Hands-on reservoir computing: a tutorial for practical implementation

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
31 March 2022

REVISED
27 June 2022

ACCEPTED FOR PUBLICATION
1 July 2022

PUBLISHED
5 August 2022

TOPICAL REVIEW

Hands-on reservoir computing: a tutorial for practical implementation

Matteo Cucchi[1,2,6], Steven Abreu[3,4,6], Giuseppe Ciccone[1], Daniel Brunner[5] and Hans Kleemann[1,*]

[1] Dresden Integrated Center for Applied Physics and Photonic Materials (IAPP), Technische Universität Dresden, Germany
[2] Laboratory for Soft Bioelectronic Interfaces, Ecole Polytechnique Fédérale de Lausanne (EPFL), Geneva, Switzerland
[3] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence & Cognitive Systems and Materials Center (CogniGron), University of Groningen, The Netherlands
[4] Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland
[5] FEMTO-ST/Optics Dept., UMR CNRS 6174, Univ. Bourgogne Franche-Comté, 15B avenue des Montboucons, 25030 Besançon Cedex, France
[*] Author to whom any correspondence should be addressed.
[6] These authors contributed equally to this work.

E-mail: hans.kleemann1@tu-dresden.de

Keywords: physical reservoir computing, hardware-based random neural networks, tutorial review

Abstract

This manuscript serves a specific purpose: to give readers from fields such as material science, chemistry, or electronics an overview of implementing a reservoir computing (RC) experiment with her/his material system. Introductory literature on the topic is rare and the vast majority of reviews puts forth the basics of RC taking for granted concepts that may be nontrivial to someone unfamiliar with the machine learning field (see for example reference Lukoševičius (2012 *Neural Networks: Tricks of the Trade* (Berlin: Springer) pp 659–686). This is unfortunate considering the large pool of material systems that show nonlinear behavior and short-term memory that may be harnessed to design novel computational paradigms. RC offers a framework for computing with material systems that circumvents typical problems that arise when implementing traditional, fully fledged feedforward neural networks on hardware, such as minimal device-to-device variability and control over each unit/neuron and connection. Instead, one can use a random, untrained reservoir where only the output layer is optimized, for example, with linear regression. In the following, we will highlight the potential of RC for hardware-based neural networks, the advantages over more traditional approaches, and the obstacles to overcome for their implementation. Preparing a high-dimensional nonlinear system as a well-performing reservoir for a specific task is not as easy as it seems at first sight. We hope this tutorial will lower the barrier for scientists attempting to exploit their nonlinear systems for computational tasks typically carried out in the fields of machine learning and artificial intelligence. A simulation tool to accompany this paper is available online[7].

Contents

[7] https://github.com/stevenabreu7/handson_reservoir.

1. Motivation

Artificial intelligence (AI) permeates close to every technological and scientific discipline, as well as our everyday lives. Considering (i) the enormous amount of data that next-generation communication technology (e.g., 6G, internet-of-things, edge computing, etc) is expected to produce, and (ii) the commercial interest in identifying patterns in unstructured data, the development of AI is foreseen to continue growing. Its rise in the past 20 years was boosted thanks to the production of faster digital processing units alongside the development of more efficient algorithms, the so-called artificial neural networks (ANNs) that, inspired by the parallel architecture of neural connections in the human brain, are capable of learning and evolving over time (see figures 1(a)–(c) for an overview on different neural network architectures). Meanwhile, digital computing had decades to develop tools for aligning hardware with software, and to build sophisticated software tools to implement ever more complex programs. This is possible because digital computing restricts the domain of its applicability to systems with bi-stable dynamics (e.g., representing the logic states 'true' and 'false') and negligible transient states. Such meta-stability is crucial as it ensures close to perfect reproducibility of a program considering hardware fabrication tolerances.

Considering these points, it becomes evident that the analogy with the brain is only superficial. Indeed, the human nervous system carries out computation through elegant and sophisticated spatio-temporal dynamics, where transients and noise are not only unavoidably present, but important. This allows our brain to process analog signals with excellent precision and power efficiency (the brain drains 20% [2, 3] of the body's energy, which amounts to an overall power-consumption estimated at around 20 W). Such energy requirements are impressive compared to digital machines, which easily exceed 100 W during inference and several kilowatts during the training of ANNs. In terms of speed, the brain is estimated to perform 100 teraflops per second, while traditional computers lag by orders of magnitude, with their computational speed limited by the serial communication between computing and memory units (the von Neumann bottleneck). Therefore, with the goal of building general-purpose, autonomous machines capable of 'human' capabilities such as cognitive learning and continuous adaptation, exploring unconventional computing paradigms as well as understanding the fundamental computational principle of the brain and implementing them on artificial devices becomes

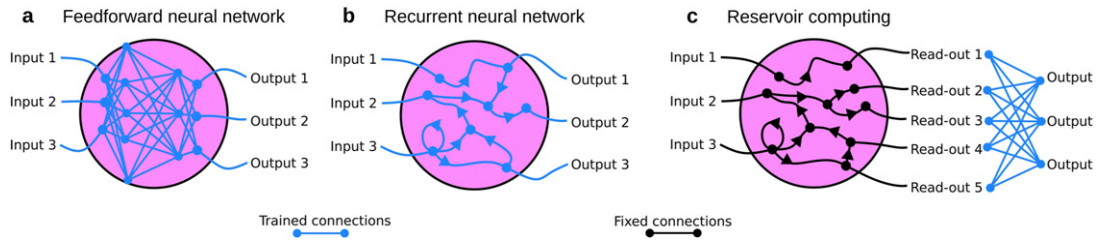

Figure 1. Illustration of (a) a feedforward neural network (FNN) with its acyclic and trainable connections, (b) a recurrent neural network featuring trainable feedback and cyclic connections, and (c) a reservoir made of untrained connections. For RC, only the connection between readout layer and output layer is trained, typically with a linear regression.

essential. However, this should not be taken too literally. Indeed, it is an ongoing debate on what abstraction of brain dynamics is most helpful in building intelligent machines. What can be stated with certainty is that the brain exploits its underlying physical substrate for computation, and we ought to do the same if we want to reach similar levels of energy efficiency and performance. Continuing to exclusively impose digital computing's symbolic nature on intrinsically dynamical and analog devices may be a wrong choice in the ANN context. Therefore, a fundamental rethinking of computational machines must aim at maximizing the direct use of the material as computing substrates, without sharp distinctions between computational unit and memory unit.

Unconventional computers aim to harness generic physical systems for computation, along with a way of 'programming' any such physical system. However, there exists a gap between descriptions at the physical and the computational level. This is partly because we lack a general framework that unifies 'computation' in dynamical systems and computation in symbolic systems, as outlined in [4]. On the practical side, we also lack methods of 'programming' physical systems, though recently, some advances have been made, e.g., using back propagation through physical media [5] or more practical implementation scheme that does not require to know the precise equations of physical reservoir [6–8]. For these reasons, exploring new computational paradigms in hardware-based neural networks has been in the spotlight. For example, dense arrays of interconnected elements (artificial synapses) whose resistance can be accessed and programmed have been shown [9]. Such elements must allow a reversible and energy-efficient mechanism to change their resistance (writing phase). Moreover, minimal to no device mismatch often is a prerequisite for computational precision [10]. Hence, an ANN incapable of tweaking its connections with accuracy cannot provide a suitable substrate for machine learning in its classical sense, and learning mechanisms that work in software-based ANNs may not work well or are unrealistic in unconventional hardware.

In this framework, reservoir computing (RC) and random neural networks come into play, as they offer power-efficient in-memory computing in random networks (see figure 2) with relaxed requirements on the accurate tweaking of synaptic connections. Physical reservoir computing (PRC) takes advantage of machine learning so that programs need not be fully specified and instead may be learned autonomously from data and teacher signals. Moreover, it is also naturally formulated in a dynamical systems context, a formalism that enables material scientists to map their physical systems onto the RC framework. This has led physical RC to become one of the leading paradigms for exploiting high-dimensional, nonlinear, dynamical substrates for computation. It is a simple, versatile, and efficient framework for physical computation.

However, simply applying the RC framework to physical systems often only serves as a proof of concept that some substrate can be made to 'compute'. One may also exploit the system for its physical properties that are not directly connected to its computational purpose. As an example, optical reservoirs [11] can be used for applications where extreme speed is desirable, while ionic or electrochemical reservoirs [12] could be used to couple biological interfaces where high speed is not an essential requirement. One might go even further and develop extensions or adaptations of RC for specific substrates that enable to take advantage not only of the sheer presence of complex dynamics but also of the specific properties of such dynamics that are offered by a specific physical substrate.

It has been argued that to fully domesticate the powers of such complex dynamics for computational purposes, it is necessary to develop more complete theories of computation [4]. At the very least, a better theoretical understanding of the connection between dynamics and computational function is required for full-scale development of physical RC [13, 14]. One way to make progress towards such a framework for general computation is through closer collaborations between material, dynamical systems and computer scientists in order to better characterize the kinds of computation that are possible in complex systems.

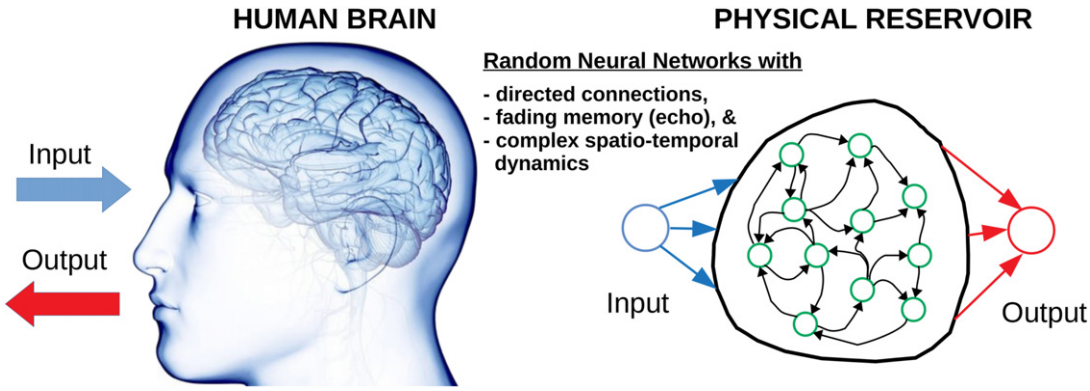

Figure 2. The human brain, as well as a physical reservoir can be seen as a random neural network with directed connections, fading memory, complex spatio-temporal dynamics. Memory and spatio-temporal dynamics are properties of the underlying substrate/material (e.g., in the case of the brain it is an electrolytic environment in which complex, spatially non-uniform electrochemical reactions occur. In a physical reservoir, different from the brain, training is done in the output layer, which is not a part of the reservoir.

This paper is our contribution to enabling better communication and fostering collaborations across these two fields to develop better physical implementations of intelligent computing machines. We organized this tutorial as follows. First, we provide a detailed, conceptional introduction to the field of RC, discussing reservoir dynamics, the importance of a fading memory, training of reservoirs, and RC. Second, we discuss the implementation of a RC experiment in hardware using a simple, hands-on example. The data of this hands-on example is publicly available, allowing the readership to follow every step of the experiment in detail. The material scientist eager to start an RC experiment with his/her system at hand might even consider skipping the introductory theory chapter and directly start reading our 'recipe' for RC in chapter 3. Reading the introductory chapter after the practical experience might then offer an enriching and inspiring perspective on the topic. To guide readers not coming from the AI community but still aiming to start with chapter 2, we summarize the essential aspects of RC in bulleted lists. Furthermore, we mark sections and subsections in chapter 2 with an asterisk that are not strictly needed for an introduction to the field. Finally, after having said so much about the implementation of physical reservoirs, we will summarize model systems and applications for physical reservoirs that have been discussed in the literature, including electronic, photonic, mechanical, chemical, and liquid state reservoirs.

2. Introduction to reservoir computing

We start this section with a brief historical review of the development of RC. Then, to provide a broad perspective, we connect the idea of RC with other disciplines such as machine learning, ANNs, dynamic system theory, and computational neuroscience. Finally, we clarify the term computation, which will be the starting point for the practical introduction to RC in the following sections.

Most introductions to RC begin with echo state networks (ESNs) [15, 16], and liquid state machines (LSMs) [17], but the ideas from RC can be traced back further into the past [18]. The principle behind RC was present already in the context of reverberation subsystems by Kirby [19], who even pointed out the possibility of hardware implementations for AI applications. Schomaker [20] described a reservoir-like system with spiking neural oscillators. In the cognitive neurosciences, Dominey [21] used a reservoir as a simplified model of the prefrontal cortex, which has led to fruitful research that continues to the present [22]. The idea behind RC has been (re)discovered multiple times. This may be on account of the conceptual simplicity of computing functions through a high-dimensional temporal expansion of the input (see section 2.1). The term RC can be traced back to [23], in which the authors experimentally unified ESNs and LSMs into a general framework of computation using a reservoir.

RC research flourished through collaborations between early pioneers. This resulted in powerful practical demonstrations and the development of a mathematical theory to ground the field. One of the early motivations for using RC as an effective training mechanism for recurrent ANNs has been largely superseded by deep learning methods, which have mastered the difficulties of gradient-descent training that RC bypasses (see section 2.2). Nevertheless, RC research is thriving [24] in a variety of fields, which reflects the many perspectives that can be taken in the study of RC:

- In machine learning, RC presents a straightforward-to-implement and efficient learning algorithm for temporal input–output mappings by a nonlinear temporal expansion (see section 2.4).
- In research on ANNs, RC presents an efficient training mechanism. ESNs [15] were originally introduced in the context of machine learning with ANNs.
- In dynamical systems theory, RC helps approximate potentially complex dynamical systems. In the field of input-driven nonlinear dynamical systems, reservoir systems themselves provide an interesting topic of investigation [25, 26].
- In computational neuroscience, RC offers a simple model of cortical computation. LSMs [17] were introduced in this context.
- For materials and nonlinear systems scientists or device engineers, RC provides a simple method to harness complex dynamics for computation or, simply, a way to program a physical system. This is often referred to as PRC [13, 27] which has roots in RC as well as unconventional computing [28].

For the last point, it is instructive to clarify that computation is used here wider than in the classical theory of digital, i.e. symbolic, computation. In this general sense of computation, we may define a computation as anything that happens inside a system that transforms input signals into output signals. A more detailed discussion on such an extended meaning of computation can be found in references [4, 29].

2.1. General principle of reservoir computing

Given this broad view of computation, the general principle of RC is intuitive. It allows the construction of a dynamical system that implements a desired mapping from input signals to output signals. This includes time series prediction, classification, and other tasks. To this end, a set of examples (training data set) that demonstrates the desired relation between input signals and output signals is usually required. In machine learning terminology, this is a temporal supervised learning task (see section 2.4).

In this setup, a training input signal $\mathbf{u}^{\text{train}}(t) \in \mathbb{R}^{N_u}$ with a desired target output signal $\mathbf{y}^{\text{train}}(t) \in \mathbb{R}^{N_y}$ is given, where N_u, N_y is the dimensionality of the input and output, respectively. In time series prediction, the input signal may show a time series and the target output signal its continuation. In classification, the input signal may present an auditory speech signal and then target output signal the corresponding phonemes that are spoken. Note that if multiple of such input–output examples exist, they may simply be concatenated in time. What is desired is a system that, when fed with input $\mathbf{u}^{\text{train}}(t)$ generates an output signal $\hat{\mathbf{y}}^{\text{train}}(t) \approx \mathbf{y}^{\text{train}}(t)$.

We may assume that the examples are generated by a function $\Psi(\mathbf{u}^{\text{train}})(t) = \mathbf{y}^{\text{train}}(t) + e$ where e is an additive error term, e.g. noise, and $\mathbf{u}^{\text{train}}$ is the entire input time series. In signal processing, such a system is called a filter. The reservoir system, denoted by $\hat{\Psi}_\theta$ (where θ is the vector of model parameters that determine the reservoir system), then aims to approximate the true generating system Ψ as closely as possible.

The solution of RC to this learning task comes in three steps.

(a) The reservoir is prepared as a high-dimensional, nonlinear dynamical system that can be driven by the input signal $\mathbf{u}^{\text{train}}(t)$. For a well-performing system, the collective dynamics of the reservoir should satisfy conditions summarized in sections 2.4 and 3.1. However, the individual components of the reservoir can be random. The reservoir's high-dimensional states are given by $\mathbf{X}(t) \in \mathbb{R}^N$, and many of its state variables must be accessible to be read out as $\mathbf{x}(t) \in \mathbb{R}^{N_x}$, which consist of $x_i(t)$ for $i = 1, \ldots, N_x$. Not all states of the reservoir need to be observable, it is possible that $N_x < N$.

(b) The reservoir is driven with the input $\mathbf{u}^{\text{train}}(t)$ and the corresponding state variables of the reservoir in response to this input are read out and stored as $x_i^{\text{train}}(t)$.

(c) Lastly, a readout function F, which maps all recorded state vectors $x_i^{\text{train}}(t)$ to an output $\hat{\mathbf{y}}(t) \approx \mathbf{y}(t)$ is learned (estimated):

$$F(x_1^{\text{train}}(t), \ldots, x_N^{\text{train}}(t)) = \hat{\mathbf{y}}(t) \approx \mathbf{y}(t) \tag{1}$$

This is done by minimizing a chosen loss function $L(\mathbf{y}, \hat{\mathbf{y}})$, which evaluates the difference between the reservoir's output $\hat{\mathbf{y}} = \hat{\Psi}_\theta(\mathbf{u})$ and the desired output $\mathbf{y} = \Psi(\mathbf{u})$. Specifically, we minimize the average loss over the given training data:

$$\bar{L}^{\text{train}}(\hat{\Psi}_\theta) = \frac{1}{N_{\text{T}}} \sum_t L(\mathbf{y}^{\text{train}}(t), \hat{\Psi}_\theta(\mathbf{u}^{\text{train}}(t))) \tag{2}$$

where N_{T} is the number of time steps. The readout function F, which minimizes the average loss, is typically obtained with a simple linear regression.

After this procedure, the reservoir can be used for inference to make predictions. When feeding a new input signal $\mathbf{u}(t)$ to the reservoir, the reservoir's state vectors $x_i(t)$ are observed and used to compute the predicted output signal $\hat{\mathbf{y}}(t) = F(x_1(t), \ldots, x_N(t))$.

We model the reservoir as a network of interacting elements or a neural network (see section 2.2). This is not necessary, it is also possible to model the reservoir as a spatially continuously extended medium, but this is outside of the scope of this tutorial.

2.1.1. Advantages of reservoir computing

A major advantage of RC is its versatility. Firstly, this scheme can be used for any temporal input–output task. Secondly, many nonlinear, high-dimensional dynamical systems can be used as a reservoir (see section 3.1). The reservoir itself does not need to be modified or trained, which allows the use of a wide variety of (physical) systems, the dynamics of which need not be characterized (or even understood). Section 2.2 shows how RC is applied to neural networks, and section 2.3 shows how RC is applied to harness arbitrary dynamical systems for computation. Furthermore, the training procedure is also straightforward. Often the readout function can be learned through simple linear regression, which is computationally inexpensive to implement on a digital computer connected to the physical reservoir.

The simplicity originates from the random configuration of the reservoir itself, which does not need to be learned, but instead can remain fixed. Additionally, a single reservoir can also be used to learn multiple different mappings from the same input signal. Given, for example, an electrocardiogram signal (ECG), a single reservoir can be used for, both anomaly detection in a single heartbeat and simultaneously for classification of the ECG signal as a whole. To do this, one simply trains one readout function for the anomaly detection and another for the classification. In inference mode, the reservoir is driven by an ECG signal, and both readout functions can be used simultaneously to obtain two different output signals from the same reservoir.

This is especially suitable for physical implementations in materials because the reservoir itself does not need to be modified during training. In RC, only the states of the neurons in the reservoir must be recorded during training, while the weights can be left unchanged.

Beyond its simplicity, RC is also extensible. Many additions to the basic RC formulation have been proposed, for example, the use of local learning rules inside the reservoir [30, 31], the use of multiple reservoirs as part of a learning ensemble [32] or using a 'deep' architecture in which each layer is a reservoir [33]. Such extensions to the basic RC setup are briefly outlined in section 3.5.4.

Reservoirs have been particularly successful in learning and predicting the dynamics of chaotic systems [34, 35], so RC is a competitive approach for tasks that deal with chaotic signals.

2.2. Training the reservoir

Although any complex dynamical system can be used as a reservoir (see next section), the reservoir is often modeled as a neural network due to its roots in ESNs and LSMs.

Neural network models are abstractions of biological neural networks in the brain. Investigated initially to explain the mechanism and function of neural circuits in the brain [36], computer scientists have adopted neural networks in technological applications. Such ANN models are used in machine learning as they are driven mainly by technological applications and consequently trade off some of their biological plausibility. The field of deep learning has become massively successful by combining large ANN models and training these with machine learning techniques.

Before comparing the reservoir approach to training with the deep learning methods, some definitions are needed. We categorize different types of ANNs based on their underlying neuron model and their interconnections, or architecture.

2.2.1. Recurrent vs feedforward neural networks

An ANN can be represented as a directed graph, where nodes represent neurons and edges represent one-way connections between neurons. We differentiate between feedforward neural networks (FNNs), which implement a directed acyclic graph (figure 1(a)), and recurrent neural networks (RNNs), which implement a directed cyclic graph (figure 1(b)). In a FNN, the network can be organized in L layers, where neurons in layer i are connected only to neurons in layer $i+1$, through a weight connection matrix W^i. By convention, the first layer $i = 0$ is the input layer, the final layer $i = L$ is the output layer, and in between are hidden layers. This leads to the following transfer function from layer i to layer $i+1$:

$$\mathbf{x}^{i+1} = f\left(W^i \mathbf{x}^i + \mathbf{b}^i\right) \tag{3}$$

where f is a nonlinear activation function, commonly $f = \tanh$, $\mathbf{x}^i$ is the state vector of neuron activations in layer i, and $\mathbf{b}^i$ is the bias vector for layer i. Each neuron in the neural network has a bias, which can be interpreted as a scalar offset for its activation value.

A FNN is a stateless input–output map where neuron activations spread through the network from the input layer through the hidden layers to the output layer. If the input layer is inactive, subsequent layers will also be inactive. In contrast, RNNs can sustain reverberations of activations after the input layer becomes inactive, giving rise to temporal dynamic behavior which can be used as memory. This makes RNNs particularly well-suited for temporal data processing. RNNs are more common in biological neural networks, as even the simplest nervous systems in the animal brain feature high recurrence of the neural sub-circuits.

RC exploits temporal dynamics. Therefore it is formulated for RNNs. However, an approach similar to RC but without temporal dynamics can be implemented in FNNs. These are investigated under the name of extreme learning machines [37].

2.2.2. Training artificial neural networks

As shown above, a FNN implements an input–output mapping, similar to a computer program written in, say, Python. The important difference is that a human programmer must come up with the Python program, which is easy enough for some programs, but time has shown that some programs are simply too complex to be written manually. What makes ANNs relevant is that we do not have to manually program them, but that we can set up a training procedure in which the ANN learns to improve its performance from a set of examples, which we call the training data. We say that a network is trained if it has already learned to implement some desired function. The network's behavior is then frozen, such that it does not change anymore, and we say that it is used only for inference, or evaluation.

The ANN's transfer function $\hat{\Psi}_\theta(\mathbf{u})$, which maps its inputs to outputs, is parameterized by $\theta \in \mathbb{R}^{N_\theta}$. In general, the set of parameters can include any of the ANN's properties, hence including its architecture. The architecture of an ANN usually includes the connectivity graph and the choice of activation function. However, usually only the connection weights and biases are taken as parameters and the architecture is fixed.

The standard way to train ANNs is by gradient descent, where the current parameters θ are updated to θ' in the direction of the average loss function's steepest descent: $\theta' = \theta - \eta \nabla_\theta \bar{L}(\hat{\Psi}_\theta)$ where ∇_θ is the gradient with respect to the parameters θ and η is the learning rate, or step size. This update step is performed iteratively, until the network converges to some minimum in the error function. In stochastic gradient descent (SGD), the update step is performed after every $k \in \mathbb{N}$ data points while in batch gradient descent the update is performed on the whole data set. In SGD with $k > 1$, one also speaks of mini-batch gradient descent.

An ANN is a good solution to the learning task if it not only minimizes the error function on the training data, but is also able to generalize to unseen validation data with low error. This generalization ability implies that it is not enough to simply memorize the training data. Such generalization is a hallmark of intelligence, and a major challenge in machine learning. One standard way to ensure that a training procedure optimizes an ANN to generalize well to unseen data is through cross-validation (CV), see e.g., chapter 5 in reference [38].

The gradient of the empirical risk function with respect to the parameters is computed with the backpropagation algorithm [39] during a forward pass and a backward pass. In the forward pass, the input is propagated through the network to the output layer, where the average loss $\bar{L}(\hat{\Psi}_\theta)$ is computed. In the backward pass, this error is back-propagated through the network, and at each layer, the gradient of the error with respect to the layers' parameters $\nabla_{\theta_i} \bar{L}(\hat{\Psi}_\theta)$ is computed, effectively applying the chain rule of differentiation.

In theory, this works for networks of any depth, but in practice the performance of gradient descent with the standard backpropagation algorithm declines with larger networks. Training can take very long until it converges, and the problem of exploding and vanishing gradients can prevent convergence altogether.

In RNNs, the training becomes even more challenging. The input and output are now time series $(\mathbf{u}^{\text{train}}(t), \mathbf{y}^{\text{train}}(t))$. In RNNs, the output of the model depends on the current input $\mathbf{u}(t)$ and also on the current state of the network $\mathbf{x}(t)$ which can be seen as the network's memory, encoding information about past inputs and initial conditions. The backpropagation algorithm has been adapted for RNNs, under the name of backpropagation through time (BPTT). BPTT essentially looks at the RNN as a deep FNN, by unrolling the network in time.

Many of the challenges of backpropagation become amplified in BPTT and convergence is not generally guaranteed in BPTT. RC provides an alternative hardware-friendly method to train RNNs.

It should be pointed out that since the introduction of RC in the early 2000s, deep learning methods have been developed to enable BPTT to work reliably for large RNNs. These methods have become the standard way to train complex, large ANNs. However, RC still provides several advantages for training neural networks, as will be pointed out next.

2.2.3. The reservoir approach to train neural networks

RC is a method to train neural networks that does not require the backpropagation of error information through the network. Neither does RC require the computation of gradients inside the network. Both of these make BPTT expensive in terms of time, memory, and energy required to train a neural network. RC offers a cheaper alternative. Moreover, RC can also be used to train non-differentiable neural networks. This enables RC to train a neural network (or, in general, a reservoir) whose dynamics are not fully described, as is often the case in physical systems, or simply not differentiable, as is the case for spiking neural networks (SNNs).

The general setup for training a reservoir (or rather: the readout function) was already explained in section 2.1. It is important to note that step (a), the preparation of the reservoir, is not trivial. As the present section has shown, a reservoir ought to satisfy several conditions to perform well on a learning task. Section 3.5 will give more practical guidance for the tuning of a physical reservoir.

2.3. Physical reservoir computing

PRC merges the key ideas of ESNs and LSMs, which were both initially only implemented in software simulations, into a general framework for computation in physical systems. Just as in simulations, a physical reservoir applies a nonlinear temporal expansion to the input signal by mapping it into the high-dimensional reservoir, in which different input signals are separable. The physical reservoir should have some short-term memory and is typically required to have the echo state property (ESP) in order to wash out initial conditions (see section 2.4). A task-specific readout mechanism then maps the reservoir state to an output state, effectively implementing the desired input–output function with the help of the reservoir. This readout function can be implemented directly in the material or through an interface to a digital computer.

As a framework for computation in physics, PRC brings computing and physics closer together. This advances the development of next-generation neuro-inspired computers [40], and can enable the development of computers using materials that have never been used for computation before [13, 41].

2.3.1. Physical artificial neural networks

The quest of finding materials and devices well suited for physical neural networks has been taken up by material scientists, physicists, and engineers in the last decade, and the challenges are summarized in a recent roadmap [10]. In these early strides of building artificial 'brains', it is not clear if information is processed at the synaptic, neuronal, or network level. However, it is reasonable to begin with designing the smallest and most numerous unit of the nervous system, the synapse. In this framework, many devices with tunable properties have been proposed as artificial synapses for neuromorphic computing and employed as edges between nodes, where the tunable property (resistance, transparency, etc) is the weight. The vast scientific interest in the development of such artificial synapses and research in this direction is mainly driven by materials and device innovation [42–44]. In order to produce functional ANN, though, such artificial synapses must be integrated into large and dense arrays of synaptic elements and connected to artificial neurons. At this stage, two obstacles hamper the development of hardware-based ANNs. Firstly, there is often a high barrier for scientists working on single-device artificial synapses to scale up and employ these synapses for typical AI tasks. Secondly, basic concepts of machine learning (e.g., backpropagation, weight tuning, etc), while being well suited to symbolic digital machines, may not work reliably on real systems as these usually cannot meet requirements for device-to-device variability and/or precise control over each individual unit/synapse [45]. The precise fine-tuning of ANNs could be circumvented by the employment of random, semi-random, or unconsciously designed ANNs, which restrict optimization to an output layer. Such a strategy is more appealing and suitable for physical systems and underlies several brain computing principles.

2.3.2. Random artificial neural networks

ANN can recognize patterns by undergoing a phase in which edges (synapses) are tweaked, in order to enhance or inhibit the activity of the connected nodes (neurons). Increasing the complexity of an ANN ensures high computational power, but a larger amount of connections will require longer time to train. With this in mind, two interesting questions arise at this stage.

(a) Considering the incredibly high synaptic density in our brain (higher than 10^{11}cm^{-3} [46]), how can each synapse be chemically targeted and adjusted in our nervous system with precision?

(b) Can an ANN be used with a random value assigned to each edge?

Considering the human brain's complexity, it appears almost impossible, and surely impractical, to target single cells and individual synapses for each potentiation and depression event. While the biological and chemical mechanisms behind the single-cell behavior have been laid down quite thoroughly, the dynamics behind larger ensembles of neurons remain largely unexplained. This holds even for primitive systems e.g., the widely studied *worm Caenorhabditis elegans*, with its tractable neural connections.

This brings us to the following question: if a network, biological or artificial, is hard or inefficient to customize in each of its building blocks, how much accuracy and computational power would be lost? It turns out that even randomly assembled neural networks (RaNNs), not only can compute precisely, but possess important advantages over other types of fine-tuned neural networks [47, 48]. For one, using a RaNN makes the configuration process cheap and simple because the network only needs to satisfy some constraints, but can otherwise be random. Random neural networks are also more flexible as they can be implemented in many physical substrates, including those that do not allow complete access to modify the network. Finally, random neural networks are often more versatile in that the same random network can be used for multiple different tasks—just as the same reservoir can be used to implement different functions by using different readout layers.

'Random' means that connections are not trained. However, the rules that govern the communication between neurons only allow for a certain probability distribution of the outcome, but an exact prediction is impossible. Also, RaNNs have been shown to be universal approximators for bounded and continuous functions.

The idea of a random computational machine is in fact paradoxical to our intuition, and contrasting with the high-precision arrangement of the components within a circuit board paramount in nanoelectronics. However, it can be imagined that small circuit units equipped with feedback loop responses may adjust their behavior depending on their input, output and back-fed signal. Feedback loops are in fact a precious circuit element in electronics, as well as a powerful chemical and neuronal mechanism harnessed by biology. By doing this, we are moving towards a 'learning' approach that is carried out by a dynamic, semi-random computational structure evolving over time. Also, it is worth noting that when we mention feedback loops, we are referring to systems in which the information flow is not mono-directional, rather totally undefined, as it happens in RNNs. On top of this, when imagining the implementation of a RaNN on hardware, the effort for its training is substantially relieved.

2.4. Reservoir computing theory

The term RC pays tribute to the central importance of the reservoir in this framework. The reservoir acts, both, as a memory and as a nonlinear temporal expansion of the input:

- The memory is realized through the recurrent connections of the reservoir, which make its current state dependent on previous states with a fading memory, just as in recurrent neural networks (RNNs, see figure 1(b)). However, as opposed to RNNs, a RC's recurrent connections are not trained. As such, the reservoir is often described as a 'black box' (figure 1).
- The reservoir realizes a nonlinear temporal expansion, mapping the input signal to a high-dimensional reservoir state. Such a projection into high-dimensional space is commonly used by kernel methods in machine learning, or by random neural networks (which are not unlike RC). Through this, different input signals can become more easily distinguishable. As schematically shown in figure 3, a linearly inseparable data set in 2D can become linearly separable in 3D after a $\mathbb{R}^2 \to \mathbb{R}^3$ mapping. The same principle applies to the expansion of the low-dimensional input signal into the high-dimensional reservoir state space. Moreover, the dynamical (short-term) memory makes past inputs interacts with the current input. The reservoir therefore also provides a temporal expansion of the input: the reservoir's response depends on the entire history of inputs (and the initial condition of the reservoir), i.e. $\mathbf{x}(n) = f(\ldots, \mathbf{u}(n-1), \mathbf{u}(n), \mathbf{x}(0))$ in the discrete-time case, which is usually rewritten into the recursive form $\mathbf{x}(n) = f(\mathbf{u}(n), \mathbf{x}(n))$. If such memory of past inputs is not necessary for the task at hand, non-temporal machine learning techniques, e.g. extreme learning machines [37], should be used.

In-memory computing, referring to the co-location of memory and computation as in reservoirs, has recently received much attention for its promise to avoid the von Neumann bottleneck, which limits certain applications in conventional computers.

2.4.1. Reservoir dynamics

In this section, we will describe the mechanisms that influence reservoir dynamics from a theoretical perspective, while section 3.5 will explain how to tune a physical reservoir in practice. While RC requires less tuning

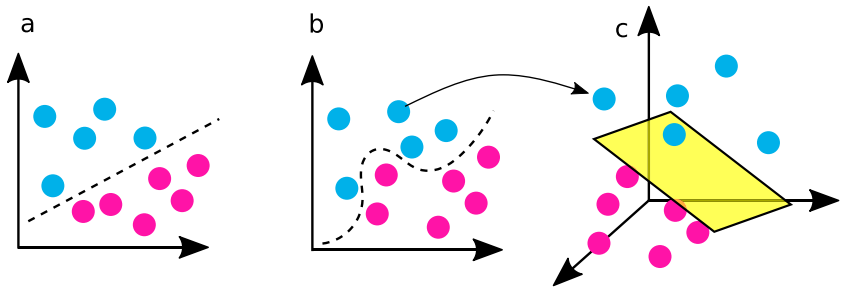

Figure 3. (a) Example of a linearly separable data set, where pink and blue circles can be divided by a straight line. In this case, a linear regression is sufficient for a successful classification task. (b) Example of a linearly inseparable data set in 2D. (c) Data points from (b) are projected nonlinearly into a higher dimension (3D). This projection into 3D space can be used to obtain linearly separable data (through the separating plane shown in yellow).

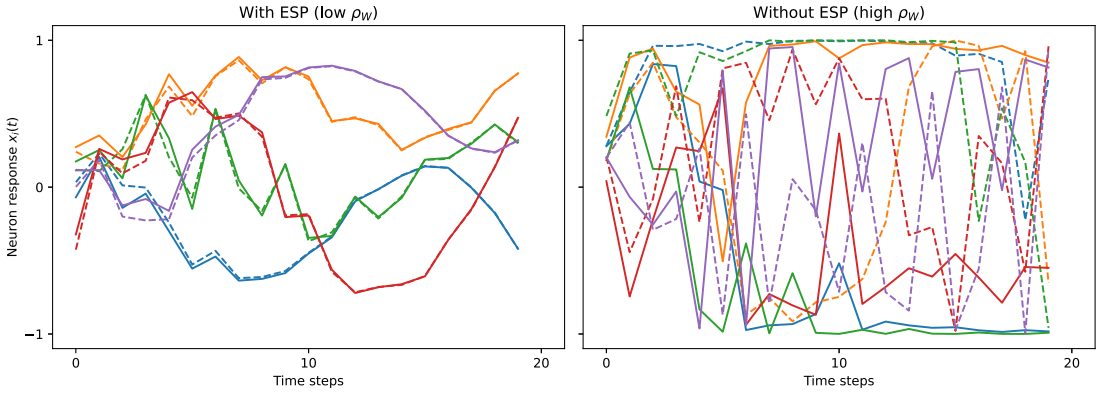

Figure 4. Shown are the response of five randomly selected neurons from two ESNs, one of which possesses the ESP (left) whereas the other does not (right). Both reservoirs are driven by a sinusoidal input signal. The dashed and solid lines show different initial conditions of the reservoir state at $\mathbf{x}(0)$. The ESN on the left has a spectral radius of $\rho(W) = 1.25$ and the plot shows that neuron responses with different initial conditions converge, thus demonstrating the ESP. The ESN on the right has a larger spectral radius of $\rho(W) = 3.0$ and the plot shows that neuron responses with different initial conditions do not converge, showing the lack of the ESP. For further explanation, see text.

and configuring of a physical system compared to e.g. hardware-based ANNs, the truth is that not every reservoir will work well in practice. For example, it is possible that the reservoir's internal dynamics are so strong that it will disregard the influence of the input signal. This is clearly undesirable. A reservoir should depend mainly on the current input, and wash out the effect of past inputs and initial conditions. Putting it another way, the reservoir's response $\mathbf{x}(t)$ at time t should be uniquely defined by the fading history of the input $\mathbf{u}(t)$, and not on the initial conditions of the reservoir $\mathbf{x}(0)$. The importance of this property is illustrated visually in figure 4 and explained in more detail in the following.

In the context of ESNs, the above-described condition is formalized as the ESP [15]. The ESP essentially states that the effect of a previous state $\mathbf{x}(t)$ and a previous input $\mathbf{u}(t)$ on a future state $\mathbf{x}(t+T)$ should vanish gradually as time passes, i.e. $T \to \infty$. In the context of LSMs, this condition is formalized as a fading memory. A reservoir has fading memory if the most significant bits of its current response $\mathbf{x}(t)$ depend just on the most significant bits of the values of its input $\mathbf{u}(\cdot)$ in some finite time interval $[t-T, t]$.

In ESNs, a sufficient condition for the ESP property is that the largest singular value of the internal weight matrix W be below unity, $\bar{\sigma}(W) < 1$. However, this condition is very restrictive and not commonly used in practice because the input can potentially wash out too quickly [49]. A less restrictive, commonly used condition is that the largest eigenvalue of the weight matrix W be below unity, $\rho(W) < 1$. The value $\rho(W)$ is also called the spectral radius of W. Although this does not guarantee the ESP in general [49], it tends to work well in practice [1].

*2.4.2. Reservoir memory**

The memory of a reservoir is not straightforward to quantify, as it is generally not possible to state a reservoir can only 'remember' up to T time into the past, and, at least in principle, a reservoir has unbounded memory length. However, a reservoir's memory fades, and it will fundamentally not be able to perform well in a task that requires longer memory. This implies that it is impossible to train a reservoir on tasks

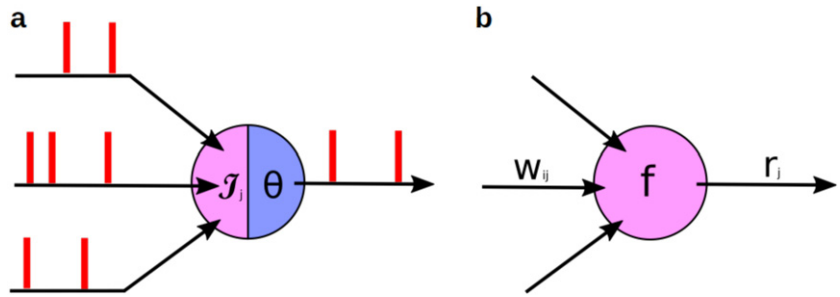

Figure 5. Comparison of neuron models. (a) Shows a biologically more plausible spiking neuron for SNNs ($\mathcal{J}_j$ and Θ denote the integration and threshold function, respectively) and (b) shows an artificial neuron used in ANNs where f denotes the nonlinear activation function of the neuron.

which require unbounded-time memory [50], and the depth of this fading memory is therefore essential information. Later in this section, we shall see that it is possible to equip a reservoir with an external working memory, but for now we restrict the discussion to the standard formulation of RC. The memory capacity of a reservoir has been analyzed for linear neurons in discrete time with activation function $f(x) = x$ [51], and more recently also for continuous reservoirs [52] and nonlinear reservoirs in discrete time [53].

To quantify the memory capacity C of a reservoir, one commonly trains a reservoir to predict, or memorize, an input signal. Hence, for a given input signal $\mathbf{u}^{\text{train}}(t)$, the desired output signal is $\mathbf{y}_T^{\text{train}}(t) = \mathbf{u}^{\text{train}}(t - T)$. The error in this learning task will tend to increase for larger values of T, i.e. for memorization further into the past. For a discrete-time reservoir like an ESN, the memory capacity C is defined as $C = \sum_{T=1}^{\infty} r^2(\mathbf{u}^{\text{train}}(t - T), \mathbf{y}_T^{\text{train}}(t))$, where r^2 is the squared correlation coefficient [50].

Other memory mechanisms also exist in ANNs. Most notably are working memory and long-term memory. As per reference [54], working memory is 'the ability (of a system) to transiently hold and manipulated goal-related information to guide forthcoming actions'. As such, a working memory allows a reservoir to store information for potentially unbounded time spans. This stands in direct conflict with the above-mentioned fading memory, or echo state, property of reservoirs, thus standard reservoirs cannot implement working memory. A working memory mechanism for ESNs was described in reference [55] by adding reservoir-external working memory units. Long-term memory stores information for unbounded duration and has to have a much larger, possibly even unbounded, storage capacity. The readout function of the reservoir can be considered a form of long-term memory. After training, the readout function contains task-specific information, i.e. how to compute the output from the reservoir's response to an input, and this information persists indefinitely. It may be thought of as a type of procedural memory, which aids the performance of a particular kind of task.

*2.4.3. Neuron models: spiking vs rate-based**

Biological neurons communicate through spike events when they are sufficiently excited through their input signals. RC can work with either of two categories of neuron models: artificial neurons that are commonly used in machine learning and spiking neurons that are commonly used in computational neuroscience and neuromorphic computing.

In artificial neurons, the spiking activity is modeled by averaging the number of spikes per time interval to yield a rate-based activity measure. They are described by difference equations in discrete time. In contrast, spiking neurons model their activity with individual spike events rather than an average and they are described by differential equations in continuous time.

Moreover, depending on how the information is represented (discrete spike events in continuous time as in figure 5(a) or time-averaged activity in discrete time as in figure 5(b)), neuron models can be classified as ANNs or SNNs. Both ANNs and SNNs have been used in RC: LSMs were introduced using spiking neuron models while ESNs were introduced using rate-based neuron models.

For clarity, we restrict formal discussions to networks of rate-based neuron models. They are simpler and more efficient to simulate on digital computers. For this reason they have become the standard neuron model in machine learning, where performance is prioritized over biological plausibility. Rate-based neuron models are also frequently used in hardware-based neural networks as they are easy to implement in digital hardware. However, also hardware-integrated spiking neurons are gaining increasing interest in applications. In particular, leaky integrate-and-fire (LIF) models are frequently used due to their strikingly simple architecture and high speed. Although more difficult to integrate in hardware, biomimetic spiking neuron models are preferred

though due to their excellent energy efficiency and rich temporal dynamics, which is key for computational power [56].

In ANNs, a neuron's state, its activation x, represents its average firing rate. At each discrete time step n, the input of a neuron i is the sum of every neuron j's activation at the previous time step $n-1$, weighted by the strength of the synaptic connection from neuron j to neuron i, given by $w_{ji} \in \mathbb{R}$. The neuron's activation $x_i(n)$ is then given as some nonlinear function f on this input:

$$x_i(n) = f\left(\sum_j w_{ji} x_j(n-1)\right). \tag{4}$$

The equation can also be expressed in matrix form with the weight matrix W whose element at row i and column j is given by w_{ij}

$$\mathbf{x}(n) = \mathbf{f}(W\mathbf{x}(n-1)) \tag{5}$$

where $\mathbf{x}(n)$ is the activation vector for all neurons at time step n and $\mathbf{f}$ is the vectorization of the function f.

Equation (5) shows that the dynamics at each time step can be computed as a matrix multiplication, or multiply-and-accumulate (MAC) operation, followed by a nonlinear function. Chaining these two operations is all that is needed to build a hardware implementation of a static ANN. The fact that GPUs can efficiently compute MAC operations shows why ANNs dominate the field of machine learning.

In SNNs, a neuron's state represents its momentary activation and the neuron will emit a spike event when it is sufficiently activated. Such a spike event, or action potential, is modeled as an all-or-nothing, binary event. The activation of the neuron models the membrane potential V_m, as in biological neurons. When the membrane potential V_m reaches a critical threshold V_thr, the neuron emits a spike to all neurons to which it is connected. This activates downstream neurons by raising their membrane potential. The details of this spike transmittance depends on synaptic and dendritic dynamics. These are often neglected, for example in the commonly used LIF model of a neuron, in which the membrane potential $V_\mathrm{m}(t)$ of the neuron is given by the differential equation

$$\frac{dV_\mathrm{m}(t)}{dt} = k_1 I(t) - k_2 V_\mathrm{m}(t) \tag{6}$$

where k_1, k_2 depend on the membrane's resistance and conductance. $I(t)$ is the input current to the neuron which depends on the spike events of all those neurons that connect to the modeled neuron. When the membrane potential reaches the threshold voltage, the neuron spikes and its membrane potential is restored to its resting value. Given that these models use continuous time and analog values to represent the membrane potential, spiking neurons are naturally implemented in analog (or mixed-signal) systems.

The realization of SNN is promising yet challenging. To realize a SNN in hardware, the neuron is implemented as a leaky integrator of its input stimulus. Note that a leaky integrator is equivalent to a low-pass filter. Additionally, there must be a spike-emitting mechanism that is triggered when the neuron's state exceeds some threshold value. The spike itself can then be transmitted to other neurons either as a digital event, or as an analog value. For example, in the original vision of neuromorphic engineering [57], SNNs are implemented in electronics using analog, subthreshold CMOS technology. This can be done through electronic circuits that mimic neural dynamics using standard circuit elements operated in the subthreshold regime [56].

*2.4.4. Time models: continuous vs discrete**

Another choice to be made concerns the modeling of time in the RC setup. In ESNs, and machine learning in general, time is usually modeled as discrete steps, with a natural number $n \in \mathbb{N}$. In LSMs, computational neuroscience, and dynamical systems in general, time is commonly modeled as a continuum, with a real number $t \in \mathbb{R}$. Spiking neuron models use continuous time whereas rate-based artificial neurons are typically, though not necessarily, expressed in discrete time. A dynamical system using discrete time is modeled as a difference equation, or recurrence relation, of the form $x_{n+1} = f(x_n)$ whereas a dynamical system in continuous time is modeled as a differential equation of the form $\dot{x} = f(x)$ where $\dot{x}$ represents the instantaneous change of the state variable $x(t)$.

The representative example of a discrete-time reservoir is the ESN which is modeled as

$$\mathbf{x}(n+1) = \mathbf{f}\big(\mathbf{W}\mathbf{x}(n) + \mathbf{W}^{\text{in}}\mathbf{u}(n+1)\big) \tag{7}$$

$$\mathbf{y}(n) = \mathbf{g}\big(\mathbf{W}^{\text{out}}[\mathbf{x}(n); \mathbf{u}(n)]\big) \tag{8}$$

where $\mathbf{x} \in \mathbb{R}^{N_x}$ is the reservoir's response, $\mathbf{u} \in \mathbb{R}^{N_u}$ is the input signal, $\mathbf{W} \in \mathbb{R}^{N_x \times N_x}$ is the reservoir's internal weight matrix, $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_x \times N_u}$ is the input weight matrix, $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_y \times (N_x + N_u)}$ is the output weight matrix, $\mathbf{f} : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x}$ is the nonlinear activation function inside the reservoir, $\mathbf{g} : \mathbb{R}^{N_y} \mapsto \mathbb{R}^{N_y}$ is the activation function for the readout, and $[\mathbf{x}(n);\ \mathbf{u}(n)]$ is the concatenation of the reservoir's response and the input signal at time n.

The representative example of a continuous-time reservoir is the LSM which models SNNs as

$$\mathbf{x}^M(t) = (\mathbf{L}^M\mathbf{u})(t) \tag{9}$$

$$\mathbf{y}(t) = \mathbf{f}^M(\mathbf{x}^M(t)) \tag{10}$$

where $\mathbf{x}^M(t) \in \mathbb{R}^{N_x}$ is the liquid state (analogous to the reservoir's response) at time t, $\mathbf{u}(t) \in \mathbb{R}^{N_u}$ is the input signal at time t, $\mathbf{L}^M : (\mathbb{R}^{\mathbb{R}})^{N_x} \mapsto (\mathbb{R}^{\mathbb{R}})^{N_x}$ is the liquid filter which transforms the input signal $\mathbf{u}(\cdot)$ onto the liquid state $\mathbf{x}^M(t)$, $\mathbf{y}(t) \in \mathbb{R}^{N_y}$ is the output at time t, and $\mathbf{f}^M : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_y}$ is the memoryless readout function. Note that the liquid filter maps the entire input function onto the liquid state and thus contains memory.

How time is modeled generally depends on how the input/output interfaces with the reservoir and what kind of task the reservoir computer aims to solve. If the task's input and output are best modeled as signals in continuous time, e.g. a raw ECG signal or neural spike events, the continuous-time description is most natural. If, on the other hand, the input and output is best modeled as a discrete sequence of inputs, the discrete-time description is most natural. An example would be a sequence of musical notes which is best represented in discrete time because music is divided into discrete time quanta, also called tatum. Another example is processed speech signal in the mel-frequency cepstral coefficient representation, which segments the audio signal into discrete steps, commonly of length 20 ms.

Naturally, any continuous-time dynamical system can be discretized into a discrete-time system. If the reservoir is modeled in continuous time, the theory of LSMs, dynamical systems, and signal processing provides guidance for implementation. If the reservoir is modeled in discrete time, the theory of ESNs and machine learning provides guidance for implementation. In the following, we will describe a reservoir using continuous-time formalisms and, wherever necessary, we will provide a separate treatment for discrete-time and continuous-time models.

*2.4.5. Mode of multiplexing: spatial vs temporal**

A reservoir is a high-dimensional nonlinear dynamical system. We require that the reservoir's state, i.e. its response to an input signal, has many degrees of freedom, in order to ensure the separation property (see section 2.4). As such, the reservoir is usually modeled by a high-dimensional state vector $\mathbf{x}(t) \in \mathbb{R}^{N_x}$ where N_x is the number of components in the state vector. Given an input signal $\mathbf{u}(t) \in \mathbb{R}^{N_u}$, we expect that $N_x \gg N_u$. This is not strictly necessary, for example when the input is high-dimensional and the dimensions are highly interdependent, but it is a useful heuristic in most cases.

When the reservoir is a neural network, we typically treat each neuron i as a single state variable $x_i(t)$ of the reservoir's state vector $\mathbf{x}(t)$. Therefore, to increase the dimensionality of the reservoir's response, we can simply increase the number of neurons in the reservoir, assuming that all neurons are connected. So far, we have assumed that these neurons are spatially separated. In this scenario, we can see that each neuron's response is spatially multiplexed to yield the reservoir's response. Each neuron provides a single degree of freedom to the reservoir's dynamics.

Beside spatial multiplexing, we can also use temporal multiplexing to create a high-dimensional reservoir. This was first proposed for the implementation of a reservoir in photonics [58]. Instead of creating N spatially separated neurons on a physical substrate, they multiplexed the response of the reservoir temporally, thereby creating so-called virtual nodes which play an analogous role to neurons (see figure 6).

We will restrict the input and output to be one-dimensional signals. The input signal, which may be a continuous-time signal $u(t) \in \mathbb{R}$ or a discrete-time signal $u(n) \in \mathbb{R}$, is first processed with a sample-and-hold operation which results in a pre-processed signal $I(t) \in \mathbb{R}$ which is constant during one delay of duration $\tau \in \mathbb{R}$ before it is updated to the next value. $I(t)$ is essentially a discrete-time signal embedded in continuous time. This input signal must be passed into the reservoir. In the standard RC setup, each input is weighted through the random input weight matrix $\mathbf{W}^{\text{in}}$ before being passed into the reservoir. In the time-multiplexed setup, the pre-processed signal $I(t)$ is passed through a random time mask $M \in \mathbb{R}^{N_x}$ (in our case this mask is a

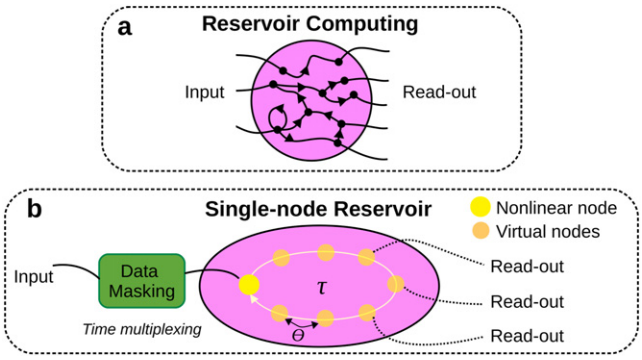

Figure 6. Strategies to improve the capacity of a reservoir. (a) RC in its standard formulation. The more degrees of freedom the reservoir has, the higher its capacity to represent complex functions. (b) Feedback of the reservoir output to its input (directed path) creating a temporally multiplexed or single-node reservoir. Note that the readout should access every virtual node (not depicted in the figure).

vector because the input is one-dimensional), which transforms the constant value during each delay duration τ into N_x different values, each held for a duration of $\theta = \tau/N$. This results in the signal $\mathbf{J}(t) = M \times I(t) \in \mathbb{R}^{N_x}$, which is added to the delayed state of the reservoir $\mathbf{x}(t-\tau) \in \mathbb{R}^{N_x}$ to drive the nonlinear node of the reservoir as

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t-\tau) + \gamma \mathbf{J}(t)) \tag{11}$$

where $\gamma \in \mathbb{R}$ is an adjustable parameter, and $\mathbf{f} : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x}$ is the nonlinear function. Finally, the output of the reservoir $y(t) \in \mathbb{R}$ is given by a linear combination of the values of the N virtual nodes within a single delay duration τ:

$$y(t) = \sum_{i=1}^{N} w_i^{\text{out}} \mathbf{x}\left(t - \frac{\tau}{N}(N-i)\right) \tag{12}$$

where w_i^{out} is the output weight for virtual neuron i. For a more detailed description of time-multiplexed RC, see reference [58].

3. Recipe for physical reservoir computing

In the following, we go step by step through the process of implementing a PRC experiment with the material system/device at hand. The general recipe for implementing a physical reservoir computer is as follows:

(a) Set up the physical reservoir, which is a material device or ensemble of such devices which respond to an external stimulus. Such excitation, or input, can be of any nature, e.g. current, voltage, light, or pressure. As such, the system of choice can be photonic, mechanical, electronic, chemical, etc. The response of the system must be nonlinear with respect to the input, and high-dimensional, meaning that the reservoir's response to external stimuli must have many degrees of freedom. Note that, depending on its complexity, a device can correspond either to a single element in the reservoir, i.e. a neuron, or to the reservoir as a whole. A system with these properties is a reservoir. The system should offer some control parameters that may be tuned to change the response of the reservoir. In this way, it can be set up in a way that shows the echo-state property, as well as other measures relevant for the task at hand. The reservoir is often tuned again together with the learning of the readout.

(b) Set up a method to inject external inputs into the reservoir. When injecting signals, the reservoir should respond by changing its internal state.

(c) Set up a readout mechanism for the reservoir's internal state. In the following, this mechanism will be called readout layer or function. The readout should yield many channels that are not highly correlated, so as to avoid redundancy. If the reservoir consists of many interconnected elements, then the response of every element in the reservoir should be read out. During training, these reservoir states need to be collected and stored, thus some mechanism for storing the states is required for the training period.

(d) Train the readout mechanism, typically through linear regression as described in section 2.1. At this point, the reservoir itself may also be tuned to increase performance of the overall system (cf section 3.5).

In the present section, we will describe each step of this recipe in detail and we will employ an example system from our labs to illustrate this process. The data for this example PRC experiment are available to the

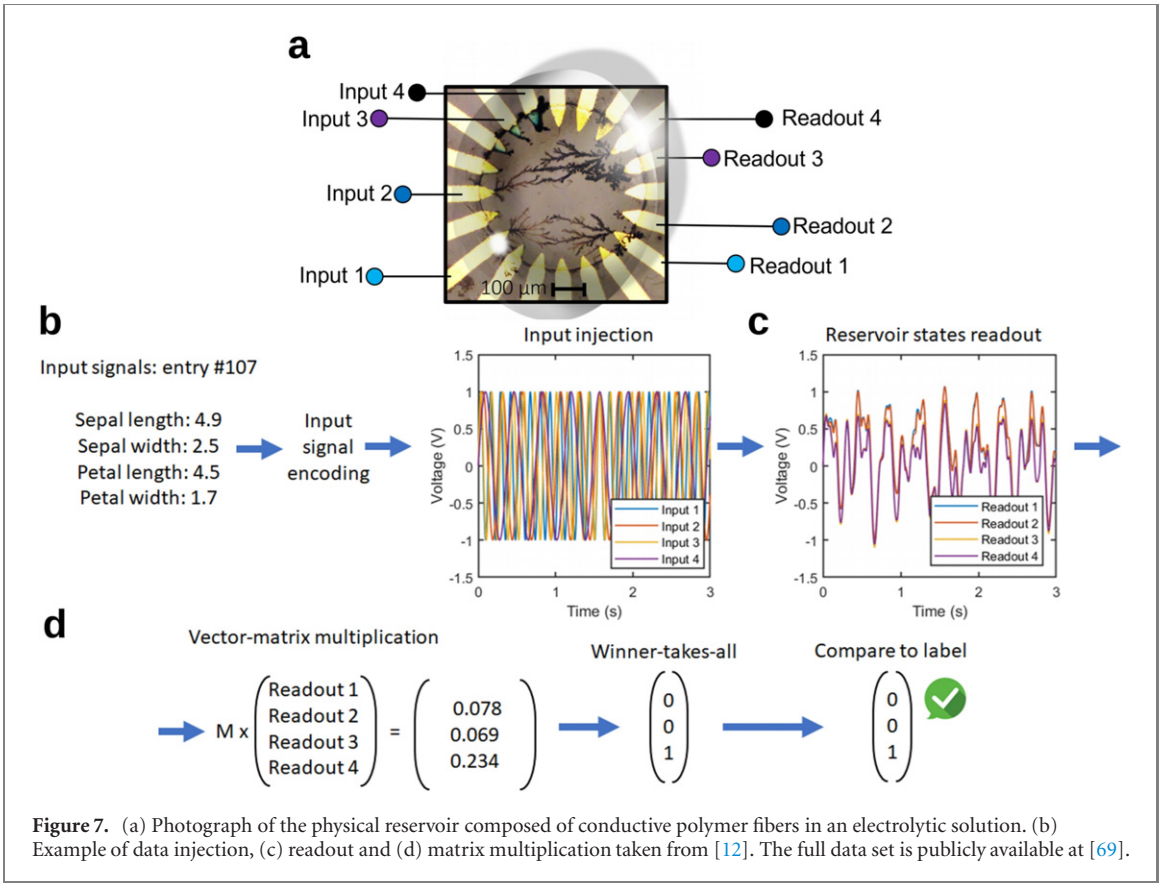

Figure 7. (a) Photograph of the physical reservoir composed of conductive polymer fibers in an electrolytic solution. (b) Example of data injection, (c) readout and (d) matrix multiplication taken from [12]. The full data set is publicly available at [69].

reader, allowing him/her to retrace every step of the implementation. Afterwards, section 3.5 will offer methods for tuning the reservoir that go beyond the presented experimental setup. This can be used as reference, when encountering physical systems that deviate from the described setup.

3.1. Setting up the reservoir

The first item on the list is the reservoir, the heart of the computational machine. In principle, any system which can project the input signal nonlinearly into higher-dimensional space may be used as reservoir. Nonlinearity is important because if the projection into the reservoir is linear, then the projection into high-dimensional space will not enable the separation of signals that were inseparable in the lower-dimensional input space, cf section 2.4. A simple system like a resistor, where the input signal is voltage and the output signal is current, would be useless because of the linearity between the two. A transistor on the other hand, represents a suitable system since the current flowing through the channel can be nonlinear with respect to the drain-source or gate-source voltage, if operated in the right regime. Numerous physical nonlinear effects have been exploited to produce reservoirs. From the nonlinear I–V curves of memristors [59, 60], transistors [61], and laser cavities [62], or from spintronic effects [63–66], or the coupling of mechanical waves [67].

To better illustrate the practical implementation of a RC experiment, throughout the following sections an example taken from reference [12] is used for illustration. In this setup, a network made of conductive polymeric fibers (see figure 7) connects a set of input electrodes on the left, to a set of output electrodes on the right. The resistance of each fiber, when immersed in an electrolytic solution, features a complex, time-dependent and nonlinear dependence on the voltages of all the other fibers (more details about the physics of this process in reference [68]). Most important for the function of this reservoir is that the polymeric fibers (each acting as an organic electrochemical transistor) gate each other through the electrolyte, which ultimately creates the recurrent and the ESP of the network. Therefore, the current flowing through each fiber allows a nonlinear transformation of the input signals. Without the presence of ions or even in a dry environment, each fiber would operate as a resistor and the reservoir function is lost.

3.2. Input injection

For computation to occur, the system must be excited with input signals. For the example here, such excitations can be mechanical (sound waves), optical (absorption of the polymeric fibers), chemical (injection

of anions or cations), electronic (applying a voltage to the input), etc. In principle, they can be partly of one type, and partly of another type. This is a strong point in favor of physical neural networks: their analog and diverse interaction with the surrounding world can be exploited at once. Two examples may shine light on this concept: photonic neural networks have the potential to solve low-latency computation operations within a digital hardware. As such, the inputs will be a number of electronic signals transformed into optical signals, processed by the photonic reservoirs, and converted again in digital, electronic values. In this framework, the system is 'expected' to interact merely with optical signals. A different scenario, which is similar to our example system, can come from bioelectronic devices, where chemical signals could be translated into electronic ones. This way, the reservoir is in direct communication with the environment, opening up energy-efficient (analog) ways of sensing. Similarly, many changes in the environment (temperature, pressure, pH, etc), usually undesirable in traditional computational systems, can be harnessed for computation.

To demonstrate a recognition task with the polymeric fiber reservoir, we employ the so-called iris data set [69], where each entry of the data set has five attributes (sepal length, sepal width, petal length, petal width, species) and the PRC experiment is used to classify the species depending on the first four attributes. To carry out this PRC experiment, input signals are injected into the input layer using a multichannel arbitrary function generator at 5000 samples per seconds for 3 s. To demonstrate a recognition task with a physical reservoir using standardized data sets, preprocessing of the data is often needed in order to apply the signals to the reservoir. For the example shown in figure 7, the data set entries are encoded into the frequency of the sinusoidal input signal (linear transformation). In principle, the information could also be encoded in the voltage amplitude of the input signal, however, the values are spanning over a wide range which the reservoir cannot tolerate (reducing the range would be possible but then the nonlinearity of the system would be reduced). Overall, preprocessing of data is often needed and the ideal transformation needs to be found empirically for a given physical reservoir. It is important to verify that the setup used to inject input signals (and to readout) does not introduce unwanted distorsions of the signals. This can happen if the time step chosen or the signal frequency approach the limits of the setup itself. At the same time, a time step much smaller than the frequency of the signals must be chosen to have smooth signals and to ensure the catching of small nonlinear deformations.

Advanced data processing. When working with numerical data, it is important to pay attention to the scaling of the data. All variables in the input should be at approximately the same scale. This ensures that all input channels have the same ability to create reverberations in the reservoir. In contrast, if some input channel always had very low values compared to others, it might fail to sufficiently stimulate the reservoir, and information from this channel will be lost. The output scaling should also be consist but this is less important, as long as the readout map can produce small values without zeroing out.

In contrast, categorical data requires some more processing. This is the case when the learning task is a classification, e.g. distinguishing which of the ten digits was uttered in a speech signal. One may model the output of such a digit classification system with a single number that represents the best guess of the spoken digit. But this is not the best way to encode the output. Rather than outputting the digit, it is better to output a probability distribution for the output by having one output signal per digit, each representing the probability for the signal belonging to this digit. If the readout map is a weight matrix, as in the ESN formulation, this is analogous to training one readout per digit, each one predicting the probability that the input signal is classified as this digit. Such an encoding of categorical data is called a one-hot encoding.

3.3. Readout & output

Usually, depictions of networks for RC and ESNs shows an input layer and the output layer, with trainable connections from the network to the output. In this paper, we differentiate to give a better picture of a hands-on experiment: we first have a 'readout' layer i.e., the measured data, the harvested states. Then, from the readout state one goes to the output layer with trainable connections.

The network/reservoir reacts to the input excitation evolving towards a new state. Such a state, in a well-designed reservoir, is not merely the consequence of the just-injected signal, but it is also influenced by the past, still reverberating in the network in the form of an echo, as explained above. The readout step consists in harvesting the network states in a matrix $S \in \mathbb{R}^{T\times N_x}$ where N_x is the number of output channels attached to the reservoir, and T is the number of time steps for which the reservoir was run. If a readout of the complete reservoir state is possible, then $N_x = N$ where N is the degrees of freedom in the reservoir. As for the conductive fibers, as mentioned, the resistance of each connection changes depending on the applied voltages. The current flowing is recorded at the readout layer: specifically, four readouts are carried out continuously using a multichannel analog-to-digital converter at 5000 samples per second for 3 s. Note that the sampling rate and the number of time steps should be the same for the readout and the input to allow easier data processing

during the linear regression. As specified in the previous subsection, it is fundamental to ensure the quality of the readout setup in those specific conditions to avoid the recording of nonlinear artifacts. Every nonlinear contribution must come from the network. It is important that the information within the N_x channel is not redundant, a problem that can quickly emerge in miniaturized systems. For example, in the electrochemical networks used by Cucchi *et al* [12], there may be one input directly connected to two adjacent output channels, effectively providing a short-circuit. As a result, those two channels can have the same readout, hence lowering the effective dimensionality of the readout. Similarly, in photonic systems, two pixels very close may detect the same signal. Such redundant information is not directly harmful, but it is inefficient and may be costly. It is also worth noting that some readout channels may appear very similar in the time domain, however, essential differences might be visible if observed in the frequency domain. In this particular example, one could avoid similar readouts by exploiting the transistor nature of the device: inputs are used as gates, and different gate voltages can be set to change the behavior of the outputs in a distinguishable way. Another way could be changing the geometry of a reservoir, apply the same inputs but having different distances between the gates and the output layers. Again, such states can be harvested under different forms and physical units. If one wishes to integrate the reservoir in a hierarchical multi-reservoir architecture, it is necessary for the input and output to have the same physical unit, e.g., both voltages or both optical power. A similar argument applies when using the time-multiplexed reservoir with delay lines (cf section 2.4.5). In order to exploit the kernel method (i.e. the projection of the input signal onto a higher dimensional space), the number of outputs should be larger than the number of inputs i.e. $N_x > N_u$ (see figure 3). This condition is not fulfilled in the example in figure 7, where the low complexity of the classification task (iris dataset) allowed for good results without increasing the number of readout channels.

3.4. Training and evaluation

Train-validation split. Before starting the training procedure, the dataset is split into a training set and a validation set. The training set is used to train the system, and the validation set to evaluate the trained system's performance. If the entire dataset is used for training, then there would be no way to evaluate the system's ability to generalize from the training data to new, unseen data. Instead, only the system's performance on the already seen training data would be evaluated, which would serve as a good measure for memorization, but not for learning.

The standard split is to use 80% of the data for training and 20% of the data for validation. This splitting should also be done mindfully. For example, if the dataset contains stock prices and the learning task is to predict stock prices into the future, then the date range of the training data should precede the date range of the validation data.

Training. The training proceeds in two phases. First, the input is injected and the reservoir states are harvested. Then, the readout map is computed, usually using linear regression.

To harvest the reservoir states, the input signal from the training data $\mathbf{u}^{\text{train}}(t)$ (a subset of $\mathbf{u}(t)$, see above) is injected into the reservoir, for a total of N_{train} time steps. Then, the reservoir states are read and saved at every time step, yielding a time series $\mathbf{x}^{\text{train}}(t) \in \mathbb{R}^{N_x}$, also of length N_{train}. Each component of this time series is a nonlinear transformation of the input signal.

In order to compute the readout map, the harvested states are arranged into the state collection matrix $S = [\mathbf{x}^{\text{train}}(1), \ldots, \mathbf{x}^{\text{train}}(N_{\text{train}})] \in \mathbb{R}^{N_x \times N_{\text{train}}}$. The desired output states are collected from the training data $\mathbf{y}^{\text{train}}(n)$ into an output collection matrix $D = [\mathbf{y}^{\text{train}}(1), \ldots, \mathbf{y}^{\text{train}}(N_{\text{train}})] \in \mathbb{R}^{N_y \times N_{\text{train}}}$.

In the following step, the optimal output weights are computed with linear regression. Linear regression attempts to fit a linear function from the reservoir states to the desired output:

$$y_j = \sum_i w_{ij}^{\text{out}} x_i \tag{13}$$

$$\mathbf{y} = W^{\text{out}} \mathbf{x} \tag{14}$$

where, by convention, the last reservoir state is fixed to unity activation $x_{N_x} = 1$. This acts as a scalar in the linear regression.

The linear regression is computed through a least squares approximation (mean square error, MSE), which minimizes the sum of the squared error terms:

$$W^{\text{out}} = \arg\min_{W^{\text{out}'}} \sum_{i=1}^{N} (W^{\text{out}'} x_i - y_i)^2 \tag{15}$$

The solution to this minimization problem is given by

$$W^{\text{out}} = (S^{\dagger} D)^{\text{T}} \tag{16}$$

where $S^{\dagger}$ is the pseudo-inverse of the state collection matrix, and $^{\mathrm{T}}$ denotes the matrix transpose.

Instead, also regularization with ridge regression can be employed which minimizes the estimated risk with an additive penalty term, i.e. $W^{\mathrm{out}} = \arg\min_w \sum_{i=1}^{N} (wx_i - y_i)^2 + \alpha^2 \|w\|^2$. This leads the learning algorithm to prefer solutions for the output weights with smaller norms, which helps to prevent overfitting. The solution of ridge regression is given by

$$W^{\mathrm{out}} = (S^{\mathrm{T}}S + \alpha^2 I)^{-1} S^{\mathrm{T}} D \tag{17}$$

where I is the identity matrix and α^2 is the non-negative regularization coefficient. Note that $S^{\mathrm{T}}S$ is proportional to the correlation matrix of the state collection matrix and $S^{\mathrm{T}}D$ is proportional to the cross-correlation matrix of the states with the desired outputs.

The regularization coefficient α^2 must be chosen manually. If $\alpha = 0$, the ridge regression formula simplifies to standard linear regression. A non-zero value for α^2 warrants numerical stability. Because the regularization parameter controls the model capacity of the linear regression, it controls how much the readout function overfits (or underfits) the training data. This affects the generalization ability of the readout.

For this reason, the regularization parameter is commonly found through a CV procedure (also called out-of-sample testing) which optimizes the generalization ability of a learning algorithm. For details on CV, see chapter 5 in reference [38].

Linear regression on hardware. As the linear regression is a mathematical problem that can be conveniently and efficiently solved on digital computers, the regression is usually not carried out in hardware in a PRC experiment but rather in software. However, approaches to exploit Ohm's law to solve matrix operations have been successfully demonstrated (e.g. [70]) and may be useful in obtaining the full neural network on hardware.

Evaluation. Once the output has been trained, the system's performance can be evaluated. To do this, the input signals from the validation data $u_{\mathrm{val}}(n)$ are injected into the physical reservoir, and then read out the corresponding output $\hat{y}_{\mathrm{val}}(n)$ from the physical system using the newly trained readout map. Then, this output is compared to the desired output from the validation data $y_{\mathrm{val}}(n)$. For regression tasks, a standard measure is the above-mentioned MSE on the validation data.

For classification tasks, it is more common to report the accuracy, i.e. the ratio of classifications that were done correctly. If the output is one-hot encoded (see above), then the output signal does not directly give the chosen class, but instead only the estimated 'probability'[8] for each class. To convert this into a classification, the maximum value is simply set to unity, and all other values to zero ('winner-takes-all' approach). For example, if the output for a four-class classification problem is $[0.9, 0.7, 0.1, 0.4]$, then it is processes into $[1, 0, 0, 0]$, which allows for an easy calculation of the classification accuracy.

In our example (see figure 7), the mean squared error (MSE) has been used to minimize the loss function between the desired output and the actual output. However, in some tasks it may be desirable to use different loss functions. In classification tasks, when the output is a probability distribution over all classes that are considered, one commonly uses the so-called cross-entropy loss [71].

3.5. Optimization

As outlined before, one of the advantages of RC lies in the ease with which a reservoir system can be trained, by modifying only the readout map and keeping the main part of the physical system, the reservoir, fixed. The readout maps is most commonly a weighted combination of the high-dimensional reservoir state and it is trained using (regularized) linear regression.

Although it is possible to keep the reservoir fixed, it is often beneficial to tune the reservoir to make the reservoir computer more effective. In a material system, the reservoir is usually not a black box that cannot be modified in any way. Rather, one can grow the reservoir in different ways, or tune the reservoir's dynamics in different ways. These are additional degrees of freedom that can be exploited to improve the overall system's performance.

Allowing the reservoir to be optimized leads us to a nested optimization procedure, see figure 8, in which the inner optimization loop is the readout training and the outer optimization loop is the reservoir tuning. While training the readout map is usually called 'learning', the tuning of the reservoir can be called 'meta-learning' or 'learning to learn' [72] because it subsequently enables more effective learning of the readout map.

In software simulations of reservoirs, the outer loop optimization can often be automated, but in physical reservoirs the outer loop optimization usually modifies the physical reservoir and is therefore done manually.

[8] The output values are not actually probabilities because they may not sum to unity.

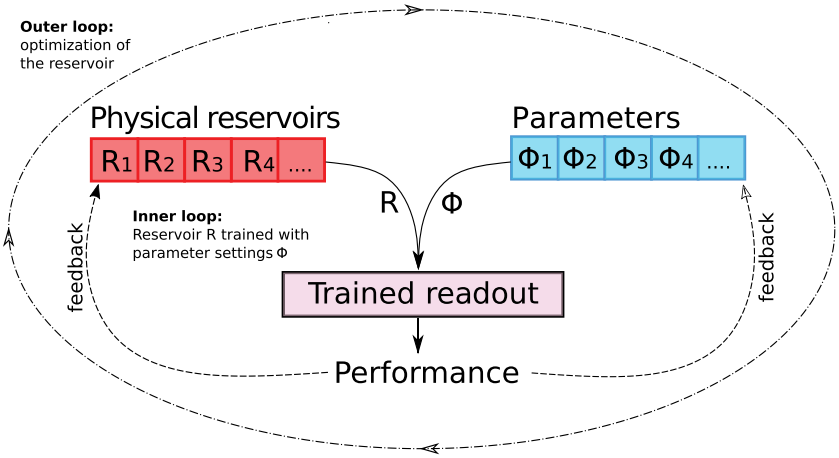

Figure 8. Nested optimization of a physical reservoir computer. In the inner loop only the readout map is trained, typically using linear regression. In the outer loop, a suitable reservoir is chosen together with other parameters (e.g. the regularization parameter α^2), based on the performance of the reservoir with a trained readout map. Figure inspired from references [72, 73].

The only parameters that are optimized in the inner loop concern the readout map. In the outer loop of this optimization, other parameters concerning the reservoir itself are optimized.

The previous section described the training procedure for the reservoir's readout map which makes up the core of the learning procedure in PRC, shown as the inner optimization loop in figure 8. The present section will outline some common practices for tuning the reservoir itself. A single reservoir may be well-suited to solve a wide variety of tasks. However, it will be observed that tuning the reservoir for some task (or set of tasks, in a multi-task setting), can dramatically increase the system's performance. A full understanding of the computational expressiveness of a reservoir is still missing [74], therefore we will only outline some 'tricks of the trade' that can be found across the literature of (physical) RC [1, 14, 75, 76].

We proceed by outlining some of the parameters that may be tuned in this outer optimization loop, investigating the effect of each parameter independently and giving some general advice for tuning each one. We will then outline some of the more general trade-offs that must be made in designing a reservoir. For this, we introduce some generic attributes that have been proposed to assess a reservoir's performance, and borrow some attributes from the machine learning literature. We point to section 2.4 for a more detailed treatment of these generic attributes.

3.5.1. General setup

The reservoir is commonly tuned for a task-specific input while the output need not be specific to a single task. In a multi-task setup, we want to optimize our reservoir for multiple functions, each of which is approximated using the same reservoir but with different readout maps. In this case, we optimize the reservoir jointly for the input of all such functions, and then train different readout maps for each task. For example, the reservoir may be tuned on speech signal input and can then be used for speaker recognition, phoneme classification or sentiment analysis. It may be noted that some sets of tasks are better suited for implementation in a single reservoir than others. In the above example, phoneme classification requires memory only on short timescales of <1 s whereas language translation would require a longer memory timespan of multiple seconds.

Generally, the tuning of the physical reservoir can be done either in the material directly, or in simulation. Tuning the reservoir in simulation allows the use of automated optimization procedures in order to find a well-suited reservoir for the task(s) at hand. However, in order to tune the reservoir in simulation, an accurate model of the physical dynamics of this reservoir is required. This is not always easily obtained, therefore we will focus on tuning the reservoir directly in the material.

If the tuning of the reservoir can be automated with a computer in the loop, one may use well-established search procedures. In such an automated setting, one commonly defines the search space of all possible parameter settings, and then chooses some search procedure in order to find a good reservoir within this search space. Such search procedures can be found in the literature on meta-learning [72], or neural architecture search [77]. Popular techniques that are easy to implement include grid search, random search, evolutionary search, among many others. In grid search, the search space is divided into a grid and one configuration in each cell of the grid is tested which ensures that the entire search space is explored. In random search, the search space is explored randomly where each configuration is sampled independently of previous runs. In evolutionary search, a population of configurations is explored and the best

configurations are combined to explore similar configurations. Evolutionary search is an adaptive search algorithm because it explores configurations that are similar to the best-performing configurations found thus far.

If the tuning of the reservoir cannot be automated, it must be done manually in the material. The best way to do this is to start with some initial configuration and change only one parameter at a time, to see the effect of each change [1]. It is often helpful to take a look at the internal dynamics of reservoir states to understand what is going on in the reservoir. For example, if different input signals always produce the same constant output signal, it may be that the responses of the reservoir's elements are saturating. The cause may either be that the input is too large or that the reservoir dynamics are over-stimulated. By taking a look into the response of the reservoirs' elements for differently scaled input signals, the two cases are easily distinguished.

3.5.2. What makes a good reservoir?

Evaluating a reservoir with the performance on some learning task is straightforward, but we may want to assess a reservoir's quality before having decided on a particular learning task. Several such task-agnostic evaluation metrics have been proposed for RC, for example the memory capacity of a reservoir—both the linear memory capacity [51] and the nonlinear information processing capacity [76].

First of all, we would like to ensure that the reservoir has the ESP so that its initial condition washes out and it has a fading memory. This ensures that the reservoir is stable when it is driven by an input signal, and that the reservoir will map the same input to the same outputs, regardless of the initial conditions. We can tune the length of this fading memory through the memory capacity, or information processing capacity [51, 76, 78] (see below). We also want to ensure that the reservoir has the separation property so that the high-dimensional reservoir dynamics allow to differentiate different input signals. This is also referred to as the kernel quality. The kernel quality describes how well the reservoir separates the input signals, i.e. whether sufficiently different input signals become linearly separable in the reservoir's state space. A linear reservoir does not provide better separation of input signals. Reservoirs with higher nonlinearity or larger overall size have higher kernel quality. As shown by Petrauskas *et al* [68] for an electrochemical reservoir similar to our example system (figure 7), the nonlinearity of such a system can be quantified by the total harmonic distortion of the input signals and its value can be tuned by the biasing conditions (operating the transistors e.g., in the linear or subthreshold regime).

A general measure of the power of a machine learning model is its capacity (or complexity, or expressivity). The higher a reservoir's capacity, the larger the set of functions that it can approximate. In general, the model capacity increases with the size and the amount of nonlinearity of a network, and also grows with larger memory capacity.

Lastly, we may also want to evaluate our reservoir on its robustness to noise, or other measures.

Trade-offs and difficulties. As already outlined above, tuning a reservoir involves a number of trade-offs such as between the memory capacity and kernel quality, which have opposing requirements on the nonlinearity of the reservoir. Furthermore, the capacity of a model tends to conflict with the model's robustness to noise.

Although the size of a reservoir generally improves its performance, this improvement does not always continue unboundedly. Instead, the reservoir may saturate in performance at some size. Many tricks and extensions have been devised to further increase the performance of a reservoir: by using multiple reservoirs and averaging their outputs, by using multiple reservoirs in some particular architecture, by using lesser-connected or modular reservoirs, or by using heterogeneous units in the reservoir (see section 2.4).

Reservoir dimensionality. The first, and most obvious, adjustment of a physical reservoir is its size. In a network-based reservoir, this is the number of neurons and in a delay-based reservoir it is the number of virtual neurons. The size of a reservoir specifies how many degrees of freedom the reservoir's response has. The larger the reservoir, the higher its capacity (if the reservoir states are independent), as well as its memory capacity and separation ability. Therefore the size is usually chosen to be as high as possible—at least as many degrees of freedom as the number of independent real values that the reservoir must remember at any time for the learning task [1]. Of course, there are other constraints on the size because a larger network is typically more expensive, so a good trade-off needs to be found [79].

Fading memory. Some material systems feature intrinsic recurrence. These recurrent connections allow responses to previous inputs to 'reverberate' in the reservoir. Once input signals are fed into the reservoir, the signal keeps propagating within the networks even after the input excitation has been interrupted. If the signal is internally amplified, the reservoir can remain active indefinitely. Without amplification, the signal is damped and it will die off over time with a time-constant τ that depends on the system.

Fading memory is given in every system that has at least two vastly different critical time constants, e.g., thermal and electronic conduction in thermistors, ionic and electronic conduction in organic electrochemical transistors, etc. An example of a physical reservoir composes of organic electrochemical transistors is shown in figure 7(a), where semiconducting polymer fibers (in an electrolytic environment) connect input and output of the reservoir. In this regard, every physical system has a typical relaxation time constant i.e., the time needed for the system to evolve towards equilibrium, when the input is taken away. Therefore, the response of a system to an external stimulus will strongly depend on the initial condition. If the system is not at equilibrium because of a previously injected input from which it is still relaxing, the response will be different. If this reverberation is long-lasting (much longer than the frequency f at which inputs are fed) within the network, it is called an 'echo'. In order to have powerful reservoirs long echos are desired and the system must be operated at speeds faster than its relaxation time. i.e. $\tau \gg \frac{1}{f}$. For example, a transistor must be operated at frequencies faster than the charging time of its gate, in order to obtain a comparably long echo.

Mode of multiplexing. From a practical standpoint, one uses a specific type of delay line based on the combination of input–output used in the system. In a photonic system, the delay is quite easily realized by employing an optical fiber that connects output and input layer. The length and material of the cable can be adjusted to modulate the delay. An optical amplifier takes care of the losses. For a electronic-input electronic-output reservoir, a delay line cannot consist solely of a conductive cable because it would allow the signal to travel both ways. Rather, it can be made with a delay element, either digital or analog, and an amplifier connected to the readout layer. The single-node RC architecture results in the superposition of nonlinear transformations, which can in turn result in chaos. Computing in the chaotic regime is not effective as, by definition of chaos, tiny differences in the input signals will be projected as two very different data points on the readout layer. However, chaotic behavior proves sufficient complexity of the system for RC as in this case, e.g. a sinusoidal input (δ-function in frequency space) would be transformed into noise (containing an infinite number of frequency components), which can be seen as the ultimate nonlinear transformation.

Memory capacity. Some tasks may require longer memory while others may require little to no memory at all. Therefore, we ought to match the memory requirements of the task to the reservoir's memory capacity. Or reversely, we want to choose a suitable task given a physical reservoir's (fixed) memory capacity. It is important to note that a standard reservoir does not have any persistent memory, but it is possible to create attractor states in the reservoir that correspond to persistent memory (see section 2.4 for a discussion). A reservoir's memory capacity generally increases with size. Dambre *et al* [76] have shown that the information processing capacity of reservoirs with fading memory that have equal numbers of linearly independent internal variables are equal. Although some such reservoirs may carry out more interesting computations than others, the 'amount of computation', as defined by the information processing capacity, is equivalent. The memory capacity is also connected to the stability regime of the reservoir.

Dynamical regime. It is important to note that a reservoir is an input-driven dynamical system and as such, its stability depends not only on its dynamics in the absence of input but also on the input that drives the reservoir, and its dynamics in this input-driven mode. A stable reservoir will quickly converge to zero without an input, and it therefore has little memory. A 'less stable' reservoir will take longer to converge to zero when the input is taken away, and therefore has longer memory. An unstable reservoir will not have the ESP, and may not reach the zero state at all. This leads to longer memory, though such unstable reservoir dynamics are not easily 'tamed'.

For tasks that require long memory, it is often said that the reservoir should operate at the 'edge of chaos', i.e. close to criticality, so that it has large memory capacity without becoming too unstable. It is often claimed that a reservoir generally performs best at this 'edge of chaos'. This is not generally correct—it depends on the task. For some tasks, a reservoir far on the stable side performs better than one close to this edge [80]. One way to tune the criticality of a physical reservoir is demonstrated in reference [68] where an electrochemical reservoir with a delay line has been used to implement a single-node reservoir (cf figure 6(b)). In this case, the amplification of the delay line can be employed to tune the criticality of the reservoir and ultimately reach chaos.

Moreover, it is also possible for a reservoir on the other side of the 'edge of chaos' to perform well. A reservoir that is chaotic in the absence of a driving input signal may perform better than a non-chaotic reservoir when driven by a sufficiently strong input signal and/or by a feedback signal [81, 82]. These signals then suppress the chaoticity of the reservoir dynamics and restore the ESP. The chaotic dynamics of a RNN can also be suppressed

through recurrent plasticity [83] which the authors have used as a framework for robust neural computation with dynamical attractors.

The dynamics of the internal reservoir may also be tuned in a physical system. Indeed, the stability regime of an ESN is commonly tuned through the so-called *spectral radius*. The spectral radius ρ_W is the largest eigenvalue of the internal weight matrix W, and therefore controls the stability of the autonomous reservoir (without input). A spectral radius less than unity ensures that the system will converge to the zero state in the absence of any input. If the spectral radius is larger than unity, then it is possible that the network will not converge to the zero state. A reasonable starting value for the spectral radius is between 1.2 and 1.5, though this depends on the desired stability regime as well as on the input signal. The spectral radius can be tuned rather easily by simply scaling the weight matrix W: first divide the matrix by its current spectral radius, then multiply it by the desired spectral radius.

Nonlinearity. In some physical reservoirs, it may be possible to change the kind of nonlinear activation function that is used in the reservoir. In software simulations of ANNs, the tanh function is the most-used nonlinear activation function. However, physical reservoirs typically have their own nonlinearity that may not correspond perfectly to some analytical function like the hyperbolic tangent. In any case, it may be possible to tune the amount of nonlinearity in a physical system.

As discussed before, the 'amount' of nonlinearity affects the memory capacity and kernel quality of the reservoir. Generally, the linear memory capacity [51] decreases for increasingly nonlinear activation functions. However, this does not directly imply that nonlinear reservoirs have bad memory capacity—information is not lost through the nonlinearity, it is merely transformed in various nonlinear ways that cannot easily be read out again with a linear operation [76]. If the goal of the reservoir is to implement a nonlinear function of the input signal, then this is even desirable and the linear memory capacity will not be a good measure of performance. On the other hand, if the computational task for the reservoir requires storing information about the input signal without nonlinearly transforming it, a linear reservoir will be better suited.

In addition to simply tuning the activation function itself, it is also possible to tune the reservoir's nonlinearity by adjusting the scaling of the input signal and the weights. Often, the nonlinear activation function saturates and therefore becomes more nonlinear for higher values, while being almost linear for small values. Therefore, by driving the reservoir with larger input signals, or by increasing the connection weights, the reservoir's response will become more nonlinear.

It is also possible to work with a fully linear reservoir. While the kernel quality (and therefore separation ability) of such a reservoir will be very low, it can be used for its high memory capacity. An example of this are linear photonic reservoirs where the nonlinearity is implemented in the readout layer [84].

Input scaling. As already mentioned, the stability regime of the reservoir depends not only on the reservoir's internal dynamics but also on the input that drives the reservoir. Specifically, it is important to match the input scaling to the reservoir's internal dynamics. For example, while a reservoir with large spectral radius may be unstable autonomously (i.e. without input), driving the reservoir with an input signal will effectively stabilize the system dynamics because of the saturating nature of the hyperbolic tangent. It is usually best to ensure the input to be bounded to some range, e.g. between 0 and 1 by preprocessing the input signal with a hyperbolic tangent function. Naturally, the input scaling can also be adjusted by scaling the input weight matrix.

Others. More opportunities for tuning the reservoir exist, for example the scaling of neuron biases (which can also tune the nonlinearity of the reservoir, see above), the sparsity of the connection matrices (usually the input matrix is full, the internal matrix is sparse, though the sparsity does not usually have a very large effect of the performance [1]), or the distribution with which the weight matrices are initialized (usually a normal distribution with zero mean and unit variance is used, sometimes also a uniform distribution between -1 and 1). When using a leaky reservoir, where the reservoir states are low-pass filtered, there exists another parameter σ which controls the leaking at each time step, i.e. the neuron's internal time constant. This allows for tuning of the multiple time scales that are involved in the reservoir dynamics. When no prior knowledge about good parameter ranges exists, it is best to set the values to those that can be found in the literature, or to optimize them through a search procedure in order to find good values.

3.5.3. Tricks of the trade

We conclude by outlining some other common practices when working with (physical) reservoir systems.

Washout period. It is possible that the effects from (random) initial conditions of the reservoir affect the initial time steps of the reservoir's response, leading to a performance decline. To remedy this, one often discards initial time steps for a certain washout period T_{washout}. Whether or not this will be helpful can be tested by passing the same input time series at two different initial conditions of the reservoir and comparing their performance. If the performance difference is significant, a washout period may be helpful.

Time averaging for classifications. In temporal classification tasks, the classification labels can be sparse in that the label is not expected at every single timestep during the sequence. In some cases, the performance of the reservoir can be improved by computing the time average of the activations for the classification.

Regression weighting. In some tasks, the errors at different timesteps are not all equally significant. In the extreme case, only outputs at specific timesteps are significant and all others are ignored. In general, it is possible, and often helpful, to weigh the error at some timesteps higher than at others. This can be done directly in the linear regression, by simply using a multiplier for different time steps.

Online learning. In the standard RC setup, we train the readout function only once, and then use the reservoir for inference with a 'frozen' readout function. However, for some tasks, it is required to train and adapt the readout function continuously, or to learn online. This tends to decrease performance, compared to a full retraining on all previously seen data, due to challenges connected to catastrophic forgetting. Therefore, online learning should only be used when necessary and suitable.

3.5.4. Extensions

In addition to the three general flavors of physical reservoirs, the standard RC formulation has been extended and modified in various ways to better exploit the physical systems at hand. Such extensions may be functionally motivated, in order to implement some particular computation, or they may be physically motivated, in order to fit the constraints of the material.

While the readout function of the reservoir is most commonly a single fully-connected layer, or a matrix multiplication, it is also possible to use more sophisticated readout functions, like support vector machines, or Gaussian processes.

Although random connections are common in RC, they are not strictly necessary. It is possible that the physical setup does not allow random all-to-all connections but rather constrains connections to nearest neighbors only. This has been shown to work well [85]. Previous work has also used brain-inspired connectivity for models of the neocortex [86]. In general, it is possible to have a reservoir with a specific, non-random connectivity.

In the standard RC framework the reservoir itself is kept fixed, and only the readout map is trained—either online or offline. But this is not strictly necessary. It may be beneficial to also optimize the reservoir itself by tuning some of its parameters, as will be explained in section 3.5. Furthermore, it is sometimes difficult to keep the reservoir's dynamics unchanged over time. RC has been applied to situations in which the physical system underlying the reservoir may be evolving [87] or adapting to its environment or learning task [88]. Local learning rules have also been used to train the internal connections in the reservoir [30, 31, 89].

In order to add some structure to a physical reservoir, and to increase its computational capacity, several architecture have been proposed to combine multiple reservoirs together. One popular approach is deep RC, in which several reservoirs are arranged in layers [33]. Another common line of research in machine learning uses weak machine learning models (here: simple reservoir systems) in an ensemble to create a stronger machine learning model [32], through bagging or boosting techniques (see chapter 8.2 in reference [38]).

So far the reservoirs mentioned in this paper have always been driven by an input signal, but it is also possible to use a reservoir as an autonomous dynamical system with no input. This can be done either by removing the input signal entirely (effectively setting all input weights to zero), or in a closed-loop setting by feeding back the reservoir's output signal back as an input signal.

The closed-loop setting has been used to learn a dynamical systems and reconstruct the long-term dynamics of its attractor [90]. This direction has received much attention recently, and the setting has been extended to also learn parameterized dynamical systems through the use of control parameters that are passed as input signals to the reservoir [91].

Autonomous reservoirs that do not receive any input are also used with *conceptors* [92]. Conceptors are generic neuro-computational mechanisms that can be used in various ways, including for the autonomous generation of patterns from a reservoir. This setup has been demonstrated for human motion pattern generation with a reservoir of 1000 artificial neurons [93]. Instead of using the input layer to feed the output back into the reservoir, the input connections are removed completely. The reservoir is then *loaded* by modifying its internal connection weights. The conceptor matrix C_i is computed for each pattern p_i that is to be generated, and the reservoir is started autonomously with the conceptor matrix in the state update function. Intuitively, the reservoir's states are filtered through the conceptor matrix and thereby select and stabilize the neural dynamics that correspond to the desired pattern.

3.6. Summary

In the following, we provide a condensed summary of how to implement and train a physical reservoir computer. We assume that a dataset of input signals and corresponding desired output signals is already given, and stored on a digital computer which can be connected to the physical reservoir.

(a) **Set up reservoir**. First identify the suitable mode of multiplexing for the physical system (spatial or temporal, see section 2.4.5). Then, identify all parameters that can be tuned in the physical reservoir, for example:

1. the reservoir's complexity, typically tuned by its size or number of interconnected elements,
2. the reservoir's dynamical regime, typically tuned by the strength of internal dynamics within the reservoir as well as the scaling of the input signal,
3. the nonlinear activation function that is used, typically tuned by the physical effect that is used to implement this nonlinearity,
4. the timescale of the reservoir can sometimes be tuned (to a limited extent) and should be made to match the timescale of the input and output signals, and
5. if the reservoir consists of interconnected elements, then the connectivity of these elements can also be tuned which influences the dynamical regime of the reservoir.

The effect of these parameters on the reservoir's final performance is explained in section 3.5.2.

(b) **Prepare input and output signal**. The input and output signals need to be pre-processed before they are suitable for RC. First the data will be randomly split into a training set and a testing set, commonly using 80% of the data as testing and the remaining data for testing. The training data is further split into training and validation (see step (f)). The signals then need to be cleaned by the removal of missing values and outliers. One can define a data point at time t as an outlier if its Z-score is above a certain cutoff value, e.g. $Z = 3$. The data is often normalized to (mostly) lie in the interval $[-1, +1]$. This can be done with the min-max scaling $x_i \mapsto 2 \cdot \frac{u_i - \min(u)}{\max(u)} - 1$ where $\min(u)$ ($\max(u)$) is the minimum (maximum) value in the training data, or zero-mean unit-variance scaling $u_i \mapsto \frac{u_i - m u_u}{\sigma_u}$ where μ_u is the mean value of the training data and σ_u is the variance of the training data. The scaling of the input may also be adjusted to affect the dynamical regime of the reservoir. If a signal is categorical, rather than numerical, then it is often converted to its one-hot encoding. If the signal contains one of k possible categorical values $\{v_i\}_{i\in[1,\ldots,k]}$, then each time step is encoded as a k-dimensional vector which is zero everywhere except at the index i which corresponds to the categorical value v_i. If the signal is fed into the reservoir by a digital computer, it will have to be discretized with some frequency. The signal can be sub- or supersampled so that it best matches the timescale of the physical reservoir.

(c) **Analyze the reservoir** (optional). Before the reservoir is actually trained on the learning task, it can be instructive to analyze the reservoir upon feeding the input signal. Two kinds of analyses are particularly helpful. Firstly, the length of the washout period can be assessed by feeding the same input signal into the reservoir with different initial conditions. The two trajectories of reservoir states over time should converge, and the time until convergence indicates a suitable duration of the washout period. Secondly, the memory length of the reservoir can be assessed by training the reservoir's output map on recalling a delayed version of the input signal $u(t - \tau)$. The performance of the reservoir on this task should decrease when increasing the delay τ, and the speed of this decrease is indicative of the reservoir's memory length.

(d) **Collect reservoir states**. In order to train the reservoir, one first feeds in the training input signal and collects the reservoir states for the entire duration of the input signal. If the training data contains multiple input signals, they can be fed into the reservoir successively while keeping track of the time steps at which each input signal ended (see washout period in the next step). The reservoir states are stored on a digital computer and time must therefore be discretized with some fixed time step Δt which is typically chosen to be equal to the time step in the input signal.

(e) **Train readout map**. The readout map from reservoir states to the output signal is typically found by linear regression using the MSE loss, see section 3.4. However, other loss functions can be used. The cross-entropy loss is a good choice for classification problems when the output of the reservoir is an estimate of probabilities for each class. Furthermore, it is also possible to use more sophisticated functions as a readout map, for example a FNN. This may be done if the reservoir itself is not powerful enough to separate different input signals.

(f) **Optimization and evaluation loop**. Once the readout map is trained, the performance of the reservoir system can be evaluated. We generally compute the performance on the training data as well as on the

testing (or validation) data. If the training performance is much better than the testing performance, the reservoir may be overfitting on the training data and therefore not generalizing well to the testing data. In this case, it is often useful to increase the regularization in the readout map, e.g., by increasing α in the ridge regression. Generally, it is expected that the performance of the reservoir will not be ideal after the first training procedure. Typically, the tuning parameters that were identified in step (a) (see above) must be optimized to get a well-performing reservoir. In order to optimize these parameters, cross-validation (CV, also called out-of-sample testing) is a powerful tool. In CV, one further divides the training data into a training set and a validation set. The validation set now takes the role of the testing data, i.e., being used to evaluate the performance on unseen data. In k-fold CV, the training data is divided into k 'folds' and the training is done k times, each time using $k-1$ folds for training and the remaining fold for validation. The final performance is averaged over all folds. Evaluating all possible parameters is often intractable because of the combinatorial explosion of possible configurations, especially if such tuning involves changing physical properties about the reservoir. Therefore one typically retreats to optimize parameters iteratively, one by one. Such an optimization and evaluation loop is demonstrated in this paper's accompanying GitHub repository[7].

4. Physical reservoirs and applications

A learning task can be either a classification or a regression, it can be temporal or static. It is in general quite difficult to say what applications are well-suited to be tackled by any given physical reservoir. Even when systems display chaotic behavior, a well-designed reservoir can easily handle them. In the past, researchers have described the memory capacity of a reservoir to assess the timescales of its memory and decided which learning tasks may be solved by this reservoir. In addition to the memory capacity, the kind of nonlinearity, the size of the reservoir, and other attributes also indicate what tasks a reservoir is well-suited for. From an application perspective, it may be desirable to use learning tasks that are relevant for a given domain. For example, an optical reservoir may be useful for communication tasks like header recognition while a biochemical reservoir may be more useful for the detection of ECG arrhythmia. For this reason, one could categorize physical reservoirs based on the origin of the nonlinearity and type of signals that are injected/readout, thus distinguishing between (opto)electronic, photonic, mechanical, chemical and physical LSMs (cf e.g., figure 9).

4.1. Optoelectronic reservoirs

Since the early physical realization of reservoirs, researchers have taken advantage of nonlinear circuit elements or integrated systems as building blocks for efficient electronic reservoirs. For example, transistors have been explored by Nako *et al* which realized a novel calculation scheme utilizing the nonlinearity and memory effect of ferroelectric MOSFETs [96]. Integrated solutions have been explored such as field-programmable gate arrays (FPGAs), which can be exploited for computation thanks to their programmability and the ease in reconfiguration of the reservoirs. Moreover, their compatibility with commonly employed silicon semiconductor fabrication allows for implementation with conventional electronics. FPGA boards have been used to perform time series prediction tasks [61, 97], and recently have been combined with optical elements to boost their performances, taking advantage of the electro-optic nonlinear transformations and ease of the delay logic implementation in FPGAs [98]. There is a increasing attention to electronic reservoirs based on memristive devices, given their high degree of nonlinearity, which is based on the changes in resistance with the history of the current flow [99]. A network of memristors can be used as a reservoir to map an input signal into a higher dimensional space using nonlinear transformations. Kulkarni and Teuscher proposed the first reservoir of this kind, which performed a wave pattern classification task and an associative memory task on a genetic algorithm [59]. Later, several studies have more deeply demonstrated implementations of memristive reservoirs. Du *et al* revolutionized the concept of a network of dependent nonlinear devices and proposed a reservoir made by a group of independent memristors [60]. A single input signal was partitioned into the reservoir where each segment was independently transformed into an output signal by the corresponding memristor. Using this approach, the reservoir produced good results in image recognition and time series prediction tasks. Milano *et al* showed an interwoven net of nanow.res whose contact points behavors [100] Pioneering work has been done on reconfigurable reservoirs by Zhang *et al* [101], where they demonstrate adjustable neuron-like features in perovskite $NdNiO_3$ devices with single-shot electric pulses.

Usually, optimized electronic reservoirs are hardware-intensive neural networks, therefore in applications where minimal hardware are required, photonic reservoirs can be preferred over electronic reservoirs. Together with that, photonic reservoirs boast the advantages of very low power consumption and

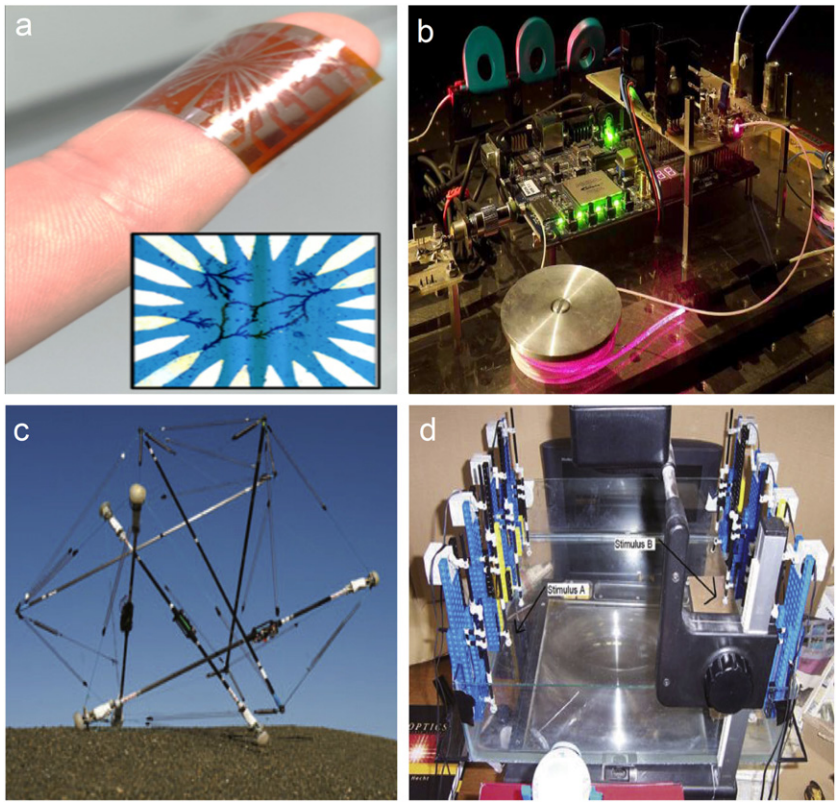

Figure 9. (a) Application of an OECT-based reservoir on biocompatible substrate. Reprinted from [12] with permission from the AAAS. (b) Photonic reservoir integrated with FPGAs. Reprinted with permission from [94]. Credits for the picture go to Romain Martinenghi. (c) Planetary rovers realized with mechanical reservoirs. Reprinted from [95] with permission from the Royal Society of Chemistry. (d) Physically implemented LSM in a bucket. Reprinted from [67], with permission from Springer Nature.

extremely fast computation. The first architecture of a photonic reservoir computer was modeled as a chip-integrated device with single-mode waveguides by Vandoorne *et al* [84]. The same authors later provided the first physical realization of the model [102] to perform logical operations, header recognition, and spoken digit classification. Another design strategy can make use of a platform in which the nodes are nonlinear micro-ring resonators, as shown by Mesaritakis *et al* [103]. Vinckier *et al* reported a reservoir made of a coherently-driven passive cavity able to perform speech recognition tasks, resulting in state-of-the-art detection accuracy [104]. More recently, a system of multiple inputs and outputs was built using photonic crystal cavities and it was shown to exhibit memory capacity up to six bits [105]. Approaches featuring both photonic and electronic integrated boards have also been explored for cryptography applications [94]. A more exotic application was reported by Zhao *et al*, in which an optical system with two feedback loops was used to perform simultaneous recognition of packet headers for two optical channels [106]. Scalable optical approaches have been proposed lately to compute challenging operations at a very high rate. Nakajima *et al* showed optimal time-series forecasting and image classifications with record-high processing speed using coherent linear photonic processors, where the input signals and reservoir weights are optically encoded in the spatiotemporal domain, enabling scalable integration on a compact chip [107].

4.2. Electrochemical reservoirs

Chemical reservoirs can be exploited to produce autonomous computational systems that, although having dynamics limited in speed by the reaction and diffusion rates, could be exploited for specific applications in liquid environments. Very recently, examples of chemical RCs that exploit redox reactions have been proposed showing excellent performances and great potential for biosensing [108] as well as machine-learning *in vitro* and *in vivo*. For examples, Kan *et al* exploited the nonlinear spatial and temporal dynamics of an electrolytic solution containing polyoxometalate [109]. Such a compound can undergo multiple oxidation and reduction states, producing Faraidic reactions with the metallic electrode upon the application of a voltage. The authors showed that an electrolytic solution alone can be used a random network

for periodic time-series predictions. The electrolyte usefulness in RC can be better leveraged, and better interfaced to traditional electronics, if the ionic dynamics are converted into electronic signals using OMIECs and OECTs. A first proof-of-concept was demonstrated by Pecquer *et al* [110], where a gate signal mediates the change in drain current in 16 channels. Key in this work is the demonstration that ionic-to-electronic transduction using OECTs can be used for machine learning and that device-to-device variability, an ubiquitous factor in organic electronics, can be harnessed as a feature. Their work showed that such a system can be used to distinguish different time-dependent signals, like square waves from triangular waves. This reservoir is still 'consciously-designed' and requires precise microfabrication of the transistor channels.

Cucchi *et al* built upon this idea and produced reservoirs made of electropolymerized OECTs. The electropolymerization offers several advantages such as easy and cheap lithography-free fabrication as well as dendritic topologies [111]. The latter emerges as an important feature when several fibers are operated in global gate conditions. The voltage drops along a fiber between the two metal connections. However, if a fiber branches out and terminates in the electrolyte, its potential will be constant as its resistance is much lower than the one of the electrolyte. Therefore, the gating effect of such branch on a neighboring fiber is stronger and builds up during the charging transient time. This grants rich nonlinear behaviors [68]. Clever (although inherently random or semirandom) design of the networks allows for remarkable results in machine learning tasks, such as flower recognition using the iris dataset. The networks worked much better using time-varying signals, highlighting the usefulness of the ionic dynamics and the nonlinear charging transient of the fibers. When a delay-line is added to the network, therefore implementing a single-node RC architecture, complex tasks such as time-prediction and heartbeat recognition can be carried out with excellent accuracy, 96% and 88%, respectively [12]. Another example is demonstrated by Usami *et al* who covered a set of circular electrodes with a film of sulfonated polyaniline [112]. The film is then immersed in an electrolytic solution. Such method can be advantageous as it does not require any patterning and was shown to be successful for time-dependent classification such as spoken digits with an accuracy of 70%.

4.3. Reservoirs outside the optoelectronic framework.

In order to easily achieve complex behaviors and nonlinear dynamics, mechanical systems can be outsourced as physical reservoirs. Hauser *et al* proposed a body-spring reservoir model, where neighboring masses were randomly connected via nonlinear springs [113]. The input signal is an external force that propagates through the network, while the output signal is chosen to be a linear combination of the lengths of the springs. This system finds its major applicability in generating patterns for robotic locomotion and the study of movement in biological systems. This architecture later evolved in a tensegrity reservoir model by substituting the springs with rigid bars. The control implementation in tensegrity reservoirs was then investigated and implemented to create planetary rovers [95]. Most of the interest in reservoir computers based on mechanical models has been shown towards soft robotics [41, 114], and in particular in generating different robot motion and behaviors, as in the case of the pneumatically driven soft robot arm proposed by Eder *et al* [115], and the dog-inspired quadruped robot developed by Wyffels *et al* [116]. Other efforts in establishing new archetypes of computation have been made in the field of chemistry and fluid dynamics. Chemical networks have been firstly investigated mainly to shine light on the secrets of the origins of life. Random chemical networks have been shown to produce self-replication properties [117] and oscillatory behaviors [118, 119]. One of the first biochemical networks was assembled by exploiting a random distribution of peptides, which showed self-organization for predicted network connectivity [120]. Randomness, which is a key feature for a computational system, has also been explored for chemical networks in the case of deoxyribozyme oscillators, shown by Farfel and Stefanovic [121] and DNA strand displacement circuits, by Goudarzi *et al* [122] and Yahiro *et al* [123]. As outlined in section 2.4.3, LSMs are considered as a computational model of SNNs with recurrent connections in a reservoir. One of the first examples of LSM models has been demonstrated by Fernando and Sojakka [67], where the authors used a bucket of water as an implementation of the LSM model. In this scenario, the input layers were injected by motors in the bucket, and projections of the surface of the water were used as liquid states to perform pattern recognition tasks. The full potential of LSMs is unlocked when this approach is used to mimic the computation of biological neurons or regulatory genes, which are embedded in a liquid environment. The strength of using equivalent models of spiking neurons (mimicking soma and synaptic regions) is to carry out computationally demanding classification tasks in real-time. Maas *et al* first demonstrated an LSM with sufficient depth and heterogeneity to serve as a fading memory [86]. Additionally, Jones *et al* first demonstrated that genetics of *Escherichia coli* can be modeled assuming that time-dependent inputs are external factors that produce transcription rates changes in genes, taken as the time-dependent output signals [124]. Encouraging results have been obtained in the years regarding applications of LSMs

in many disciplines, spanning from robotics [125], to music production [126]. Although the exciting perspectives advanced by the use of LSMs in the last years, geometrical and power constraints of the traditional LSMs cannot easily deal with resource-intensive state-of-the-art algorithms. For this reason, other types of computing have been generally preferred due to the lower demand for computational power and complexity. As a solution for this, Soures and Kudithipudi showed a novel approach for performing high-demanding spatio-temporal tasks: video activity recognition with their platform based on deep-LSMs outperformed state-of-the-art algorithms in accuracy, consuming less memory storage and resources [127]. Last but not least, nonlinear quantum effects, such as entanglement or coherence, can be harnessed for information processing using the RC framework while easing the strict requirements and complex fabrication constraints of quantum computing. The first formalization of quantum computing is relatively recent and goes back to 2017, when Fujii and Nakajima showed that a few qubits have the computational power of RNNs of hundreds of nodes [128]. Since then the field has been growing, also boosted by advances in hardware for quantum computing. As a result, a number of interesting demonstration have been put forth leveraging spin dynamics [129, 130], magnetic resonance [131], or superconductive systems [132]. It is important to note that many quantum systems are linear. As such, they should not be suitable candidates for RC. However, as demonstrated by Fujii and Nakajima, first-order linear systems feature second-order nonlinearities that can be harnessed in RC networks [128].

5. Conclusions

Analog data processing using nonlinear material systems is a rapidly-growing field that is envisioned to bring about novel computational substrates and paradigms where latency and power dissipation are minimized. However, there is a considerable mismatch between the algorithmic implementation of AI on digital machines and the physical realization of physical/hardware computing networks used in material science. Such discrepancy limits the development of physical systems for AI, and it is exacerbated if particular machine learning approaches are used. One of these, called RC, represents a promising route for hardware-based RNNs. Despite its usefulness, the existing literature on RC for beginners in machine learning can be cumbersome. Therefore, we tried to bridge these two worlds and provide a guide on how to exploit a nonlinear system, material, or device for information processing using the RC approach. With the aid of examples and data, we highlighted the essential principles and requirements needed and outlined how to build an experimental setup.

Acknowledgments

We thank Herbert Jaeger for discussions and valuable comments on early drafts of this paper. This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Grant Agreement No. 860360 (POST DIGITAL) and from the Hector Fellow Academy (30000619). Furthermore, the authors thank the Bundesministerium für Bildung und Forschung (BMBF) for funding from the project BAYOEN (01IS21089).

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

ORCID iDs

Daniel Brunner https://orcid.org/0000-0002-4003-3056
Hans Kleemann https://orcid.org/0000-0002-9773-6676

References

[1] Lukoševičius M 2012 A practical guide to applying echo state networks *Neural Networks: Tricks of the Trade* (Berlin: Springer) pp 659–86
[2] Shulman R G, Rothman D L, Behar K L and Hyder F 2004 Energetic basis of brain activity: implications for neuroimaging *Trends Neurosci.* **27** 489–95
[3] Fox M D and Raichle M E 2007 Spontaneous fluctuations in brain activity observed with functional magnetic resonance imaging *Nat. Rev. Neurosci.* **8** 700–11
[4] Jaeger H 2021 Towards a generalized theory comprising digital, neuromorphic, and unconventional computing *Neuromorphic Comput. Eng.* **1** 012002

[5] Hermans M, Schrauwen B, Bienstman P and Dambre J 2014 Automated design of complex dynamic systems *PLoS One* **9** e86696
[6] Zoppo G, Marrone F and Corinto F 2020 Equilibrium propagation for memristor-based recurrent neural networks *Front. Neurosci.* **14** 240
[7] Wright L G, Onodera T, Stein M M, Wang T, Schachter D T, Hu Z and McMahon P L 2022 Deep physical neural networks trained with backpropagation *Nature* **601** 549–55
[8] Nakajima M, Inoue K, Tanaka K, Kuniyoshi Y, Hashimoto T and Nakajima K 2022 Physical deep learning with biologically plausible training method https://doi.org/10.48550/arXiv.2204.13991
[9] Yao P, Wu H, Gao B, Tang J, Zhang Q, Zhang W, Yang J J and Qian H 2020 Fully hardware-implemented memristor convolutional neural network *Nature* **577** 641–6
[10] Berggren K *et al* 2020 Roadmap on emerging hardware and technology for machine learning *Nanotechnology* **32** 012002
[11] Larger L, Soriano M C, Brunner D, Appeltant L, Gutierrez J M, Pesquera L, Mirasso C R and Fischer I 2012 Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing *Opt. Express* **20** 3241–9
[12] Cucchi M *et al* 2021 Reservoir computing with biocompatible organic electrochemical networks for brain-inspired biosignal classification *Sci. Adv.* **7** eabh0693
[13] Tanaka G, Yamane T, Héroux J B, Nakane R, Kanazawa N, Takeda S, Numata H, Nakano D and Hirose A 2019 Recent advances in physical reservoir computing: a review *Neural Netw.* **115** 100–23
[14] Dale M, Miller J F, Stepney S and Trefzer M A 2021 Reservoir computing in material substrates *Reservoir Computing* (Berlin: Springer)
[15] Jaeger H 2001 The 'echo state' approach to analysing and training recurrent neural networks *Technical Report GMD Report 148* (GMD-German National Research Institute for Computer Science)
[16] Jaeger H 2002 Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the 'echo state network' approach (Bonn: GMD-Forschungszentrum Informationstechnik)
[17] Maass W, Natschläger T and Markram H 2002 Real-time computing without stable states: a new framework for neural computation based on perturbations *Neural Comput.* **14** 2531–60
[18] Jaeger H 2021 *Reservoir Computing: Theory, Physical Implementations, and Applications* (Berlin: Springer)
[19] Kirby K G 1991 Context dynamics in neural sequential learning *Proc. Florida AI Research Symp. (FLAIRS)* pp 66–70
[20] Schomaker L R B 1992 A neural oscillator-network model of temporal pattern generation *Hum. Mov. Sci.* **11** 181–92
[21] Dominey P F 1995 Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning *Biol. Cybern.* **73** 265–74
[22] Dominey P F 2021 Cortico-striatal origins of reservoir computing, mixed selectivity, and higher cognitive function *Reservoir Computing: Theory, Physical Implementations, and Applications* (Berlin: Springer) pp 29–58
[23] Verstraeten D, Schrauwen B, D'Haene M and Stroobandt D 2007 An experimental unification of reservoir computing methods *Neural Netw.* **20** 391–403
[24] Nakajima K and Fischer I (ed) 2021 *Reservoir Computing* (Berlin: Springer)
[25] Grigoryeva L and Ortega J-P 2018 Universal discrete-time reservoir computers with stochastic inputs and linear readouts using non-homogeneous state-affine systems *J. Mach. Learn. Res.* **19** 892–931
[26] Grigoryeva L and Ortega J-P 2018 Echo state networks are universal *Neural Netw.* **108** 495–508
[27] Dale M, Stepney S, Miller J F and Trefzer M 2017 Reservoir computing *in materio*: a computational framework for *in materio* computing *2017 Int. Joint Conf. Neural Networks (IJCNN)* (IEEE)
[28] Adamatzky A (ed) 2018 *Unconventional Computing* (Berlin: Springer)
[29] Horsman C, Stepney S, Wagner R C and Kendon V 2014 When does a physical system compute? *Proc. R. Soc.* A **470** 20140182
[30] Steil J J 2007 Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning *Neural Netw.* **20** 353–64
[31] Paugam-Moisy H, Martinez R and Bengio S 2008 Delay learning and polychronization for reservoir computing *Neurocomputing* **71** 1143–58
[32] Wen G, Li H and Li D 2015 An ensemble convolutional echo state networks for facial expression recognition *2015 Int. Conf. Affective Computing and Intelligent Interaction (ACII)* pp 873–8
[33] Gallicchio C, Micheli A and Pedrelli L 2017 Deep reservoir computing: a critical experimental analysis *Neurocomputing* **268** 87–99
[34] Jaeger H and Haas H 2004 Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication *Science* **304** 78–80
[35] Pathak J, Hunt B, Girvan M, Lu Z and Ott E 2018 Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach *Phys. Rev. Lett.* **120** 024102
[36] McCulloch W S and Pitts W 1943 A logical calculus of the ideas immanent in nervous activity *Bull. Math. Biophys.* **5** 115–33
[37] Huang G, Huang G-B, Song S and You K 2015 Trends in extreme learning machines: a review *Neural Netw.* **61** 32–48
[38] James G, Witten D, Hastie T and Tibshirani R 2021 *An Introduction to Statistical Learning* (Berlin: Springer)
[39] Rumelhart D E, Hinton G E and Williams R J 1986 Learning representations by back-propagating errors *Nature* **323** 533–6
[40] Hirose A, Takeda S, Yamane T, Numata H, Kanazawa N, Héroux J B, Nakano D, Nakane R and Tanaka G 2019 Physical reservoir computing: possibility to resolve the inconsistency between neuro-AI principles and its hardware *Aust. J. Intell. Inf. Process. Syst.* **16** 49–55 http://ajiips.com.au/papers/V16.4/v16n4_53-59.pdf
[41] Nakajima K 2020 Physical reservoir computing—an introductory perspective *Japan. J. Appl. Phys.* **59** 060501
[42] van de Burgt Y, Lubberman E, Fuller E J, Keene S T, Faria G C, Agarwal S, Marinella M J, Alec Talin A and Salleo A 2017 A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing *Nat. Mater.* **16** 414–8
[43] Demasius K-U, Kirschen A and Parkin S 2021 Energy-efficient memcapacitor devices for neuromorphic computing *Nat. Electron.* **4** 748–56
[44] Stathopoulos S, Khiat A, Trapatseli M, Cortese S, Serb A, Valov I and Prodromakis T 2017 Multibit memory operation of metal-oxide bi-layer memristors *Sci. Rep.* **7** 17532
[45] Adam G C, Khiat A and Prodromakis T 2018 Challenges hindering memristive neuromorphic hardware from going mainstream *Nat. Commun.* **9** 5267
[46] Cragg B G 1967 The density of synapses and neurones in the motor and visual areas of the cerebral cortex *J. Anat.* **101** 639
[47] Gelenbe E 1993 Learning in the recurrent random neural network *Neural Comput.* **5** 154–64
[48] Gelenbe E 1989 Random neural networks with negative and positive signals and product form solution *Neural Comput.* **1** 502–10
[49] Yildiz I B, Jaeger H and Kiebel S J 2012 Re-visiting the echo state property *Neural Netw.* **35** 1–9

[50] Jaeger H 2007 Echo state network *Scholarpedia* **2** 2330
[51] Jaeger H 2002 Short term memory in echo state networks *Technical Report GMD Report 152* (GMD-German National Research Institute for Computer Science)
[52] Schuecker J, Goedeke S and Helias M 2018 Optimal sequence memory in driven random networks *Phys. Rev.* X **8** 041029
[53] Haruna T and Nakajima K 2019 Optimal short-term memory before the edge of chaos in driven random recurrent networks *Phys. Rev.* E **100** 062312
[54] Durstewitz D, Seamans J K and Sejnowski T J 2000 Neurocomputational models of working memory *Nat. Neurosci.* **3** 1184–91
[55] Pascanu R and Jaeger H 2011 A neurodynamical model for working memory *Neural Netw.* **24** 199–207
[56] Sourikopoulos I, Hedayat S, Loyez C, Danneville F, Hoel V, Mercier E and Cappy A 2017 A 4-fJ/spike artificial neuron in 65 nm CMOS technology *Front. Neurosci.* **11** 123
[57] Mead C 1990 Neuromorphic electronic systems *Proc. IEEE* **78** 1629–36
[58] Appeltant L, Soriano M C, Van der Sande G, Danckaert J, Massar S, Dambre J, Schrauwen B, Mirasso C R and Fischer I 2011 Information processing using a single dynamical node as complex system *Nat. Commun.* **2** 468
[59] Kulkarni M S and Teuscher C 2012 Memristor-based reservoir computing *2012 IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)* (IEEE) pp 226–32
[60] Du C, Cai F, Zidan M A, Ma W, Lee S H and Lu W D 2017 Reservoir computing using dynamic memristors for temporal information processing *Nat. Commun.* **8** 2204
[61] Penkovsky B, Larger L and Brunner D 2018 Efficient design of hardware-enabled reservoir computing in FPGAs *J. Appl. Phys.* **124** 162101
[62] Guo X X, Xiang S Y, Zhang Y H, Lin L, Wen A J and Hao Y 2019 Four-channels reservoir computing based on polarization dynamics in mutually coupled VCSELs system *Opt. Express* **27** 23293–306
[63] Taniguchi T, Tsunegi S, Miwa S, Fujii K, Kubota H and Nakajima K 2021 Reservoir computing based on spintronics technology *Reservoir Computing* (Berlin: Springer) pp 331–60
[64] Torrejon J *et al* 2017 Neuromorphic computing with nanoscale spintronic oscillators *Nature* **547** 428–31
[65] Furuta T, Fujii K, Nakajima K, Tsunegi S, Kubota H, Suzuki Y and Miwa S 2018 Macromagnetic simulation for reservoir computing utilizing spin dynamics in magnetic tunnel junctions *Phys. Rev. Appl.* **10** 034063
[66] Tsunegi S, Taniguchi T, Nakajima K, Miwa S, Yakushiji K, Fukushima A, Yuasa S and Kubota H 2019 Physical reservoir computing based on spin torque oscillator with forced synchronization *Appl. Phys. Lett.* **114** 164101
[67] Fernando C and Sojakka S 2003 Pattern recognition in a bucket *European Conf. Artificial Life* (Springer) pp 588–97
[68] Petrauskas L, Cucchi M, Grüner C, Ellinger F, Leo K, Matthus C and Kleemann H 2022 Nonlinear behavior of dendritic polymer networks for reservoir computing *Adv. Electron. Mater.* **8** 2100330
[69] Matteo C 2021 iris_dynamicx10_2 zip *Figshare Dataset* https://doi.org/10.6084/m9.figshare.17206934.v1
[70] Sun Z, Pedretti G, Bricalli A and Ielmini D 2020 One-step regression and classification with cross-point resistive memory arrays *Sci. Adv.* **6** eaay2378
[71] Gordon-Rodriguez E, Loaiza-Ganem G, Pleiss G and Cunningham J P 2020 Uses and abuses of the cross-entropy loss: case studies in modern deep learning *Proc. 'I Can't Believe It's Not Better!' at NeurIPS Workshops* (Proceedings of Machine Learning Research (PMLR) vol 137) ed J Z Forde, F Ruiz, M F Pradier and A Schein pp 1–10
[72] Subramoney A, Scherr F and Maass W 2021 Reservoirs learn to learn *Reservoir Computing* (*Natural Computing Series*) (Berlin: Springer) pp 59–76
[73] Bohnstingl T, Scherr F, Pehle C, Meier K and Maass W 2019 Neuromorphic hardware learns to learn *Front. Neurosci.* **13** 483
[74] Goudarzi A and Teuscher C 2016 Reservoir computing: quo vadis? *Proc. 3rd ACM Int. Conf. Nanoscale Computing and Communication* (ACM)
[75] Dale M, Miller J F, Stepney S and Trefzer M A 2019 A substrate-independent framework to characterize reservoir computers *Proc. R. Soc.* A **475** 20180723
[76] Dambre J, Verstraeten D, Schrauwen B and Massar S 2012 Information processing capacity of dynamical systems *Sci. Rep.* **2** 514
[77] Elsken T, Metzen J H and Hutter F 2019 Neural architecture search: a survey *J. Mach. Learn. Res.* **20** 1997–2017
[78] Kubota T, Takahashi H and Nakajima K 2021 Unifying framework for information processing in stochastically driven dynamical systems *Phys. Rev. Res.* **3** 043135
[79] Moon J, Wu Y and Lu W D 2021 Hierarchical architectures in reservoir computing systems *Neuromorphic Comput. Eng.* **1** 014006
[80] Cramer B, Stöckel D, Kreft M, Wibral M, Schemmel J, Meier K and Priesemann V 2020 Control of criticality and computation in spiking neuromorphic networks with plasticity *Nat. Commun.* **11** 2853
[81] Sussillo D and Abbott L F 2009 Generating coherent patterns of activity from chaotic neural networks *Neuron* **63** 544–57
[82] Inoue K, Nakajima K and Kuniyoshi Y 2020 Designing spontaneous behavioral switching via chaotic itinerancy *Sci. Adv.* **6** eabb3989
[83] Laje R and Buonomano D V 2013 Robust timing and motor patterns by taming chaos in recurrent neural networks *Nat. Neurosci.* **16** 925–33
[84] Vandoorne K, Dambre J, Verstraeten D, Schrauwen B and Bienstman P 2011 Parallel reservoir computing using optical amplifiers *IEEE Trans. Neural Netw.* **22** 1469–81
[85] Coulombe J C, York M C A and Sylvestre J 2017 Computing with networks of nonlinear mechanical oscillators *PLoS One* **12** e0178663
[86] Maass W, Natschläger T and Markram H 2004 Fading memory and kernel properties of generic cortical microcircuit models *J. Physiol.* **98** 315–30
[87] Qiao J, Li F, Han H and Li W 2017 Growing echo-state network with multiple subreservoirs *IEEE Trans. Neural Netw. Learn. Syst.* **28** 391–404
[88] Chatzidimitriou K C and Mitkas P A 2013 Adaptive reservoir computing through evolution and learning *Neurocomputing* **103** 198–209
[89] Xue F, Hou Z and Li X 2013 Computational capability of liquid state machines with spike-timing-dependent plasticity *Neurocomputing* **122** 324–9
[90] Lu Z, Hunt B R and Ott E 2018 Attractor reconstruction by machine learning *Chaos* **28** 061104
[91] Kim J Z, Lu Z, Nozari E, Pappas G J and Bassett D S 2021 Teaching recurrent neural networks to infer global temporal structure from local examples *Nat. Mach. Intell.* **3** 316–23
[92] Jaeger H 2014 Conceptors: an easy introduction https://doi.org/10.48550/arXiv.1406.2671
[93] Jaeger H 2014 Controlling recurrent neural networks by conceptors https://doi.org/10.48550/arXiv.1403.3369

[94] Larger L, Martinenghi R, Jacquot M and Chembo Y K 2013 Complex photonic nonlinear delay dynamics for high performance signal and information processing *Int. Conf. Optics & Photonics Taïwan (OPTIC 2013)*

[95] Caluwaerts K, Despraz J, Işçen A, Sabelhaus A P, Bruce J, Schrauwen B and SunSpiral V 2014 Design and control of compliant tensegrity robots through simulation and hardware validation *J. R. Soc. Interface* **11** 20140520

[96] Nako E, Toprasertpong K, Nakane R, Wang Z, Miyatake Y, Takenaka M and Takagi S 2020 Proposal and experimental demonstration of reservoir computing using $Hf_{0.5}Zr_{0.5}O_2$/Si FeFETs for neuromorphic applications *2020 IEEE Symp. VLSI Technology* (IEEE) pp 1–2

[97] Alomar M L, Canals V, Martínez-Moll V and Rosselló J L 2014 Low-cost hardware implementation of reservoir computers *2014 24th Int. Workshop Power and Timing Modeling, Optimization and Simulation (PATMOS)* (IEEE) pp 1–5

[98] Kumar P, Jin M, Bu T, Kumar S and Huang Y-P 2021 Efficient reservoir computing using field programmable gate array and electro-optic modulation *OSA Contin.* **4** 1086–98

[99] Chua L 1971 Memristor-the missing circuit element *IEEE Trans. Circuit Theory* **18** 507–19

[100] Milano G, Pedretti G, Montano K, Ricci S, Hashemkhani S, Boarino L, Ielmini D and Ricciardi C 2022 *In materia* reservoir computing with a fully memristive architecture based on self-organizing nanowire networks *Nat. Mater.* **21** 195–202

[101] Zhang H-T *et al* 2022 Reconfigurable perovskite nickelate electronics for artificial intelligence *Science* **375** 533–9

[102] Vandoorne K, Mechet P, Van Vaerenbergh T, Fiers M, Morthier G, Verstraeten D, Schrauwen B, Dambre J and Bienstman P 2014 Experimental demonstration of reservoir computing on a silicon photonics chip *Nat. Commun.* **5** 3541

[103] Mesaritakis C, Papataxiarhis V and Syvridis D 2013 Micro ring resonators as building blocks for an all-optical high-speed reservoir-computing bit-pattern-recognition system *J. Opt. Soc. Am.* B **30** 3048–55

[104] Vinckier Q, Duport F, Smerieri A, Vandoorne K, Bienstman P, Haelterman M and Massar S 2015 High-performance photonic reservoir computer based on a coherently driven passive cavity *Optica* **2** 438–46

[105] Laporte F, Katumba A, Dambre J and Bienstman P 2018 Numerical demonstration of neuromorphic computing with photonic crystal cavities *Opt. Express* **26** 7955–64

[106] Zhao Q, Yin H and Zhu H 2018 Simultaneous recognition of two channels of optical packet headers utilizing reservoir computing subject to mutual-coupling optoelectronic feedback *Optik* **157** 951–6

[107] Nakajima M, Tanaka K and Hashimoto T 2021 Scalable reservoir computing on coherent linear photonic processor *Commun. Phys.* **4** 20

[108] Przyczyna D, Pecqueur S, Vuillaume D and Szaciłowski K 2020 Reservoir computing for sensing: an experimental approach (arXiv:2001.04342)

[109] Kan S, Nakajima K, Asai T and Akai-Kasaya M 2021 Physical implementation of reservoir computing through electrochemical reaction *Adv. Sci.* **9** 2104076

[110] Pecqueur S, Mastropasqua Talamo M, Guérin D, Blanchard P, Roncali J, Vuillaume D and Alibart F 2018 Neuromorphic time-dependent pattern classification with organic electrochemical transistor arrays *Adv. Electron. Mater.* **4** 1800166

[111] Cucchi M, Kleemann H, Tseng H, Ciccone G, Lee A, Pohl D and Leo K 2021 Directed growth of dendritic polymer networks for organic electrochemical transistors and artificial synapses *Adv. Electron. Mater.* **7** 2100586

[112] Usami Y *et al* 2021 *In materio* reservoir computing in a sulfonated polyaniline network *Adv. Mater.* **33** 2102688

[113] Hauser H, Ijspeert A J, Füchslin R M, Pfeifer R and Maass W 2011 Towards a theoretical foundation for morphological computation with compliant bodies *Biol. Cybern.* **105** 355–70

[114] Nakajima K, Li T, Hauser H and Pfeifer R 2014 Exploiting short-term memory in soft body dynamics as a computational resource *J. R. Soc. Interface* **11** 20140437

[115] Eder M, Hisch F and Hauser H 2018 Morphological computation-based control of a modular, pneumatically driven, soft robotic arm *Adv. Robot.* **32** 375–85

[116] Wyffels F, D'Haene M, Waegeman T, Caluwaerts K, Nunes C and Schrauwen B 2010 Realization of a passive compliant robot dog *2010 3rd IEEE RAS & EMBS Int. Conf. Biomedical Robotics and Biomechatronics* (IEEE) pp 882–6

[117] Segré D, Lancet D, Kedem O and Pilpel Y 1998 Graded autocatalysis replication domain (GARD): kinetic analysis of self-replication in mutually catalytic sets *Orig. Life Evol. Biosph.* **28** 501–14

[118] Stadler P F, Fontana W and Miller J H 1993 Random catalytic reaction networks *Physica* D **63** 378–92

[119] Larger L, Penkovsky B and Maistrenko Y 2013 Virtual chimera states for delayed-feedback systems *Phys. Rev. Lett.* **111** 054103

[120] Ashkenasy G and Ghadiri M R 2004 Boolean logic functions of a synthetic peptide network *J. Am. Chem. Soc.* **126** 11140–1

[121] Farfel J and Stefanovic D 2005 Towards practical biomolecular computers using microfluidic deoxyribozyme logic gate networks *Int. Workshop DNA-Based Computers* (Springer) pp 38–54

[122] Goudarzi A, Lakin M R and Stefanovic D 2013 DNA reservoir computing: a novel molecular computing approach *Int. Workshop DNA-Based Computers* (Springer) pp 76–89

[123] Yahiro W, Aubert-Kato N and Hagiya M 2018 A reservoir computing approach for molecular computing *ALIFE 2018: The 2018 Conf. Artificial Life* (MIT Press) pp 31–8

[124] Jones B, Stekel D, Rowe J and Fernando C 2007 Is there a liquid state machine in the bacterium *Escherichia coli*? *2007 IEEE Symp. Artificial Life* (IEEE) pp 187–91

[125] Hertzberg J, Jaeger H and Schönherr F 2002 Learning to ground fact symbols in behavior-based robots *ECAI* vol 2 pp 708–12

[126] Jaeger H and Eck D 2008 Can't get you out of my head: a connectionist model of cyclic rehearsal *Modeling Communication with Robots and Virtual Humans* (Berlin: Springer) pp 310–35

[127] Soures N and Kudithipudi D 2019 Deep liquid state machines with neural plasticity for video activity recognition *Front. Neurosci.* **13** 686

[128] Fujii K and Nakajima K 2017 Harnessing disordered-ensemble quantum dynamics for machine learning *Phys. Rev. Appl.* **8** 024030

[129] Chen J and Nurdin H I 2019 Learning nonlinear input–output maps with dissipative quantum systems *Quantum Inf. Process.* **18** 198

[130] Martínez-Peña R, Nokkala J, Giorgi G L, Zambrini R and Soriano M C 2020 Information processing capacity of spin-based quantum reservoir computing systems *Cogn. Comput.* 1–12

[131] Negoro M, Mitarai K, Fujii K, Nakajima K and Kitagawa M 2018 Machine learning with controllable quantum dynamics of a nuclear spin ensemble in a solid (arXiv:1806.10910)

[132] Suzuki Y, Gao Q, Pradel K C, Yasuoka K and Yamamoto N 2022 Natural quantum reservoir computing for temporal information processing *Sci. Rep.* **12** 1353