since the reservoir computing concept has been originally suggested. However, there are many issues that are still open, as pointed out throughout the text. For example, in the classical setup, reservoir computing does not have the universal computing power, not even in the fading memory (on-line computation) sense. There is still no sufficient understanding of the expressive power of reservoir computing in the classical (reservoir machine) setup. The understanding provided by the use of SWA theorem is an important first step (e.g. LSM and ESN studies) but to understand the computing capacity of a reservoir machine we clearly need a much better theory of reservoir computing.

[23] To a mathematician, an interesting question, perhaps, might be: given that a class of functions forms an algebra and separates inputs up to a certain accuracy, is there any way to characterize a class of functions that can be represented that way? Which types of theorems could be proven?

3. The third option is to further explore and refine the fading memory and the echo state concepts, despite the fact that, perhaps, they are the part of the reservoir computing theory that has been mostly developed and investigated. For one thing, the equivalence between the two concepts has been proven, but under very specific conditions. It should be clear from the presented discussion that these concepts are strongly related. Accordingly, it is surprising that the equivalence has not been proven under more generic conditions.

To conclude, as a technological platform, the single reservoir perspective featured in Reservoir Machine should be less restrictive when it comes to practical implementations. Admittedly, it is also less expressive, but might have a larger technological impact. It seems that the envisioned reservoir machine technology might solve complicated unconventional information processing problems. As an illustration of how the Reservoir Machine concept can be applied in the context of material computation (e.g. for building material machines) see [15]. Reservoir Machine might be suitable as a platform for improving the existing, and realizing new unconventional computing scenarios. It is perfectly possible that important information processing applications can be realized by using relatively simple reservoirs, since not all relevant information processing tasks are complicated.[24] For example, one can envision plethora of applications of reservoir computing in situations where *in-silico* realization is not feasible, e.g. in medical sensing applications when bio-compatibility is an issue rather than the computing capacity. This suggests that while reservoir computing machines might be used for high-performance computing, their natural zone of application is very likely elsewhere. Such machines could be used for the information processing tasks that require deep integration of the information processing equipment and biological systems.

Acknowledgments This work was supported by Chalmers University of Technology and by the European Commission under the contracts FP7-FET-318597 SYMONE and HORIZON-2020-FET-664786 RECORD-IT.

Appendix

An Alternative Echo State Property Definition

Intuitively, the following definition of the echo state property might be easier to understand. [5] The following notation is useful to rephrase the original definition in the ambience of this chapter. Let $x_t = \mathcal{R}(u|x_n)(t)$ denote the configuration of

[24] For example, consider the problem of on-line time series data analysis and pattern recognition. This is typical example where the separation property is not a strong requirement. What is needed is that only particular input patterns drive the system to a different regions of the configuration space when compared to the background input. It is possible that relatively simple reservoir machines could handle such tasks.

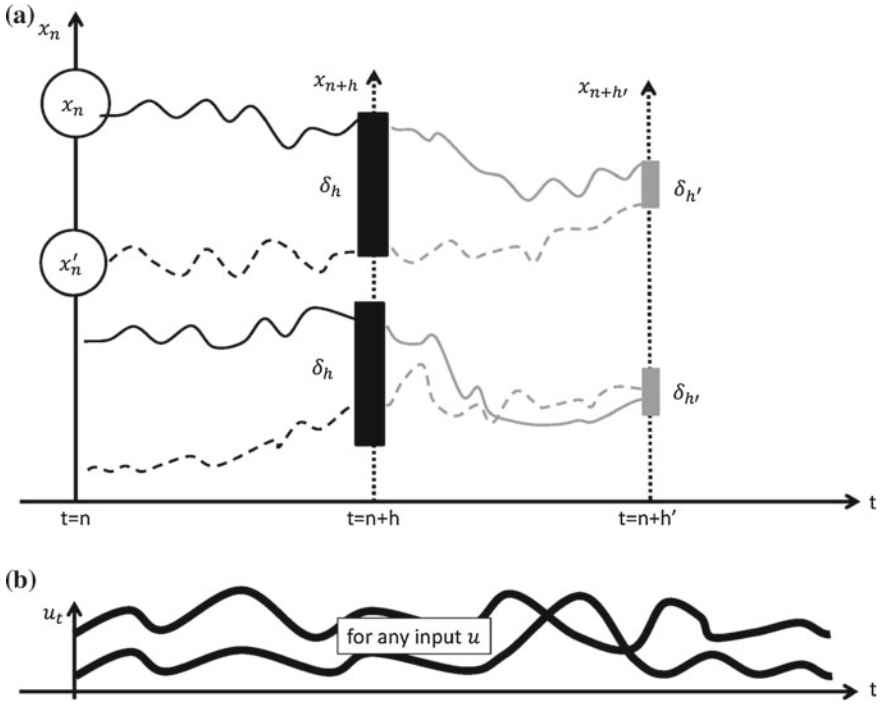

Fig. 23.2 An illustration of the state contracting property. Any pair of trajectories start aligning if one waits long enough. For example, for a fixed pair of initial conditions the trajectories after time h can be both "covered" with the *black box* by moving it vertically. Note that the height of the box δ_h is fixed beforehand and depends only on h. Likewise, if one waits a bit longer, until $t = n + h'$ with $h' > h$ then the size of the box becomes smaller. Note that the *gray box* is smaller than the *black box*. There is always a tendency for each pair of curves to start aligning. The more one waits the more aligned they become. **a** The desired behaviour for any pair of trajectories. **b** The behavior must hold for any input

the system that was at time n in state x_n and that has been exposed to the input u after that, during the time interval $[n : t] \equiv (n, n+1, n+2, \ldots, t-2, t-1, t)$. To emphasize that the system has been exposed to the input during h time steps we write $\mathcal{R}(u|x_n)(n+h)$. Further, let $\|x_t - x'_t\|_{\mathbb{S}}$ denote the distance between any two elements x_t and x'_t in $\mathbb{S}$ for arbitrary t.[25]

Definition 7 (*state contracting*) The network is uniformly state contracting iff there exists a null sequence[26] of bounds δ_h with $h = 0, 1, 2, \ldots, \infty$ such that for every input sequence u and every pair of initial conditions $x_n, x'_n \in \mathbb{S}$ the chosen pair of trajectories approach each other, i.e. $\|\mathcal{R}(u|x_n)(n+h) - \mathcal{R}(u|x'_n)(n+h)\|_{\mathbb{S}} < \delta_h$.

[25] The notation is implicitly suggesting that $\mathbb{S}$ is a vector space but this need not be the case. This form is used to make such expressions more readable.

[26] A null sequence is a sequence of positive numbers that converges to zero.

Note that, in contrast to Definition 6, this definition does not feature any knowledge of an infinite past. For example, there is no need to know what the input looked like in the interval $(-\infty : n) \equiv (\ldots, n-3, n-2, n-1)$. Accordingly, the above definition is easier to understand since it is "forward oriented" and more aligned with the human intuition of how dynamic systems behave. Figure 23.2 is a graphical representation of this property. It describes a system that equilibrates in some sense, i.e. by "locking" onto the input.

The fact that the definition above is equivalent to Definition 6 was proven rigorously in the original publication by Jaeger from 2001 [4]. Note that Figs. 23.1 and 23.2 are different. They depict a priori genuinely different behaviors. Thus a mathematical proof that these two behaviors are equivalent was indeed necessary. The above definition might be more useful if one wants to check whether a given physical system has the echo state property.

References

1. Maass, Wolfgang, Natschläger, Thomas, Markram, Henry: Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput. **14**(11), 2531–2560 (2002)
2. Markram, H., Natschlger, T., Maass, W.: The "liquid computer": A novel strategy for real-time computing on time series (special issue on foundations of information processing). TELEMATIK, **8**, 39–43 (2002)
3. Jaeger, Herbert, Haas, Harald: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
4. Jaeger., H.: The "echo state" approach to analysing and training recurrent neural networks. Technical Report GDM Report 148 (contains errors), German national research center for information technology (2001)
5. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks - with an erratum note. Technical Report erratum to GDM Report 148, German national research center for information technology (2010)
6. Jaeger, H., Lukoöevicius, M., Schrauwen, B.: Reservoir computing trends. KI - Konstliche Intelligenz, **26**, 365–371 (2012)
7. Lukoöevicius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**, 127–149 (2009)
8. ORGANIC-EU-FP7. Reservoir Computing: Shaping Dynamics into Information (2009)
9. Kulkarni, M.S., Teuscher, C.: Memristor-based reservoir computing. In: 2012 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), pp. 226–232 (2012)
10. Carbajal, J.P., Dambre, J., Hermans, M., Schrauwen, B.: Memristor models for machine learning. Neural Comput. **27**, 725–747 (2015)
11. Zoran, Konkoli, Goran, Wendin: On information processing with networks of nano-scale switching elements. Int. J. Unconv. Comput. **10**(5–6), 405–428 (2014)
12. Appeltant, L., Soriano, M.C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C.R., Fischer, I.: Information processing using a single dynamical node as complex system. Nat. Commun. **2**, 468 (2011)
13. Larger, L., Soriano, M.C., Brunner, D., Appeltant, L., Gutierrez, J.M., Pesquera, L., Mirasso, C.R., Fischer, I.: Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. Opt. Express **20**(3), 3241–3249 (2012)

14. Mesaritakis, C., Bogris, A., Kapsalis, A., Syvridis, D.: High-speed all-optical pattern recognition of dispersive fourier images through a photonic reservoir computing subsystem. Opt. Lett. **40**, 3416–3419 (2015)
15. Konkoli, Z., Stepney, S., Dale, M., Nichele, S.: Reservoir computing with computational matter. In: Amos, M., Rasmussen, S., Stepney, S. (eds.) Computational Matter. Springer, Heidelberg (2016)
16. Dambre, J., Verstraeten, D., Schrauwen, B., Massar, S.: Information processing capacity of dynamical systems. Sci. Rep. **2**, 514 (2012)
17. Massar, M., Massar, S.: Mean-field theory of echo state networks. Phys. Rev. E **87** (2013)
18. Goudarzi, A., Stefanovic, D.: Towards a calculus of echo state networks. Procedia Comput. Sci. **41**, 176–181 (2014)
19. Soriano, M.C., Brunner, D., Escalona-Moran, M., Mirasso, C.R., Fischer, I.: Minimal approach to neuro-inspired information processing. Front. Comput. Neurosci. **9**, 68 (2015)
20. Bennett, C., Jesorka, A., Wendin, G., Konkoli, Z.: On the inverse pattern recognition problem in the context of the time-series data processing with memristor networks. In: Adamatzky, A. (ed.) Advances in Unconventional Computation. Springer, Heidelberg (2016)
21. Zoran, K.: A perspective on Putnam's realizability theorem in the context of unconventional computation. Int. J. Unconv. Comput. **11**, 83–102 (2015)
22. Putnam, H.: Representation and Reality. MIT Press, Cambridge (1988)
23. Chalmers, D.J.: A computational foundation for the study of cognition. J. Cogn. Sci. **12**, 325–359 (2011)
24. Chalmers, D.J.: Does a rock implement every finite-state automaton? Synthese **108**, 309–333 (1996)
25. Scheutz, M.: When physical systems realize functions. Minds Mach. **9**, 161–196 (1999)
26. Joslin, D.: Real realization: Dennett's real patterns versus Putnam's ubiquitous automata. Minds Mach. **16**, 29–41 (2006)
27. Kirby, K.: Nacap 2009 Extended Abstract: Putnamizing the Liquid State (2009)
28. Rudin, W.: Principles of Mathematical Analysis. McGraw-Hill (1976)
29. Dieudonne, J.: Foundations of Modern Analysis. Read Books (2008)
30. Boyd, S., Chua, L.O.: Fading memory and the problem of approximating nonlinear operators with Volterra series. IEEE Trans. Circuits Syst. **32**, 1150–1161 (1985)
31. Maass, W., Markram, H.: On the computational power of circuits of spiking neurons. J. Comput. Syst. Sci. **69**, 593–616 (2004)
32. Yildiz, I.B., Jaeger, H., Kiebel, S.J.: Re-visiting the echo state property. Neural Netw. **35**, 1–9 (2012)
33. Jaeger, H.: Echo state network. Scholarpedia **2**, 2330 (2007)
34. Maass, W., Joshi, P., Sontag, E.D.: Computational aspects of feedback in neural circuits. Plos Comput. Biol. **3**, 15–34 (2007)
35. Maass, W., Natschlger, T., Markram, H.: A model for real-time computation in generic neural microcircuits. In: Becker, S., Thrun, S.., Obermayer, K. (eds.) NIPS (Advances in Neural Information Processing Systems 15), pp. 229–236. MIT Press, Cambridge (2003)
36. Maass, W., Joshi, P., Sontag, E.D.: Principles of real-time computing with feedback applied to cortical microcircuit models. In: Weiss, Y., Schölkopf, B., Platt, J.C. (eds.) NIPS (Advances in Neural Information Processing Systems 18), pp. 835–842. MIT Press, Cambridge (2006)