Fall 11-15-2016

Theory and Practice of Computing with Excitable Dynamics

Alireza Goudarzi
University of New Mexico

Alireza Goudarzi

Candidate

Department of Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Darko Stefanovic, Chairperson

Stephanie Forrest

Lance Williams

Melanie Moses

Christof Teuscher

Theory and Practice of Computing with Excitable Dynamics

by

Alireza Goudarzi

M.S., Portland State University, 2012
M.S., Portland State University, 2011

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2016

Dedication

Dedicated to Junko whose support was invaluable for finishing this dissertation and to Cyrus whose face reminded me everyday that I have to do it sooner rather than later.

Acknowledgments

I would like to acknowledge invaluable comments and conversation not only from my committee members and my advisor but also my friends whose contributions helped shape this work: Sarah Marzen, Guy Feldman, and Alireza Shabani. I would also like to thank Mattia Rigotti, David Sussilo, and Taro Toyoizumi for constructive comments on the theoretical basis of this dissertation.

Theory and Practice of Computing with Excitable Dynamics

by

Alireza Goudarzi

M.S., Portland State University, 2012
M.S., Portland State University, 2011

Ph.D., Computer Science, University of New Mexico, 2016

Abstract

Reservoir computing (RC) is a promising paradigm for time series processing. In this paradigm, the desired output is computed by combining measurements of an excitable system that responds to time-dependent exogenous stimuli. The excitable system is called a *reservoir* and measurements of its state are combined using a *readout layer* to produce a target output. The power of RC is attributed to an emergent short-term memory in dynamical systems and has been analyzed mathematically for both linear and nonlinear dynamical systems. The theory of RC treats only the macroscopic properties of the reservoir, without reference to the underlying medium it is made of. As a result, RC is particularly attractive for building computational devices using emerging technologies whose structure is not exactly controllable, such as self-assembled nanoscale circuits.

RC has lacked a formal framework for performance analysis and prediction that goes beyond memory properties. To provide such a framework, here a mathematical theory of

memory and information processing in ordered and disordered linear dynamical systems is developed. This theory analyzes the optimal readout layer for a given task. The focus of the theory is a standard model of RC, the echo state network (ESN). An ESN consists of a fixed recurrent neural network that is driven by an external signal. The dynamics of the network is then combined linearly with readout weights to produce the desired output. The readout weights are calculated using linear regression.

Using an analysis of regression equations, the readout weights can be calculated using only the statistical properties of the reservoir dynamics, the input signal, and the desired output. The readout layer weights can be calculated from a priori knowledge of the desired function to be computed and the weight matrix of the reservoir. This formulation explicitly depends on the input weights, the reservoir weights, and the statistics of the target function. This formulation is used to bound the expected error of the system for a given target function. The effects of input-output correlation and complex network structure in the reservoir on the computational performance of the system have been mathematically characterized. Far from the chaotic regime, ordered linear networks exhibit a homogeneous decay of memory in different dimensions, which keeps the input history coherent. As disorder is introduced in the structure of the network, memory decay becomes inhomogeneous along different dimensions causing decoherence in the input history, and degradation in task-solving performance. Close to the chaotic regime, the ordered systems show loss of temporal information in the input history, and therefore inability to solve tasks. However, by introducing disorder and therefore heterogeneous decay of memory the temporal information of input history is preserved and the task-solving performance is recovered. Thus for systems at the edge of chaos, disordered structure may enhance temporal information processing. Although the current framework only applies to linear systems, in principle it can be used to describe the properties of physical reservoir computing, e.g., photonic RC using short coherence-length light.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The pursuit of fast ubiquitous computing has left us with two apparently conflicting goals. On one hand we seek ever faster and more reliable computers to perform a spectrum of signal-processing tasks, such as tracking, navigation, system identification, pattern recognition, and control. On the other hand the miniaturization trend in conventional silicon electronics to achieve faster speed is approaching its physical limits, resulting in higher energy consumption, more expensive fabrication, and more vulnerable devices; faster speed is undermining reliability and energy efficiency. The fields of natural computing and nanotechnology have emerged to explore alternative approaches to building computers. This pursuit has led to the development of various techniques, such as adaptive "programming" [1], and self-organization [2] strategies for a variety of underlying architectures including molecular switches, self-assembled nanowires, and DNA nanotechnology. However, all of these strategies rely on post-fabrication control of the microstructure of the underlying architectures to achieve their goals.

An alternative paradigm, called *intrinsic computation*, is to use the natural behavior

of a system for computation. In this dissertation I focus on a particular instance of intrinsic computing, called *reservoir computing* (RC), which uses the excitable dynamics of systems to perform real-time computation. I analyze excitable dynamics as a paradigm to build reliable application-specific computers using unconventional computer architectures. RC enables us to perform computation using the *collective dynamics* of a system, i.e., the global dynamics resulting from interactions of many elementary components, without relying on the specifics of the underlying architecture.

RC was initially proposed in the neural network community for the purpose of reducing the training time of recurrent neural networks. Unfortunately, the main analytical understanding of this approach was under an annealed approximation and most studies in this field relied on numerical experiments, which made it hard to draw firm conclusions about correctness and performance or to gain fundamental insight into the workings of RC.

To study RC in depth, I developed a mathematical framework to calculate an exact solution for the dynamics and the memory of a classical implementation of RC. I extended this framework to gain analytical insight into the expected performance of RC for real-world signal-processing tasks. Furthermore, I used the analytical results to calculate bounds on the estimated performance of a given system on a given task. Using this framework, I mathematically analyzed the effect of structure of the network on the performance of a given task. Although the formulation only applies to linear systems, in principle it can be used to describe the properties of physical reservoir computing, e.g., propagation of short coherence-length light through the reservoir medium. I hope that the insights here can help us develop useful approximations to nonlinear systems in the future.

1.2 Motivation

My three main sources of motivation for studying RC are its potential for (1) building application-specific and embedded systems using unconventional architectures, (2) building intelligence into smart matter and designing cyber-physical systems, and (3) bringing down the computational cost of training recurrent neural networks. I see the ultimate outcome of these three lines of research to be bringing down the cost of intelligent systems so they can be used widely.

Collective dynamics, i.e., global dynamics of systems with many interacting parts, has been long studied in cybernetics, artificial life, and complex systems communities [3, 4]. Applications of collective dynamics have only recently been realized as advances in materials science and nanotechnology have enabled the use of physical systems to perform computation for real-world signal processing scenarios. The term RC has been coined to refer to various frameworks in which computation takes place within the transient dynamics of a system. Within the community of intrinsic and natural computing there are other approaches that use the properties of physical systems to perform computation, such as chaos computing, neural networks, amorphous computing, and DNA computing. However, all these rely on manipulation of the microstructure of the underlying system. In contrast, RC takes a system as given, and only takes advantage of the dynamics of the system, without modifying its underlying structure. This makes RC particularly attractive for nanoscale self-assembled systems, whose structure cannot be engineered precisely. Moreover, RC has shown superb performance in chaotic time series analysis and prediction, and has become a valuable signal-processing tool [5–7]. Another advantage of RC is that it uses some aspects of kernel methods [8], in which the input is projected into a high-dimensional feature space. The features are linearly combined to produce the desired output. The linear coefficients in the process can be calculated using efficient closed-form solutions [8]. This is in contrast with common adaptive approaches, which use expensive iterative algorithms based on error backpropagation, genetic algorithms, or simulated

annealing [9]. In addition, RC has experimentally proven to be robust to physical implementation using many different architectures [10–13]. In addition, RC's performance has been shown to be resistant to failure of some of the components of the system or temporal variations in its structure [14, 15].

1.2.1 Unconventional Architectures

The approaching physical limits of silicon-based semiconductor technology are making conventional top-down designed computer architecture prohibitive [16]. Recent advances in materials science and nanotechnology suggest that unconventional computer architectures could be a viable technological and economical alternative for building energy-efficient and reliable application specific computers [17, 18]. Some proposed alternative architectures are based on molecular switches and memristive crossbars [19, 20] that possess highly regular structure. Another emerging approach is self-assembly of nanowires and memristive networks [21, 22], which results in irregular structure. Major obstacles to using such architectures are design variations, defects, faults, and susceptibility to environmental factors, such as thermal noise and radiation [23]. How should one *program* an unreliable system with unknown structure to perform reliable computation? Research in RC presents an opportunity to answer this question.

1.2.2 Smart Matter and Cyber-Physical Systems

Amorphous computing [4], a discipline that aims to find computing solutions using many interacting and unreliable parts. Two research fields that have emerged from amorphous computing are Smart materials, i.e., systems that can be programmed to behave according to a history of environmental stimuli, and cyber-physical systems, i.e., systems that can be programmed to interface with physical systems. One of the most promising developments in this area is DNA computing, which can have significant applications in health care, such

as monitoring and smart and localized drug delivery. So far, DNA computing focuses on implementing simple logic operation for identifying target chemical species and releasing appropriate chemicals to nullify the target. Using RC one could design DNA-based systems that monitor concentration of chemical species and respond appropriately.

1.2.3 Artificial Intelligence and Machine Learning

The past decade has seen a rapid development of artificial intelligence (AI) through advances in neural networks. AI systems based on deep and recurrent neural networks have been used to solve classification and control problems with human-level accuracy. Continuation of this trend inevitably makes intelligent systems as pervasive as computers: every computer, embedded, and cyber-physical system will incorporate an intelligent adaptive component. Computational cost is a critical obstacle to the wide deployment of intelligent systems. Cutting down the computational cost of intelligence is one of the main challenges I am tackling in my research. To this end, my research focuses on understanding and leveraging the most ubiquitous source of intelligence that naturally occurs in excitable physical systems: short-term memory.

1.3 Background and Related Work

The multidisciplinary nature of this dissertation can only be served by an appropriate multidisciplinary literature review. Here, after giving an overview of existing work in the field of RC (see Section 1.3.1), I try to give an adequate context for this dissertation in different fields to elucidate the significance of the work for each field. Broadly speaking, the idea of computing with transient states of the system puts RC in the realm of chaos computing and computing with dynamical systems (see Section 1.3.3). The systems discussed in chaos computing usually adhere to Turing universality of computation, i.e., the inputs are explic-

itly encoded into the dynamics of the system and the system is designed in such a way that its time evolution can reach any desired halting state. RC, on the other hand, works in the realm of real-time computing and although it has been proven to be a universal approximator [24], it was not originally intended to perform universal computation. As I mentioned above, the hallmark of RC is the absence of task-related modification to the microstructure of the underlying substrate, which makes it particularly suitable for building computational systems using emerging technologies at the nanoscale whose microstructure is difficult to engineer exactly. These include DNA nanotechnology and self-assembled circuits (see Section 1.3.4 and Section 1.3.5). The only adaptive part in RC is the readout layer, which is usually trained using ordinary linear regression (OLR) and follows the convergence theorems in the theory of learning and generalization (see Section 1.3.2). An important aspect of RC is how its structure affects the performance. Our main theoretical results explain this dependency in networks with linear transfer function and additive presynaptic integration. To reason about nonlinear systems, we then use computational tools to extend our results to multiplicative integration and nonlinear transfer function (see Section 1.3.6 and 1.3.7). These studies are motivated by the appearance of nonlinear effects in the transfer function as well as the presynaptic integration in biological neurons. Finally, I will summarize the background information and explain how our study relates to each field (see Section 1.4).

1.3.1 Reservoir Computing

The origin of reservoir computing is rooted in theoretical cognitive and neuroscience. Understanding contextual information processing in the brain is one of the long-standing challenges in neuroscience [25]. Specifically, the challenge emerges while explaining neurodynamic mechanisms that allow time-lagged stimulus-dependent behavior in humans and animals. Such behaviors require a working memory characterized by persistent neural activity following a received stimulus [26–31]. “Persistent activity provides the cellular basis of a central neural mechanism, as postulated by Hebb, ‘to account for the delay, between

stimulation and response, that seems so characteristic of thought?'. In order for a neural persistent activity to subserve working memory, it must be stimulus-selective, and therefore information-specific" [25, p. 455]. A popular approach to modeling persistent activity in neural networks is through attractor networks; however, these models do not address the design of the network to encode desired attractors [32]. Alternatively, Dominey et al. [33] proposed a simple model to explain the interaction between the prefrontal cortex, corticostriatal projections, and basal ganglia in context-dependent motor control of eyes. In this model, visual input drives stable activity in the prefrontal cortex, which is projected onto basal ganglia using learned interactions in the striatum. This model has also been used to explain higher-level cognitive tasks such as grammar comprehension in the brain [34]. This model could be called a precursor to reservoir computing, but it focused mainly on modeling the prefrontal cortex without any mention of generic task solving or common machine learning concepts such as learning and generalization.

RC was independently introduced by Maass, Natschläger, and Markram [24] under the name *liquid state machine*, and by Jaeger [35] under the name *echo state network* (ESN). Later the two approaches were unified under the name *reservoir computing* [36,37]. ESNs are one of the most popular RC paradigms, and have shown promising results in time series computing and prediction [5, 6], voice recognition [38], nonlinear system identification [39], and robot control [40]. In Ref. [5], it was shown that ESN outperformed the best state-of-the-art results achieved through approaches such as classical neural networks [41], chaos theory [42], and even one of the most popular architectures today, long short-term memory (LSTM) [43], by a factor of $2,400$ in terms of mean-squared-error.

An ESN [37,39,44,45] consists of an input-driven recurrent neural network, which acts as the reservoir, and a readout layer that reads the reservoir states and produces the output. Unlike a classical recurrent neural network where all the nodes are interconnected and their weights are determined during a training process, in ESN the nodes are interconnected using random weights and random sparse connectivity between the nodes. The input and

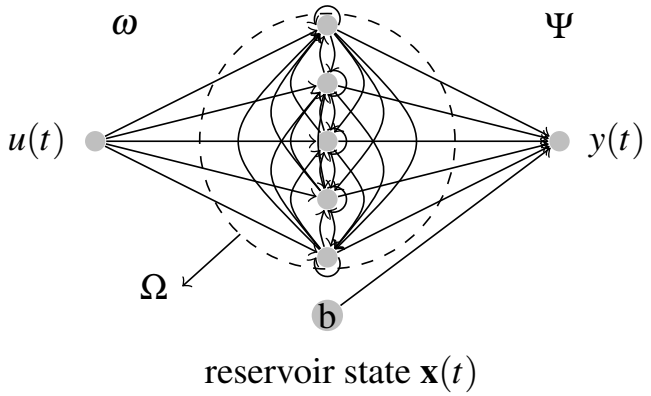


Figure 1.1: Schematic of an ESN. A dynamical core called a reservoir is driven by input signal $\mathbf{u}(t)$. The states of the reservoir $\mathbf{x}(t)$ extended by a constant 1, are combined linearly to produce the output $\mathbf{y}(t)$. The reservoir consists of N nodes interconnected with a random weight matrix Ω. The connectivity between the input and the reservoir nodes is represented with a randomly generated weight matrix ω. The reservoir states and the constant are connected to the readout layer using the weight matrix Ψ. The reservoir and the input weights are fixed after initialization, while the output weights are learned using a regression technique.

reservoir connections are initialized and fixed, usually with no further adaptation.

Figure 1.1 shows a schematic of an ESN. The readout layer is a linear combination of the reservoir states. The readout weights are determined for each task using supervised learning techniques, where the network is driven by a teacher input and its output is compared with a corresponding teacher output to estimate the error. Then, the weights are calculated using any closed-form regression technique [45] in offline training contexts or using adaptive techniques if online training is needed [39]. Mathematically, the input-driven reservoir is defined as follows. Let N be the size of the reservoir. We represent the time-dependent inputs as a scalar $u(t)$, the reservoir state as a column vector $\mathbf{x}(t)$, and the output as a scalar $y(t)$. in general input and output could be multidimensional (represented by column vectors), but in this dissertation we focus on scalar values. The input connectivity is represented by the matrix ω and the reservoir connectivity is represented by an $N \times N$ weight matrix Ω. For simplicity, we assume that we have one input signal

and one output, but the notation can be extended to multiple inputs and outputs. The time evolution of the reservoir is given by:

$$\mathbf{x}(t+1) = f(\Omega\mathbf{x}(t) + \omega u(t) + b). \tag{1.1}$$

where f is the transfer function of the reservoir nodes that is applied element-wise to its operand. This is usually the hyperbolic tangent, but sigmoidal or linear functions can be used as well. An optional bias term b can be used to adjust the nonlinearity of the reservoir [46]. The output is generated by the multiplication of an output weight matrix Ψ of length $N+1$ and the reservoir state vector $\mathbf{x}(t)$ extended by an optional constant 1:

$$y(t) = \Psi\mathbf{x}(t). \tag{1.2}$$

The output weights Ψ need to be trained using a teacher input-output pair. A common training technique is the pseudo-inverse method [37, 47, 48]. To use this method, one would drive the ESN with a teacher input and record the history of the reservoir states into a matrix $\mathbf{X} = [x_{it}] = [\mathbf{x}(0)|\ldots|\mathbf{x}(T)]$, where the columns are the reservoir state in time, and indices i and t refer to node index and time step respectively, and T is the total length of the input time series. A constant row of "1"s is added to $\mathbf{X}$ to serve as a bias. The corresponding teacher output will be denoted by the row vector $\widehat{\mathbf{Y}} = [y(0)|\ldots|y(T)]$. To keep the notation consistent I will refer to the expected output vectors with $\widehat{\mathbf{Y}}$ even though we are considering only 1-dimensional outputs. The readout can be calculated as follows pseudo-inverse solution to the least-squares problem [47]:

$$\Psi = \langle\mathbf{X}\mathbf{X}^\top\rangle^{-1}\langle\mathbf{X}\widehat{\mathbf{Y}}^\top\rangle, \tag{1.3}$$

where $\top$ is the transpose operator. Since the ESN can be trained in closed form, it is very efficient compared with classical recurrent neural network training, which requires a time-consuming iterative process [49].

ESN belongs to a class of neural network architectures and training algorithms, such as back-propagation decorrelation (BPDC) [50] and extreme learning machines (ELM) [51], that attempt to reduce the training time of neural networks. In all these cases, the network architecture leverages a fixed hidden layer acting as a kernel projecting the inputs into a feature space that can be interpreted by the readout layer. In the case of ESN, the feature space is created with a large recursive kernel which creates an expressive spatiotemporal code for the input signal [52]. In ESN, to create the required spatiotemporal feature space, the reservoir must enjoy the so-called echo state property [44] (ESP): over time the asymptotic state of the reservoir only depends on the history of the input signal $\mathbf{u}(t)$ and not the initial reservoir state. Jaeger [44] showed that to satisfy this condition, the reservoir weight matrix Ψ must have the spectral radius λ^{max} and the largest singular values σ^{max} smaller than 1, a bound that was less conservative than the global stability analysis of the reservoir dynamics for all input signals [53]. Other approaches have proposed using self-organization techniques to adjust the poles of the reservoir, i.e., the eigenvalues of the reservoir weight matrix, to ensure optimal response of the reservoir to all frequencies and amplitudes of the input signal [40, 54–56]. However, it has been observed that these optimizations are task dependent and there are no reservoirs that are optimal for all tasks. As a result, in practical applications the reservoir parameters are usually adjusted for each task using a training and validation procedure [37, 48, 57–60].

The ESP conditions on reservoir weight matrix ensure that the dynamics of the reservoir is near the critical dynamical regime, also known as the edge of chaos. Many studies have pointed out the connection between the edge of chaos and the optimality of computation in network systems, such as the brain, genetic networks, artificial neural networks, and network automata models [61–70]. In the context of RC, Büsing et al. [71] studied the relationship between the reservoir and its performance and found that while in continuous reservoirs the performance of the system does not depend on the topology of the reservoir network, coarse-graining the reservoir state will make the dynamics and the performance of the system highly sensitive to its topology. Verstraeten et al. [72] used a novel method

to quantify the nonlinearity of the reservoir as a function of input weight magnitude. They used the ratio of the number of frequencies in the input to the number of frequencies in the dynamics of the input-driven reservoir as a proxy for the reservoir nonlinearity. In ESN, if the spectral radius of the reservoir is very small, $\lambda^{max} \ll 1$, distinctions in the state space of the reservoir converge very quickly and the reservoir cannot retain any memory of the past inputs. At the edge of chaos where $\lambda^{max} \approx 1$, the distinction will never converge, which means that the system always remembers the past inputs, but also remembers the initial states of the reservoir; this is not desirable. In a reservoir in the chaotic regime $\lambda^{max} > 1$, the distinctions in the state space diverge very rapidly and therefore the reservoir is hypersensitive to inputs and the initial states. In a system with the echo state property, the state space is slightly convergent and can retain enough memory of the inputs to be able to compute the desired output while being independent of the initial state of the reservoir. Figure 1.2 shows the dynamics of the ESN reservoir and the result of the computation in a reservoir with 100 nodes and tanh transfer function. The *activation* row shows the values of each node over time, the *reservoir nodes* row shows the time evolution of the reservoir nodes arranged in an arbitrary fixed spatial order, the *result* row shows the output and the expected output of the ESN, and the *squared error* row shows the calculated error in each time step given by $(y(t) - \widehat{y}(t))^2$. The system is trained to solve the order 10 NARMA task (see Section 2.2.4), a nonlinear autoregressive task (used in the neural net community to test neural net performance for capturing nonlinear long-term dependencies). The performance is evaluated by the normalized mean squared error (NMSE):

$$NMSE = \frac{\left\langle (y(t) - \widehat{y}(t))^2 \right\rangle}{\left\langle (\widehat{y}(t) - \langle \widehat{y}(t) \rangle)^2 \right\rangle}. \tag{1.4}$$

In the ordered regime, the convergent dynamics of the reservoir lead to very regular behavior in the node activations. This is visible in the reservoir nodes plots as the vertical lines where all the nodes synchronize and by the wide bands of horizontal lines showing groups of reservoir nodes having the same values over time. This homogeneity in the state space does not allow the readout layer to recover enough information from the reservoir to construct the target output, resulting in high error values, $NMSE = 0.76$. In the crit-

ical regime, the rapid divergence in the state space results in overexcitation of the node activations, which smears the information in the input signal over the state space in an unrecoverable way. This is reflected in the irregular waves in the reservoir nodes plot. This also results in high error $NMSE = 0.81$. Finally, in reservoirs with ESP, the nodes are moderately responsive to the input signal and the reservoir can retain enough information about the signal to compute the correct output. This is reflected in the complex micro-patterns in the reservoir nodes plot. Consequently the ESN with this reservoir has the lowest error $NMSE = 0.11$.

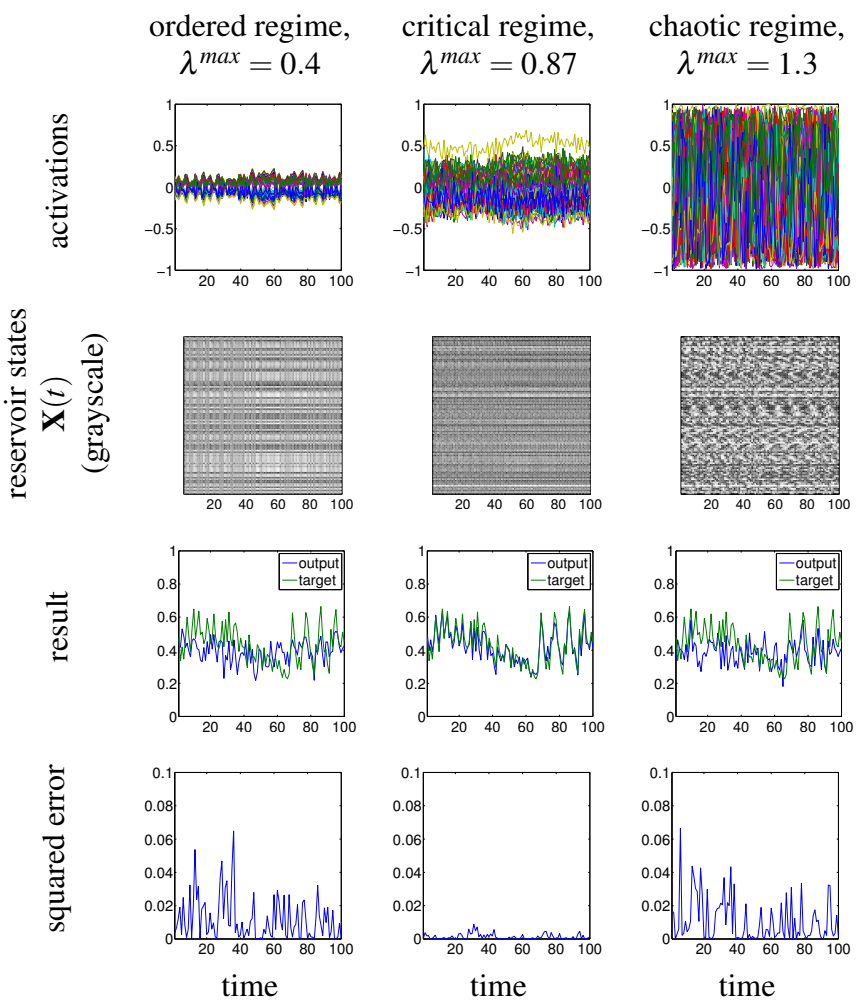


Figure 1.2: Illustration of signal processing using intrinsic dynamics. A randomly generated linear system is set in ordered, critical, and chaotic dynamical regimes and subjected to input stimuli. The *activations* row shows the values of the reservoir nodes over time, the *reservoir nodes* row shows the reservoir state vector $\mathbf{X}(t)$ in a fixed order presenting the spatiotemporal coding of the reservoir, the *result* row shows the ESN output and the target output, and *squared error* shows the squared error of the output at each time step. The reservoir is trained to solve the NARMA 10 task (see Section 2.2.4). The imprint of the input on the dynamics of the system creates a spatiotemporal code that can be used to recreate the output using a linear readout layer. In the ordered regime, the information is lost very quickly and cannot be used to compute the correct output. In the chaotic regime, the diverging dynamics scrambles the input to a point that it cannot be used to produce the output. In a reservoir with the echo state property, the information is preserved and it is recoverable by the readout layer. This reservoir is capable of optimal signal processing, which achieves minimum squared error.

Other ESN aspects that relate to my dissertation and have been previously studied in different contexts are noisy inputs, sparsity and structural properties of the reservoir connectivity, and continuity of the state space. Jaeger [39] studied ESN training in the context of time series processing and adaptive nonlinear system identification and found that using noisy teacher input/output increases the stability of the training. In this study, the readout layer is provided with the input, the reservoir states, and their squared value which greatly reduced the generalization error. It was postulated [73,74] that sparsity of the reservoir connectivity enhances the ESP, however the sparsity was loosely defined as having a zero mean in the reservoir weight matrix. Consequently, experiments by Jaeger [39, 45] were conducted by sampling the reservoir weights uniformly from the interval $[-1, 1]$. Jaeger [44] made the sparsity explicit by only choosing 10% of all the reservoir weights to be -0.47, another 10% to be 0.47, and the rest of the weights as zero. Furthermore, it was observed experimentally [37] that setting 50% of all the connections to have nonzero weights improved the results. Later [71] it was observed that in discrete state reservoirs the exact value of the sparsity significantly affects the performance while in continuous reservoirs the performance is not sensitive to the fraction of connections with zero weights. Inspired by models of brain morphology [73, 74], Song and Feng [75] studied ESN with reservoirs having complex connectivity structure and found that low clustering between the reservoir nodes improves performance. On the other hand, using the scale-free network growth model [76], Deng and Zhang [77, 78] showed that highly clustered nodes and short average path length between the reservoir nodes improve performance. Ganguli, Huh, and Simpolinsky [79] proposed an optimization for the input weights of linear dynamical systems to improve the performance of information processing. However, Rodan and Tiňo [48] did not find any improvement in reservoir performance using this optimization even when using linear transfer functions. The only analytical work on ESNs to date is the calculation of the memory curve for ESNs with orthogonal reservoirs [80]. This approach uses an annealed approximation, in which the reservoir weight matrix is assumed to be sampled from the Gaussian orthogonal ensemble (GOE) at each time step.

1.3.2 Learning and Generalization

By its nature, the material presented in this dissertation is deeply rooted in vast and deep disciplines of neural networks [9] and statistical learning theory [81]. Learning is a necessary process for a machine to solve a problem without being explicitly programmed for it. Valiant [82] was the first to formulate a theory of learning from examples in a computational framework with requirements on its computational complexity. However, the problem of learning had been first considered within the neuroscience community with the introduction of Rosenblatt's perceptron model and a discussion of its adaptive and learning properties [83], and later, in the mathematics community with Novikoff's convergence proof of the perceptron model [84]. As a simple example of a learning problem, let us consider the McCulloch-Pitts [85] model of neuron with N inputs $\mathbf{x} = (x_1, \ldots, x_n) \in X \subset R^n$ and one output $y \in \{-1, 1\}$. The input-output relation is defined as follows

$$y = \text{sgn}\{\mathbf{w}^\top \mathbf{x} - b\}, \tag{1.5}$$

where sgn is the transfer function, $\mathbf{w}^T\mathbf{x}$ is the inner product between the two vectors w and x, and b is the intercept. Geometrically, the neuron divides X into the region where $y = +1$ and the region where $y = -1$, with the boundary given by the hyperplane $\mathbf{w}^T\mathbf{x} - b = 0$. During learning, a learning algorithm should choose appropriate w and b to find the correct hyperplane that agrees with a given set of training examples $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^l, y^l)\}$. This is done by appropriately adjusting w and b after presentation of each example $(\mathbf{x}^i, y^i)$. Computational learning theory started with Novikoff's proof [84] showing that if

1. the norm of the training vectors x is bounded by some constant $|\mathbf{x}| < R$,
2. the training data can be separated with margin ρ,
3. and the training sequence is presented to the perceptron a sufficient number of times,

then after at most $N \leq \left\lceil \frac{R^2}{\rho^2} \right\rceil$ adjustments of w the optimal hyperplane for the training data can be constructed. Although Rosenblatt [83] had experimented with perceptrons orga-

nized in multiple layers, he did not know how to train all the network parameters at the same time. His solution was to fix all the parameters and only adjust the last layer that connects to the output. The solution to training all the weights in a multi-layer perceptron was found by Rumelhart, Hinton, and Williams [86] and is popular today under the name *error backpropagation*. As evident from the above example, the central goal in learning theory is to find algorithms that can efficiently deduce the functional relation between inputs and outputs from a finite set of examples in a way that generalizes to the rest of the input space. A good survey of major challenges, breakthroughs, and historical developments in learning theory can be found in [81].

1.3.3 Chaos Computing and Computational Power of Dynamical Systems

Theory of computation consists of analysis of various computation models and their computational power. Each of these models is effectively a discrete-state dynamical system with different state space size and memory. This naturally raises this question: "Is there a computer in the heart of every dynamical system?" This question has been analyzed in the context of various dynamical systems. This is done by carefully analyzing the generic structure of a given class of dynamical systems and building a mechanism that effectively simulates the operation of a classical computational model, such as a universal Turing machine, within the natural dynamics of the system [87]. For instance, an analysis of simple discrete dynamical systems called generalized shift registers revealed that they are effectively Turing universal [88, 89]. Koiran and Moore [90] showed that closed-form analytical maps in one and two dimensions are universal. Koiran et al. [91] discovered that even piece-wise linear maps in two dimensions are computationally universal. Branicky [92] analyzed computation in ordinary differential equations (ODE) and concluded that Lipschitz ODEs with as few as three dimensions are universal, while Bournez and Cosnard [93] proved that general analog continuous-time dynamical systems have super-

Turing ability because they use reals. Siegelmann [94] also argued that recurrent neural networks with real weights could provide a substrate for super-Turing computation. All of these efforts provide insight into the fundamental computational power of dynamical systems. However, for a particular desired computation, they do not prescribe how one should build a computer using said dynamical systems.

One of the first experimental examples of computation in dynamical systems was given in Ercsey-Ravasz and Toroczkai [95], which demonstrated a mapping between the $K-$SAT problem and differential equations (DE), in which the system effectively searches for a solution as it follows a trajectory to one of its attractors and eventually finds the right solution when it settles into an attractor. They found that although $K-$SAT is an NP-complete problem, their implementation always finds the right solution in polynomial time, but with exponential fluctuation in the system's energy. Murali et al. [96, 97] showed that coupled chaotic logistic maps may be programmed to implement arithmetic and logic operations. In this implementation the system uses initial conditions as an input and its dynamics is controlled via an external signal to ensure that the system will end in an attractor that encodes the desired result [98]. This technique, however, requires precise control of the chaotic dynamics of the coupled chaotic system, which might not always be easy or even possible. Alternative techniques have been proposed by other authors [99, 100] to use coupled complex-valued oscillators for implementing a variety of logic operations in a robust manner.

1.3.4 DNA Nanotechnology

The medical application of DNA nanotechnology, and in particular DNA-based long-term health monitoring and management, is an area that could benefit from a physical realization of RC. Here I give an overview of the field of DNA computing and nanotechnology.

Built on our ability to manipulate the information-carrying capacity of nucleic acid

molecules, DNA nanotechnology is a rapidly growing field that promises to deliver a wide array of biomechanical devices that can autonomously operate in a variety of biological environments [101–103]. DNA nanotechnology can be traced back to the discovery of sequence-specific recognition of DNA molecules using proteins by Ned Seeman [104], and his later pioneering work on construction of geometric shapes and mechanical devices using DNA [105–108].

In recent years, DNA has been established as a versatile programmable matter, owing to its universal information-processing capability [109] and rich controllable behavior [110]. The first analysis of computation power of DNA was done by Winfree [111], in which he proved that algorithmic self-assembly is computationally universal. Soloveichik and Winfree [112] analyzed the complexity of building a variety of shapes using the abstract self-assembly model. Several techniques for reducing the complexity of self-assembly and dealing with errors have been proposed [113–115]. DNA self-assembly has also been proposed to perform logic operation [116]. The first attempt to use DNA to solve hard optimization problems was made by Adleman [117]. This demonstrated the use of massive parallelism and plausibility of performing computation problems using biomolecules. However, Adleman's approach is not easily extensible because it requires monolithic design of the biomolecular reactions to solve a given problem. The largest problem solved using this approach has been the 20 variable 3-SAT problem [118]. Another approach has been to design simple logic gate motifs based on DNA strand displacement or DNAzymes [119, 120], and use them to build large-scale logic circuits [121–123].

One of the most intriguing applications of DNA nanotechnology is in medicine [124, 125]. For example, programmed drug delivery has been demonstrated using pH-sensitivity [126] and DNA-based logic controlled circuits [127]. It has also been demonstrated that DNA circuits can be designed to reprogram internal reaction pathways of a cell by targeting and changing their local environment through the release of appropriate molecules [127–129].

Another promising application of DNA is in long-term high-density information storage, owing to its small size factor, high fidelity, and robust capacity to store digital information [130]. DNA-based large-scale digital information storage with associative search and retrieval capability was first demonstrated in [131] with its postulated impact in medical data storage [132]. Advances in high-speed sequencing have enabled petabyte-scale DNA-based storage systems [133, 134]. Despite fast sequencing technologies, such massive storage may require *in vitro* parallel distributed information processing systems for efficient information retrieval and modification, which may be possible through advances in DNA computing [117, 118].

A different approach is to use DNA nanotechnology to implement an existing model of storage and information processing in distributed systems, such as the Hopfield neural network [135]. The Hopfield neural network is a popular model of large-scale distributed information processing and storage in physical systems. Although the implementation of these networks using chemical kinetics goes back to the early 1990s [136–138], a DNA implementation was proposed a decade later by introducing a generic scalable design for implementing matrix algebra using DNA strand displacement [139, 140]. This design was postulated to scale up to billions of neurons; however, a realizable Hopfield network has been demonstrated with only four neurons [141] using *in silico* calculations of the correct interactions between the neurons.

DNA and RNA circuits for higher-order tasks such as digital filters and Fourier transforms [142–145] and classifiers [146] have been proposed. Moreover, software packages for automated design of DNA circuits have been developed [147–149] and design of a DNA *proportional-integral* (PI) controller has been reported [150].

Neural network computation using DNA also has been demonstrated [141], but cur-

rent implementation lacks training within the DNA circuitry. However, theoretical designs for such adaptive circuits have been proposed [151, 152]. Although still at early stages, DNA nanotechnology could give us a new perspective in advanced medical diagnosis and treatments, such as long-term sensing, and potentially controlling, gene expression patterns over time in a cell. This would require appropriate sensors to detect cell state; for example, the pH-sensitive DNA nanomachine recently reported by Modi et al. [153] or other suitable functional DNA structures [154]. This may result in new methods for smart diagnosis and treatment using DNA signal translators [155–157].

1.3.5 Nano-Scale and Self-Assembled Electronics

In this dissertation, I frequently refer to the potential application of RC in physical computation and cite examples of such attempts. Here, I give a broader context for computing using unconventional and emerging technologies.

The quantum revolution in the twentieth century, and consequently, our increased understanding of fundamental properties of matter, led to a vision of building mechanical devices at the molecular and atomic scale. This is famously reflected in a statement made in a 1959 speech by the celebrated physicist Richard Feynman [158]: "There is plenty of room at the bottom." However, because of various technical difficulties, it has not been until the twenty-first century that we have learned how to enforce nanowires and carbon nanotubes into predefined structured patterns [159, and references therein]. Typically, nanowires and nanotubes are organized into a lattice structure or cross-bar, in which the intersections show semi-conducting behavior [19, 20]. Xiang et al. [160] demonstrated that transistors based on nanowires and carbon nanotubes easily outperform metal-oxide-semiconductor field-effect transistors (MOSFETs) with respect to energy consumption, fabrication density, and switching delay. For example, a very high-density dynamic random access memory (DRAM) at 10^{11} bits cm^{-2} was demonstrated in [161], in which a

160kB memory was fabricated the size of a white blood cell. Another example is atomic switch networks (ASNs) based on silver nanowires. These were based on a technology developed by Terabe et al. [162] aimed at reducing the cost and energy consumption of electronic devices. They can achieve a memory density of $2.5\,\mathrm{Gbit\,cm^{-2}}$ without any optimization, and a switching frequency of 1 MHz.

Recently, techniques have been developed to self-assemble nanowires into irregular network structures [21, 22]. Sillin et al. [12] combined bottom-up self-assembly and top-down patterning to self-assemble ASN. These networks are formed using deposition of silver on pre-patterned copper seeds. They have a three-dimensional structure that contains cross-bar-like junctions, and can be transformed into metal-insulator-metal (MIM) atomic switches in the presence of external bias voltage [12]. Nanoscale finite-state machines based on deterministic bottom-up assembly have also been demonstrated [163].

Major obstacles for using any of the above nanoscale and self-assembled architectures are design variations, irregular structure, defects, faults, and susceptibility to environmental factors such as thermal noise and radiation [23]. To overcome these challenges most approaches to date assume knowledge of the underlying architecture and rely on reconfiguration and redundancy to achieve programming and fault tolerance [164–168]. There have been two recent proposals on how to program such devices to perform classification or logic operations using a "black-box" approach [1,169]. In this model, called a randomly assembled computer (RAC), a network of interacting nodes with sparse and irregular connectivity is assumed. All nodes are initialized to zero and update their state according to a global clock, and each node calculates its next state using its transfer function and connections to other nodes. Three types of external signals (inputs, outputs, and controls) are connected to randomly chosen nodes. The task is to program the device to compute the desired output for a given input using a proper control signal. The optimal control signal can modify the propagation of input across the network such that the input is processed as required and the desired result will be ready at the output. The optimal control

values are computed using simulated annealing. A similar approach was also proposed by Tour et al. [2], but requires direct manipulation of microscopic structure of the system using genetic algorithms, which renders the approach impractical. A detailed study of nanoelectronics revealed that it is significantly more effective to rely on architectures with physically fault-tolerant devices than to achieve error correction [170]. Finding physically fault-tolerant phenomena for building computers is currently an active area of research.

1.3.6 Transfer Function Nonlinearity in Neural Networks

The theoretical core of this dissertation is based on the analysis of a model of RC with an arbitrary reservoir weight matrix and a linear transfer function. The exact analysis of RC with a nonlinear transfer function is mathematically prohibitive. However, we will use computational tools to investigate the degree of nonlinearity that helps boost the performance in RC compared with a linear model. In this section, we will provide broader context for the significance of such a study within the neuroscience and neural network literature.

McCulloch and Pitts [85] showed that the computational power of the brain can be understood and modeled at the level of a single neuron. Their simple model of the neuron consisted of linear integration of synaptic inputs followed by a threshold nonlinearity. Current understanding of neural information processing reveals that the role of a single neuron in processing input is much more complicated than a linear integration-and-threshold process [171]. In fact, the morphology and physiology of the synapses and dendrites create important nonlinear effects on the spatial and temporal integration of synaptic input into a single membrane potential [172]. Moreover, dendritic input integration in certain neurons may adaptively switch between supralinear and sublinear regimes [173]. From a theoretical standpoint this nonlinear integration is directly responsible for the ability of neurons to classify linearly inseparable patterns [174]. The advantage of nonlinear processing at

the level of a single neuron has also been discussed in the artificial neural network (ANN) community [175].

1.3.7 Neural Networks with Product Nodes

Product nodes in neural networks emerged with the study of nonlinear presynaptic integration in biological neurons. In particular, multiplicative presynaptic integration has been observed in several types of neurons, but has not been fully investigated in the context of recurrent networks and RC.

In a node with a nonlinear transfer function, the inputs are linearly combined and passed through a saturating nonlinearity. Another model that achieves nonlinear computation in a node is nonlinear integration of inputs before applying the transfer function. Product nodes are a special case of this model, in which the inputs are raised to the power of the synaptic weight and then multiplied together. The use of product nodes in neural networks was introduced in [176] in an effort to learn suitable high-order statistics for a given task. It has been reported that most synaptic interactions are multiplicative [177]. Examples of such multiplicative scaling in the visual cortex include gaze-dependent input modulation in parietal neurons [178], modulation of neuronal response by attention in the V4 area [177] and the MT area [179]. In addition, locust visual collision avoidance mediated by LGMD neurons [180], optomotor control in flies [181, 182], and barn owl's auditory localization in inferior colliculus (ICx) neurons can only be explained with multiplicative interactions [183].

Another popular architecture which uses product nodes is the ridge polynomial network [184]. In this architecture the learning algorithm iteratively adds groups of product nodes with integer weights to the network to compute polynomial functions of the inputs. This process continues until a desired error level is reached. The advantage of the product node with variable exponent over the ones used in polynomial networks is that instead of

providing fixed integer power of inputs, the network can learn the individual exponents that can produce the required pattern [175].

1.4 Significance of This Dissertation

Three major disciplines that benefit from advances in RC are neuroscience, machine learning, and unconventional computing. In neuroscience, RC represents one of the best models today to explain short-term memory in the prefrontal cortex; in machine learning, RC is a computationally efficient way to train a recurrent neural network for accurate chaotic prediction; and in unconventional computing, RC is an approach to compute using a system's intrinsic dynamics. In the remainder of this section, I will elaborate on the connection between RC and each of the said disciplines and how the content of this dissertation contributes to it.

1.4.1 RC in Neuroscience

In theoretical neuroscience, short-term memory is the central feature in RC. Using annealed approximation, linear random orthogonal networks driven by uncorrelated inputs have been found to hold *extensive memory*, wherein the capacity of the network to reconstruct τ prior inputs scales linearly with the system size N [80]. The same study also revealed (through computational tools) that non-orthogonal networks do not have extensive memory. Later in [79], generic recurrent networks with saturating nonlinearity were analyzed using mean-field approximation and their memory was bounded by $\sqrt{N}$. More recently, Toyoizumi [185] showed that networks with uniform weights with saturating nonlinearity tuned to have attractor dynamics can have nearly extensive memory with a scaling regime of $N/\log N$. Moreover, super-linear memory has only been observed in orthogonal networks with sparse input under compressed sensing setup [186]. Notwith-

standing the search for better memory scaling, an exact evaluation of the memory of a generic network under a generic input signal has been lacking in the theoretical RC literature. This dissertation fills this gap. Current theories either focus on a restricted class of systems, or use an annealed approximation, which ignores the specific structure of a particular network, or provide an exact solution for a uniformly weighted network. In addition, they do not consider how the memory changes if input is correlated. The theory introduced in this dissertation addresses this shortcoming.

1.4.2 RC in Machine Learning

In machine learning, RC is used as a computationally efficient method to train recurrent neural networks to solve temporal problems, such as chaotic prediction [5]. Since the premise of RC is that the specifics of the underlying network do not affect the RC's computation significantly, many have tried to use different classes of networks, such as small-world and scale-free networks, to improve the performance of RC on task solving [37, 39, 45, 48, 71, 75–78]. Choosing the right class of networks as well as adjusting the spectral properties of the network and the input coefficients are hyper parameter optimizations that have to take place for each dataset and task. Another issue is that although the power of RC is attributed to its short-term memory, there is no direct relationship between the short-term memory of the system and its performance in a given task. The theoretical framework of this dissertation extends the evaluation of memory capacity to an expected mean squared error (MSE) and worst-case error bound for a given task based on the spectral properties of the network, the input, and the expected output. As a result, one can determine exactly how a microscopic and macroscopic change in the structure of the network, along with changes to task properties, influence the performance of the system.

1.4.3 RC and Other Recurrent Networks

Several neural network architectures exist for temporal pattern discovery and time series processing, such as autoregressive networks, time-delayed networks, simple recurrent neural networks [9]. More recently, long short-term memory network (LSTM) [187] and gated recurrent neural network [188] have been proposed to learn arbitrary long-term dependencies. Here we briefly compare and contrast these networks.

One of the earliest models for neural-network-based time series processing is the autoregressive neural network, or, closely related to it, time-delay networks [9]. In a time-delay network, input is passed through a tapped delay line which is connected to the hidden layer of the network. In a an autoregressive network, the output of the network is also passed through a tapped delay line and fed to the hidden layer. The hidden layer combines information from the input and output history which is used to generate the output. The problem with this architecture is that the length of the delay line is fixed in advance, which means the hidden layer can only access a predefined number of previous outputs and inputs. This is called fixed memory depth. Simple recurrent networks, or Ellman's networks, fix this problem by considering a fully connected recurrent network driven by input [9]. These networks do not have the problem of fixed memory depth but their training is very time-consuming and unstable due to the vanishing and exploding gradient problem [189]. A new Hessian-free optimization has been developed recently that partially alleviates this problem; however, this algorithm is still computationally expensive and depending on the training dataset it may not converge while learning arbitrarily long time dependencies [190].

Long short-term memory network [187] and its variant, gated recurrent neural network [188], are specialized network architectures with a generic feedforward structure and a recurrent circuit (called memory cell) that is designed to trap network activations inside it for arbitrarily long time and provide it to downstream nodes for further processing. These

architectures are much more robust to the vanishing gradient problem but one has to adjust the number of memory cells based on the needs of a particular application. To address this problem neural Turing machines [191] have been proposed which are networks augmented with virtually infinite memory and a specialized training algorithm to teach the network how to access the memory.

All of the mentioned architectures require modification to the microscopic structure of the system during the training, which is not only computationally expensive, but also infeasible if a physical realizations of these systems were possible in atomic scale, which is a general issue with nanoscale self-assembled devices. While RC does not have the generality of these systems due to restricted training, it does provide a way of using transient states of a dynamical system as a memory without modifying its microscopic structure. This feature makes RC attractive for physical implementations using emerging technologies.

1.4.4 RC in Unconventional Computing

Unconventional computing is concerned with building computers from alternative architectures to von Neuman's. Recent advances in material science and nanotechnology have infused unconventional computing with a new wave of possible substrates to work with, such as self-assembled nanowires, memristors, and DNA. One challenge in using these new substrates is that structures built from them are hard or impossible to control precisely and even harder to reconfigure after initial construction [12, 192]. This controllability is essential to encode, maintain, and manipulate information in these systems to achieve computation. Achieving controllability in systems where many unreliable parts interact to achieve computation is central to unconventional computing research.

Neural networks as a model of distributed computing have long been a subject of research in unconventional computing community [3, 135], and RC in particular provides

an additional advantage that the microstructure of the system does not need to be modified. Therefore, understanding how the system can be tuned collectively to give suitable macroscopic properties for computation, and how deviation from this can hurt performance is of great interest in RC and its application to unconventional computing. The framework introduced in this dissertation makes such an investigation possible in a precise and mathematical way. Although our focus on linear systems is limiting, since many possible physical implementations of RC behave nonlinearly [12, 193, 194], our framework is still relevant for analyzing photonic implementation of RC, where propagation of short coherence-length light through the reservoir medium is known to be governed by linear operators [195]. In principle, in such a photonic reservoir, given the linear operator governing the reservoir and the task specification, one could analytically compute the optimal readout layer without training. Finally, whereas our original design for DNA-based RC would have required expensive microfluidic setup and circuit design to work properly [193], here we introduce a new design that solves many of the challenges and could potentially be realized using buffered strand displacement techniques [196, 197].

1.4.5 Contributions

In this dissertation, I develop a mathematical framework for analytical investigation of computational properties of linear ESNs. Using this framework I provide the first exact solution for the memory capacity of ESNs under both uncorrelated and correlated input. I give the first exact solution to expected error for a given task and a worst-case error bound. I show that it is possible to analytically reason about the memory and information processing performance of a linear ESN based on the input-output statistics and the spectrum of the ESN connection matrix.

Using the mathematical framework for linear ESNs, I formulate a way to reason about nonlinear ESNs. I investigate two types of nonlinearity: (1) ESN with nonlinear transfer

function and (2) ESN with linear transfer function and nonlinear presynaptic integration. I show that the recurrent dynamics of the ESN generate higher-order correlation information from the smallest possible nonlinearity that significantly enhances the performance of the ESN. In addition, I prove the stability of ESN with product nodes and show their application in chaotic prediction.

I investigate a possible design for DNA reservoir computing and computationally show that it is capable of predicting glucose concentration. This design simplifies previous attempts in DNA reservoir computing by eliminating the need for a timed microfluidic system for DNA-based oscillators as the reservoir.

Section 1.4.6 lists the publications that are directly or indirectly based on the material presented in this dissertation.

1.4.6 Publications

The publication list in this section represents the body of work that I contributed to during the course of my Ph.D. The work has been done through various collaborations. I have limited this list to the papers that have been published or are close to publication.

1. A. Goudarzi, M. R. Lakin, and D. Stefanovic, *DNA reservoir computing: A novel molecular computing approach*, in DNA Computing and Molecular Programming, (D. Soloveichik and B. Yurke, eds.), Vol. 8141 of *Lecture Notes in Computer Science*, Springer International Publishing, 2013, pp. 76–89

2. D. Snyder, A. Goudarzi, and C. Teuscher, *Computational capabilities of random automata networks for reservoir computing*, Physical Review E **87** (2013), 042808

3. A. Goudarzi, M. Lakin, and D. Stefanovic, *Reservoir computing approach to robust computation using unreliable nanoscale networks*, in Unconventional Computation

and Natural Computation, (O. H. Ibarra, L. Kari, and S. Kopecki, eds.), Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 164–176

4. A. Goudarzi and D. Stefanovic, *Towards a calculus of echo state networks*, Procedia Computer Science **41** (2014), 176–181

5. A. Goudarzi, M. R. Lakin, D. Stefanovic, and C. Teuscher, *A model for variation- and fault-tolerant digital logic using self-assembled nanowire architectures*, in Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH) IEEE Press, 2014, pp. 116–121

6. J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, *Computational capacity and energy consumption of complex resistive switch networks*, AIMS Materials Science **2** (2015), 530–545

7. J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, *Hierarchical composition of memristive networks for real-time computing*, in Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on, Boston, Ma, July 8-10 2015, pp. 33–38

8. A. Goudarzi, A. Shabani, and D. Stefanovic, *Exploring transfer function nonlinearity in echo state networks*, in Computational Intelligence for Security and Defense Applications (CISDA), 2015 IEEE Symposium on, May 2015, pp. 1–8

9. A. Goudarzi, A. Shabani, and D. Stefanovic, *Product reservoir computing: Time-series computation with multiplicative neurons*, in 2015 International Joint Conference on Neural Networks (IJCNN), 2015

10. A. Goudarzi, D. Arnold, D. Stefanovic, K. B. Ferreira, and G. Feldman, *A principled approach to HPC event monitoring*, in Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale, FTXS '15, ACM, New York, NY, USA, 2015, pp. 3–10

11. M. Stella, C. S. Andreazzi, S. Selakovic, A. Goudarzi, and A. Antonioni, *Parasite spreading in spatial ecological multiplex networks*, arXiv:1602.06785 **[q-bio.QM]** (2016), Under review

12. A. Goudarzi, S. Marzen, P. Banda, G. Feldman, M. R. Lakin, C. Teuscher, and D. Stefanovic, *Memory and information processing in recurrent neural networks*, arXiv:1604.06929v1 **[cs.NE]** (2016), Under revision

Chapter 2

Benchmarks for Experimental Evaluation of RC Memory Capacity and Computational Ability

RC is used for blackbox modeling of dynamical processes with memory. Hence, theoretical studies of RC focus on understanding its computational abilities in context-dependent computation, where we are interested in computing some function of an input sequence $u(t)$, i.e., $y(t) = f(u(t), u(t-1), \ldots, u(t-\tau))$. Therefore, a conceptual distinction is made in theoretical studies of reservoir computing between the complexity of f and its required memory depth τ [206].

In this chapter we first motivate what we are interested in studying (Section 2.1) and then review some of the common benchmarks commonly used to assess RC's performance (Section 2.1.1) that have been adopted in the RC literature because of their ability to test different properties of RC, such as memory, nonlinear processing capacity, trajectory prediction, and regressive computation. We will then motivate our choice of benchmarks (Section 2.1.2) and describe their details and what they will measure, and we will evaluate

the performance of the system on these benchmarks (Sections 2.2.1).

2.1 Methodology

The central objective in time series analysis is to discover how the current value or the future of the time series can be causally described in terms of its history and how such information can be represented in an efficient way [207–210]. Time series analysis techniques in one way or another seek to discover dependencies between future and the past of a sequence of data points that constitute the time series. In the neural network literature, it has been known that discovering long-term dependencies is especially hard [211]. Therefore, our benchmarks for evaluating different architectures and training algorithms for neural network time series processing should consist of a series of tasks to test if the architecture and the algorithm can discover and represent long-term dependencies and the complexity of the function it can compute based on such dependencies [43,49]. Below we will discuss some of the commonly used benchmarks in neural-network-based time series processing.

2.1.1 Common Benchmarks

In [49], the authors summarize some of the landmark achievements in recurrent neural networks in 1990's and propose a few benchmark tasks to test the state of the art algorithms. In particular, at the time the universal nonlinear approximation properties of neural networks had been understood well and non-trivial questions about recurrent neural networks were focused on learning long-term dependencies. Therefore, all of the proposed benchmark tasks are synthetic nonlinear autoregressive systems that generate time series with long-term dependencies that span from two to 10 time steps ago. The input to these systems is usually a white noise signal and the output is stochastic as well. Of

these tasks, NARMA 10 has emerged as a standard benchmark for testing if a network can learn long-term dependencies. In [5] and [43], the authors used a series of tasks based on prediction and generation of continuous chaotic signals. Unlike previous autoregressive tasks, these chaotic signals have deterministic dynamics and time dependencies that far exceed 10 time steps. In [190] a Hessian-free optimization method for training recurrent neural network was introduced. The resulting network was compared with long short-term memory (LSTM) [43, 187] networks on a series of pathological examples to specifically test for discovering long-term dependencies. Jaeger [212] used the same set of problems as in [43] to show that RC may solve them as well. These problems included producing addition and multiplication of two numbers in a sequence, memorizing long bit streams, and solving XOR on binary sequences. In [14], we showed that it is even possible to decompose a single nonlinear Boolean function such as XOR into a circuit with a number of linear functions such as NAND and have the reservoir compute the final value by including a feedback to the reservoir from the output. In addition to the above tasks, memory capacity and nonlinear memory capacity tasks have been introduced specifically in the RC community to directly test for the memory depth and nonlinear computation in the reservoir [80, 213]. Below, we motivate our choice of tasks for evaluating the performance of RC in this dissertation.

2.1.2 The Choice of Benchmark

Broadly speaking, the hypothesis behind the working of RC is that the recurrent dynamics of the reservoir creates a short-term memory, which encodes the input sequence into the reservoir. This short-term memory is distributed (it can only be recovered by reading all the nodes and not just a single node), is instantaneous (the input history is encoded in the state of the reservoir in a single time step), and independent of the initial state of the reservoir. The readout layer than interprets the states of the reservoir to produce a desired output.

In my analysis, I focus on memory and information processing properties of RC and how they are influenced by the choice of the reservoir. One of the fundamental approaches to analyze RC is to directly measure the total number of recent inputs it can reproduce with reasonable accuracy (Section 2.2). I will take the same approach to start my analysis and develop a framework for exact evaluation of total memory capacity in the network. I will then extend this framework for cases in which we desire to output a value that is a non-identity function of the input history. Any input dependent function shall suffice for this analysis, however we would like this function to have long-term memory temporal dependencies to really leverage the memory capacity of the RC. Our choice here is the NARMA10 benchmark (Section 2.2.4), which is a discrete-time stochastic system where the output at time t is nonlinearly dependent on previous input and output from $t-10$. For the second choice of task solving, I picked the prediction of Mackey-Glass system (Section 2.2.3). This is a deterministic continuous-time chaotic system with non-vanishing temporal dependencies and is used for assessing the ability of neural networks to discover nontrivial long-term dependencies. We will explain the details of these tasks below.

2.2 Tasks

2.2.1 Memory Capacity with Uncorrelated Input

The memory capacity in RC is defined as the total number of recent inputs that can be reconstructed from the instantaneous state of the reservoir $x(t)$ with reasonable accuracy [80]. To calculate the memory capacity the reservoir is driven with an input sequence $u(t)$ drawn from i.i.d. uniform distributions with zero mean $\langle u(t)\rangle = 0$ and variance $\langle u^2(t)\rangle$. The desired output for this task is defined as:

$$\widehat{y}_\tau(t) = u(t-\tau), \tag{2.1}$$

where τ is the lag for the memory task. The quality of reconstruction is measured by the coefficient of determination between the reconstructed output $y(t)$ and expected output $\widehat{y}(t)$ as follows [45, 80]:

$$m(\tau) = \frac{\mathrm{Cov}^2(y(t), \widehat{y}_\tau(t))}{\mathrm{Var}(y(t))\mathrm{Var}(\widehat{y}_\tau(t))}, \tag{2.2}$$

The capacity function $m(\tau)$ is a curve that at each point is between 0 (for no reconstruction) and 1 for perfect reconstruction. To make the relationship between $m(\tau)$ and number of recovered inputs concrete, one defines a threshold $m(\tau) > k$ (usually $k = 0.5$) to count the input at $u(t-\tau)$ as recovered. The summation $\sum_\tau m(\tau)$ gives the total memory capacity in terms of the number of recent inputs recoverable.

2.2.2 Memory Capacity with Correlated Input

Memory capacity of RC is usually analyzed under uncorrelated input [79, 80, 185]. However, it has been reported that existence of structure in the input can greatly enhance the memory capacity [186]. In this dissertation, I extend my framework to the calculation of memory capacity under correlated input and investigate its effects. To this end, I pick the exponential autocorrelation function, $R(\tau) = e^{-\alpha\tau}$ with decay rate α. Exponential autocorrelation arises from linear systems and short-term memory processes [214] and has been observed in some natural signals [215–217]. It constitutes an autocorrelation model with nontrivial correlation structure which is simple enough to analyze exactly. The output is defined as before by Equation 2.1 and the memory capacity is measured using Equation 2.2.

2.2.3 Mackey-Glass Chaotic Trajectory Prediction

The Mackey-Glass system [218] was first proposed as a model for feedback systems that may show different dynamical regimes. The system is a one-dimensional delayed feedback

differential equation and manifests a wide range of dynamics, from fixed points to strange attractors with varying divergence rates (Lyapunov exponent). This system has been used as a benchmark task for chaotic signal prediction and generation [5]. It is defined as:

$$\frac{dx(t)}{dt} = \beta \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t), \tag{2.3}$$

where $\beta = 2, \gamma = 1, n = 9.7451, \tau = 2$ ensure the chaoticity of the dynamics [5]. the performance is evaluated by calculating the mean-squared error MSE as follows:

$$MSE = \frac{1}{T}\sum_{t=0}^{T}(y_t - \widehat{y_t})^2 \tag{2.4}$$

where y_t is the network output and $\widehat{y_t}$ is the desired output, and T is the total length of the output.

2.2.4 Nonlinear Autoregressive System Computation

Nonlinear autoregressive moving average of order 10 (NARMA10) is a classic autoregressive task commonly used in benchmarking recurrent neural networks because of its nonlinearity and long-term temporal dependencies. The task is defined by the following system:

$$y_t = \alpha y_{t-1} + \beta y_{t-1}\sum_{i=1}^{n} y_{t-i} + \gamma u_{t-n}u_{t-1} + \delta, \tag{2.5}$$

where $n = 10$, $\alpha = 0.3, \beta = 0.05, \gamma = 1.5, \delta = 0.1$. The input u_t is drawn from a uniform distribution over the interval $[0, 0.5]$. The performance is evaluated using Equation 2.4.

Chapter 3

Memory and Task Solving in ESN, An Analytical View

Since the introduction of reservoir computing [5,24] to leverage short-term memory in the transient dynamics of recurrent networks for computation, four major theoretical works have analyzed short-term memory properties in recurrent networks [79,80,185]. This was done by defining a memory function $m(\tau)$ to measure the ability of the system to reconstruct input from τ time steps ago, i.e., $u(t-\tau)$, from the present system state $\mathbf{x}(t)$. The integral of this function, i.e., $\sum_\tau m(\tau)$, is called the memory capacity. The short-term memory of orthogonal networks with linear transfer function was analyzed under uncorrelated input signal $u(t)$ and annealed approximation [80]. A regime of extensive memory was identified in these networks, i.e., the regime where the total memory scales with the size of the network $O(N)$. The authors reported discrepancies between analytical and numerical results because the annealed approximation cannot account for the correlation between higher powers of connection matrices. In numerical studies, they also observed a lack of an extensive memory regime for non-orthogonal networks. The extensive memory regime was rediscovered in [79] by calculating how much Fisher information the system states $\mathbf{x}(t)$ retain from the inputs $u(t-\tau)$. In addition, the authors showed that saturating nonlin-

earity disrupts the extensive memory regime and that the memory capacity scales with $\sqrt{N}$. In [186], the memory capacity of linear orthogonal networks was shown to exceed $O(N)$ for sparse input sequences. Later, it was shown that in networks with identical weights and saturating transfer function, with enough nonlinearity to ensure attractive states, an error correction property emerges wherein the decaying past values are corrected due to the existence of attractors. In such a network, the memory capacity scales with $N/\log(N)$ [185].

In this chapter, I review the core results of this dissertation. My approach to analyzing the memory and information processing in the reservoir is to derive an algebraic expression for the linear regression equation of the readout layer (Equation 1.3). My focus is not on finding the best architecture or the memory scaling regime, but on deriving an expression for exact evaluation of network performance for a given task. The expression will enable determination of task performance based on the properties of the given network and the task itself, viz., network and signal spectra. This approach will bridge the existing gap between task performance and memory capacity in RC and allow one to ask precise questions about how the network performance for a given task depends on the specifics of a given network structure. While the linear transfer function assumption is a limitation of our analysis, advances in understanding the information processing in linear systems are nevertheless relevant, even for physical implementations of RC systems, e.g., photonic reservoir with short coherence-length light source and linear-rectifier feedback [195]. I understand this approach is in contrast from the explanation of computation in RC through interpretation or simulation arguments that is usual in theoretical computer science [94, 219–221]. Therefore, I dedicate Section 3.1 to justify this approach. I then introduce the general structure of my mathematical framework (Section 3.2) and proceed to apply the framework to analytically evaluate the reservoir on the memory and signal processing benchmark tasks (Sections 3.3, 3.4, and 3.5). I end the chapter with an analysis of the effect of the structure of the reservoir on the benchmark results (Section 3.6), followed by a discussion of the results (Section 3.8).

3.1 The Choice of Analysis

Siegleman and Sontag [220] showed for the first time that the class of fully connected recurrent neural networks with a sigmoid activation function is a universal Turing machine. Their proof is based on a simulation argument in which they designed the structure of the network to simulate a counter machine, which is equivalent to a Turing machine. This type of simulation argument is commonly used in theoretical computer science to show the expressive power of a mathematical structure [222]. However, such arguments do not consider how the structure is used by a training algorithm to achieve some function. On the other hand, neural networks and other learning machines are used for their ability to learn and generalize, a behavior that emerges from the operation of a learning algorithm on the network architecture [9]. It is often difficult or impossible to analyze the operation of a learning algorithm exactly. However, it is possible to construct a series of hypotheses from the computer science perspective and test them against the behavior of a certain trained neural network. Yet, such arguments give us at best an interpretation of what a learning machine is doing, and in the case of RC will result in interpretations that may be confusing. For example, the readout layer of RC can be trained to compute the XOR (parity) function over a sequence of input bits (Section 2.1.1). It is tempting to conceptualize the XOR operation in this case as a modulus 2 of summation of a sequence of bits and therefore the reservoir is counting the bits in its nonlinear dynamics. This interpretation would mean that since the only adaptive part of RC is the output layer, the network is able to count the number of zero and one bits in a given interval of the input sequence. However, this explanation does not work because the same reservoir can be used to solve the XOR task over different lengths of input with any time lag (given a large network). In fact one can change the task and train the output to compute any other function of the input using the same reservoir. Then would that mean that the reservoir is computing all possible functions over the input history and the readout layer is only picking out one of them?

Here I ask a different question: since the output is generated by the readout layer from the reservoir states, why don't we analyze what the readout layer does? In this dissertation, we aim to give a mathematical description of what exactly the reservoir looks like to a linear readout layer.

3.2 Core Results

Here we describe the general setup for our core results. Derivations for 1-dimensional and N-dimensional systems as well as validation of calculations are given in Appendix A.

Consider a discrete-time network of N nodes. The network weight matrix Ω is a full-rank real $N \times N$ matrix with spectral radius $\lambda < 1$. A time-dependent scalar input signal u_t is fed to the network using the input weight vector ω drawn from some distribution, e.g., a Gaussian or uniform distribution [35, 37, 206]. The evolution of the network state $\mathbf{x}_t$ and the output y_t is governed by

$$\mathbf{x}_{t+1} = \Omega \mathbf{x}_t + \omega u_t, \text{ and} \tag{3.1}$$

$$y_{t+1} = \Psi \mathbf{x}_{t+1}, \tag{3.2}$$

where

$$\Psi = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1} \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}} \tag{3.3}$$

is an N-dimensional column vector calculated for a desired output $\widehat{y}_t$. Without loss of generality we are restricting ourselves to scalar input and output.

Here $\mathbf{X} = [x_{it}] = [\mathbf{x}(0)|\ldots|\mathbf{x}(T)]$ represent the system states in time. We denote the desired output by $\widehat{\mathbf{Y}} = [y(0)|\ldots|y(T)]$, and T is the total length of input time series. In practice it is sometimes necessary to reguralize the readout weights:

$$\Psi = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \gamma^2 \mathbf{I}\right)^{-1} \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}}, \tag{3.4}$$

where γ is a regularization factor that needs to be adjusted depending on Ω, ω, and u_t [206]. Calculating Ψ for a given problem requires the following input-output-dependent evaluations (Section A.4):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \sum_{i,j=0}^{\infty} \Omega^i \omega R_{uu}(i-j)\omega^{\mathrm{T}}(\Omega^{\mathrm{T}})^j, \text{ and} \tag{3.5}$$

$$\mathbf{X}\mathbf{Y}^{\mathrm{T}} = \sum_{i=0}^{\infty} \Omega^i \omega R_{u\widehat{y}}(i), \tag{3.6}$$

where $R_{uu}(i-j) = \langle u_t u_{t-(i-j)} \rangle$ and $R_{u\widehat{y}}(i-j) = \langle u_t \widehat{y}_{t-(i-j)} \rangle$ are the autocorrelation of the input and the cross-correlation of the input and target output. This may also be expressed in terms of the power spectrum of the input and the target [205], but the two derivations are dual with respect to the Fourier transform, therefore I will proceed with the representation introduced here.

The performance can be evaluated by the mean-squared-error (MSE) as follows:

$$\langle E^2 \rangle = \left\langle (\widehat{y}(t) - y(t))^2 \right\rangle = \widehat{\mathbf{Y}}\widehat{\mathbf{Y}}^{\mathrm{T}} - \widehat{\mathbf{Y}}\mathbf{Y} \tag{3.7}$$

$$= \widehat{\mathbf{Y}}\widehat{\mathbf{Y}}^{\mathrm{T}} - \widehat{\mathbf{Y}}\mathbf{X}^{\mathrm{T}} \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1} \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}}. \tag{3.8}$$

The MSE gives us an upper bound on the instantaneous squared-error through the application of Markov inequality [223]:

$$P\left[(\widehat{y}(t) - y(t))^2 \geq a\right] \leq \frac{\langle E^2 \rangle}{a}. \tag{3.9}$$

This upper bound is independent of the distribution of $u(t)$. Intuitively, this means that the probability of error at time t being larger than a constant a is always less than or equal to the average error divided by a. The existence of a worst-case bound is important for engineering applications of RC.

3.3 Memory Capacity for Correlated Inputs

In Appendix A I exhibit a mathematical framework for analyzing the memory of ESN with linear activation function and arbitrary weight matrix Ω and input vector ω. I also validated the theory by comparing both the raw values and memory function with several system sizes and spectral radii for uncorrelated input. Here we focus on a minimal ESN model that consists of a uniform ring with all identical weights λ. This is a circulant matrix with spectral radius λ. This model was first introduced in [48] to investigate the role of randomness in ESN, and it was reported that this simple deterministically instantiated network achieves similar results to a random reservoir. Furthermore, this model lets us control the reservoir using just two parameters: the number of nodes N and the spectral radius λ. In Section 3.6 we will systematically deviate from this regular structure to analyze the effect of structure on the performance of the reservoir.

Here, I will test our framework on this regular ESN to calculate memory capacity under correlated input and later (Section 3.4 and 3.5) I will apply it to compute the expected error of the system on nonlinear autoregressive and chaotic prediction problems. I compute the memory function and the total memory of the ESN described in Section A.4 for exponentially correlated input where $R_{uu}(\tau) = e^{-\alpha\tau}$. For justification of memory function and the exponentially correlated input refer to Section 2.2.1. The total memory of the system is given by the following summation over the memory function [80]:

$$\sum_{\tau} m(\tau) = \sum_{\tau} (\mathbf{Y}\mathbf{X}^{\mathrm{T}})_{\tau} (\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1} (\mathbf{X}\mathbf{Y}^{\mathrm{T}})_{\tau}, \tag{3.10}$$

where $\mathbf{Y}$ is the input with lag τ, $u_{t-\tau}$. The term $(\mathbf{Y}\mathbf{X}^{\mathrm{T}})_{\tau}(\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1}(\mathbf{X}\mathbf{Y}^{\mathrm{T}})_{\tau}$ in this summation is equivalent to Equation 2.2 written using the components we have computed and calculates the total number of most recent inputs to the reservoir that is recoverable from the reservoir state by an optimal linear readout layer. The notation $(\mathbf{Y}\mathbf{X}^{\mathrm{T}})_{\tau}$ means the desired output is input from τ time steps ago.

Computing $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ requires the evaluation of:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \sum_{i,j=0}^{\infty} \Omega^i \omega R_{uu}(i-j)\omega^{\mathrm{T}}(\Omega^{\mathrm{T}})^j. \tag{3.11}$$

By applying some linear algebra (Appendix A.5) we can show that:

$$\begin{aligned}\mathbf{X}\mathbf{X}^{\mathrm{T}} &= \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}(\mathbf{I} - e^{-\alpha}\Omega^{\mathrm{T}})^{-1} \\ &+ (\mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}(\mathbf{I} - e^{-\alpha}\Omega^{\mathrm{T}})^{-1})^{\mathrm{T}} \\ &- \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}.\end{aligned} \tag{3.12}$$

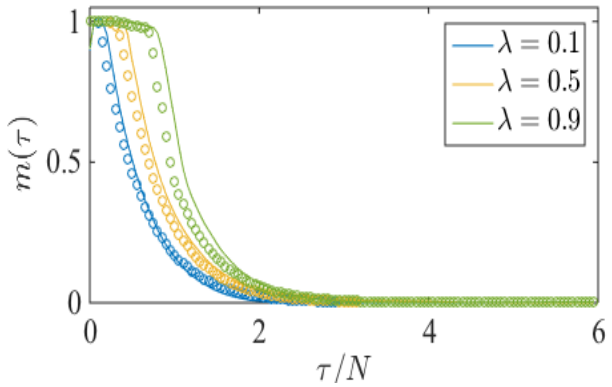
Here, the columns of $\mathbf{U}$ are the eigenvectors of Ω, $\mathbf{d} = (d_i)$ is the column vector containing the eigenvalues of Ω, $\Lambda = \mathbf{U}^{-1}\omega\omega^{\mathrm{T}}\mathbf{U}^{-1^{\mathrm{T}}}$, and $\mathbf{B}$ is an $N \times N$ matrix whose elements are given by $B_{ij} = \Lambda_{ij}\left(1 - d_i d_j\right)^{-1}$. The first and second term in this equation calculate the contributions for elements of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$, where $i \geq j$ and $i \leq j$, respectively and the last term corrects for the double counting of the diagonal elements, where $i = j$.

The covariance of the network states and the expected output is given by:

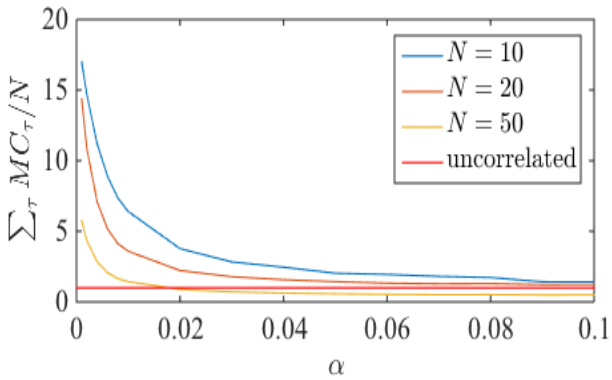
$$\mathbf{X}\mathbf{Y}_{\tau}^{\mathrm{T}} = \sum_i \Omega^i \omega R(|i-\tau|) = \sum_i \Omega^i \omega e^{-\alpha|i-\tau|}. \tag{3.13}$$

For $\alpha \to \infty$, the signal becomes i.i.d. and the calculations converge to the formula developed in Section A.2.

To validate our calculations, we use a network of $N = 20$ nodes in a ring topology and identical weights. The spectral radius $\lambda = 0.9$. The input weights ω are created by sampling the binomial distribution and multiplying with 0.1. The magnitude of the input weights does not affect the memory and the performance in linear systems and therefore we adopt this convention for generating ω throughout the paper. Note that there is only a single uniform ring with a fixed N. This choice constrains the effect of variability due to structure. The effect of structural variability will be discussed in Section 3.6. We also assumed $\alpha = 0.05$, the number of samples $T = 30{,}000$, washout period of $5{,}000$ steps, and regularization factor $\gamma^2 = 10^{-9}$. To generate exponentially correlated input



(a)



(b)

Figure 3.1: (a) Agreement of analytical and empirical memory function for different λ. (b) Scaling of memory capacity with increasing structure in the input.

we draw T samples u_i from a uniform distribution over the interval $[0,1]$. The samples are passed through a low-pass filter with a smoothing factor α [214]. We normalize and center u_t so that $\langle u(t)\rangle_t = 0$ and $\langle u(t)^2\rangle_t = 1$. The resulting normalized samples $u(t)$ have exponential autocorrelation with decay exponent α, i.e., $R_{uu}(\tau) = e^{-\alpha\tau}$. Figure 3.1(a) shows good agreement between result of the single-instance calculation of the analytical and the empirical memory curves $m(\tau)$ for different λ.

We also study the memory capacity for different levels of structure in the input signal. Here, we use simple ring topologies with $\lambda = 0.9$ and vary the decay exponent α,

Figure 3.1(b). For a fixed system size, decreasing α increases the correlation in the input, which subsequently increases the memory capacity. There are two special cases that can be analyzed from the equations of memory under exponentially correlated inputs. First, when $\alpha \to \infty$ the input approaches i.i.d., a case that has been extensively analyzed before [79, 80] and it is known that under this condition the network can only remember up to N previous inputs. At the other extreme when $\alpha \to 0$ we have a constant signal which the network can remember infinitely. As α goes from ∞ to 0 the signal will gradually have more redundancy which the network exploits to recover longer sequences.

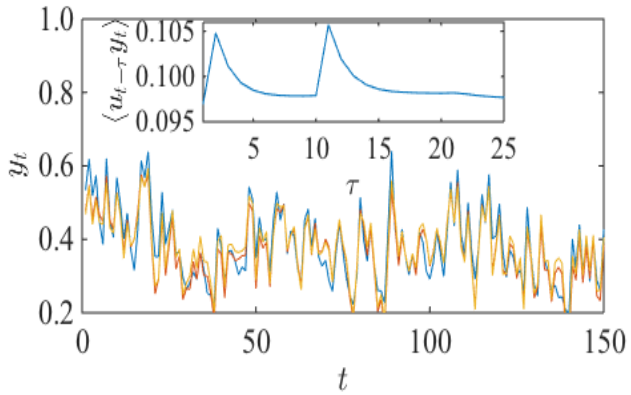
3.4 Solving a Nonlinear Autoregressive Problem

Now we turn to the NARMA10 example previously described in Chapter 2. Several previous studies have reported good performance on this synthetic task using ESN [5, 15, 37, 39, 48]. Here we show that we are able to predict the expected error analytically. Here the evaluation of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ follows the same calculation as for the memory capacity for the uniform distribution. For $\mathbf{X}\mathbf{Y}^{\mathrm{T}}$ we must estimate the cross-correlation of y_t and u_t and substitute it into Equation A.43. We used a sequence of $1,000,000$ time steps to get an accurate estimation of the cross-correlation of input and output. We plug this cross-correlation in Equation 3.6 to evaluate our analytical derivation. For simulation we used $5,000$ time steps to train the readout layer and $5,000$ time steps for testing. Figure 3.2(a) shows the output of a network of $N = 20$ nodes connected in a uniform ring topology with $\lambda = 0.9$. Once again there is no averaging in these results due to uniqueness of topology. The outputs of the network trained with numerical simulation (red curve) and analytically (yellow curve) are in agreement. The outputs agree with the correct output of the NARMA10 system (blue curve). To illustrate the robustness of our analytical results we repeat this experiment with randomly generated networks. Here the networks are fully connected with random weights drawn from a normal distribution with zero mean and standard deviation

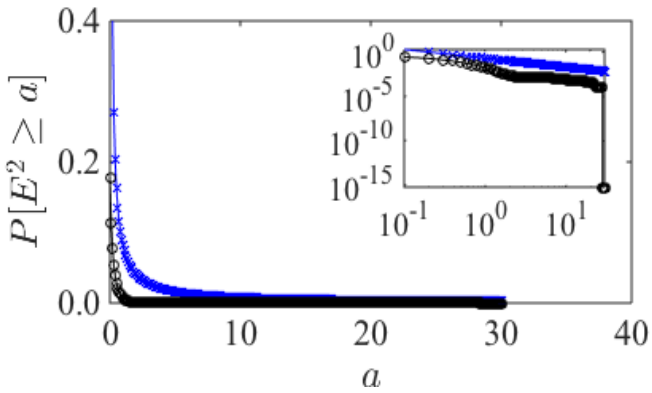
1, then scaled to have spectral radius λ. We then measure the error between the output of the system produced with analytical calculation of the readout weights and readout weights from numerical simulation. Figure 3.3 shows the log of mean-squared-error between the output of the reservoir produced analytically and through simulation for different system size N and spectral radius λ averaged over 10 instantiations of RC. The large negative values over all λ and N indicate that the raw error values are close to zero. In conclusion, we can compute the expected optimal readout layer analytically and therefore predict the performance of the system directly from the reservoir structure. The cross-correlation of the system used for the calculation is shown in the inset. Figure 3.2(b) shows the worst-case error bound for this system and the empirical errors generated from the system output, showing that the bound we derived is tight.

3.5 Solving A Chaotic Prediction Problem

For the NARMA10 example above, the output is a stochastic function of inputs and the inputs are uncorrelated. Moreover, the cross-correlation of the input and output shows significant correlation up to about $\tau = 20$. Here, we use another popular benchmark, the prediction of a chaotic Mackey-Glass system (Chapter 2.) This system is continuous, deterministic and shows chaotic trajectory with non-vanishing input autocorrelation. Again, we will evaluate Equation 3.5 and Equation 3.6, where for predicting τ steps ahead we have $\mathbf{X}\mathbf{Y}^{\mathrm{T}} = \sum_{i=\tau}^{\infty} \Omega^i \omega R_{u\hat{y}}(i)$. We validate our result by calculating the optimal readout weights for predicting $\tau = 10$ step ahead for this time series. We use a ring topology with $N = 100$ and spectral radius $\lambda = 0.9$, $\gamma^2 = 0.0001$. The autocorrelation and cross-correlation were evaluated on a discretized Mackey-Glass time series with $10,000$ steps and 500 washout time steps were used. As before, our goal here is to show that our analytical method can exactly estimate the optimal readout layer given by numerical evaluation of the regression equation for the readout layer. Figure 3.4(a) shows the prediction result



(a)



(b)

Figure 3.2: Target output and system output generated with analytical weights and trained weights for the NARMA10 task (a), the worst-case bounds using the Markov inequality, with the same plot on a log-log scale in the inset (b).

for 10 time steps ahead for the Mackey-Glass system (Section 2.2.3) and the inset shows the autocorrelation at different lags. The autocorrelation is characterized by a long correlation length evident from non-zero correlation values for large τ (Figure 3.4(b)). This long memory is a hallmark of chaotic systems. Since we use a uniform ring topology, a single instance is representative of the network class and no averaging is required. To test robustness of our validation we repeat the experiment with a randomly generated network drawn from a normal distribution with mean zero and standard deviation 1 and rescaled so as to have spectral radius λ. We systematically study different network sizes and spectral

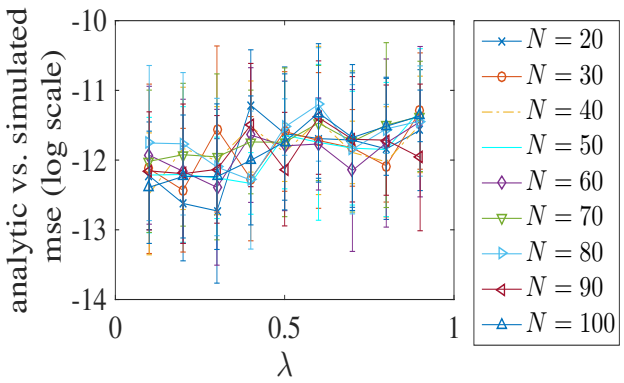
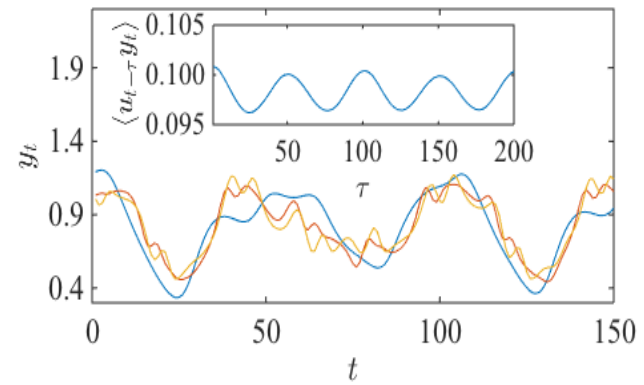


Figure 3.3: Mean-squared-error between analytical and simulated output of reservoir for different system sizes and spectral radii showing the robustness of our analytical methods over different regions of parameter space.

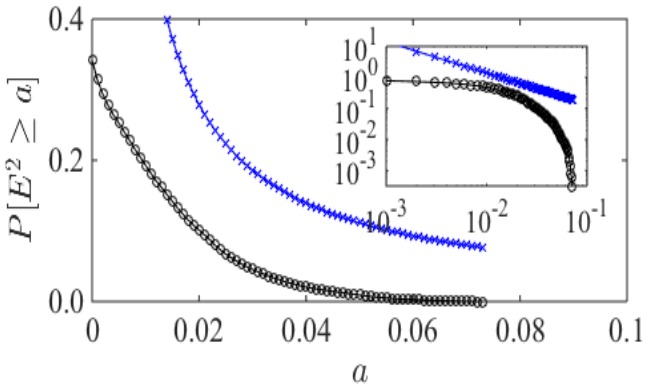
radii and average the results over 10 instantiations of the network for each configuration. We measure the log of mean-squared-error between output of the network with analytical readout weights and numerically simulated readout weights. Figure 3.5 shows the result of this experiment. The large negative values show the raw error values are close to zero and our analytical method is robust to variations in the topology.

3.6 The Effect of Network Structure

The effect of randomness and sparsity of reservoir connectivity has been a subject of debate [206]. To study the effect of network structure on memory and performance, we systematically explore the range between sparse deterministic uniform networks and random graphs. We start from a simple ring topology with identical weights and induce noise to ℓ random links by sampling the normal distribution $\mathcal{N}(0,1)$. We then re-evaluate the memory and task solving performance keeping the weight matrix fixed. We evaluate system performance on a memory task with exponentially correlated inputs, the nonlinear autoregressive NARMA10 task, and the Mackey-Glass chaotic time series prediction. We



(a)



(b)

Figure 3.4: Target output and system output generated with analytical weights and trained weights for the Mackey-Glass 10 step ahead prediction task (a), the worst-case bounds using the Markov inequality, with the same plot on a log-log scale in the inset (b).

systematically explore the effects of λ and ℓ on the performance of the system for fixed system size. For the memory and NARMA10 tasks we set $N = 50$, and for the Mackey-Glass task $N = 100$. The length of data for each tasks is the same as used before in corresponding sections. The results are averaged over 100 runs for memory and Mackey-Glass and 1000 runs for NARMA10 (NARMA10 is an unstable system and needs many samples to generate smooth plots).

Figure 3.6(a) shows the resulting total memory capacity normalized by N as a function

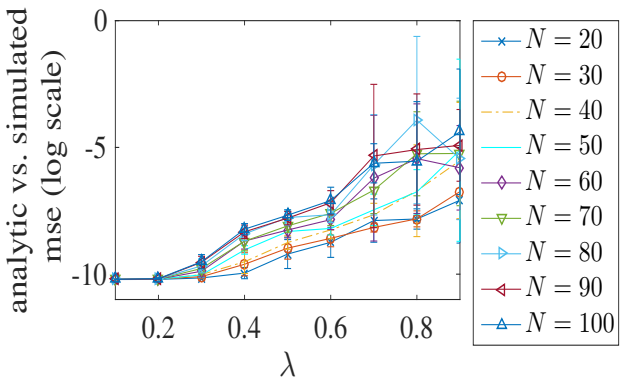


Figure 3.5: Mean-squared-error between analytical and simulated output of reservoir for different system sizes and spectral radii showing the robustness of our analytical methods over different regions of parameter space.

of increasing randomness $\frac{\ell}{N^2}$ for different spectral radii λ. The expected theoretical total memory capacity for an uncorrelated signal is $\sum_\tau m(\tau)/N = 1$. Here the system exploits the structure of the input signal to store longer input sequences, i.e., $\sum_\tau m(\tau)/N > 1$. This effect has been studied previously under annealed approximation and in a compressive sensing setup [186]. However, here we see that even without the sparse input assumption and L_1 optimization in the output (a computationally expensive optimization used in compressive sensing) the network can achieve capacity greater than its degrees of freedom N. Figure 3.6(b) and (c) show the error in the NARMA10 and the Mackey-Glass prediction tasks. Here, best performance is achieved for a regular architecture. A slight randomness significantly increases error at first, but additional irregularity will decrease it. This can be observed for the NARMA10 task at $\lambda = 0.6$ and for the Mackey-Glass prediction task at $\lambda = 0.9$.

The reduced performance in irregular networks can be explained by the distribution of eigenvalues of the weight matrix and their effects on the memory of the system. These eigenvalues define the rate of information decay along the corresponding dimensions of the state space of the reservoir. To analyze this we consider two general cases of completely regular structure, a simple ring and a random network. In a regular network, the eigen-

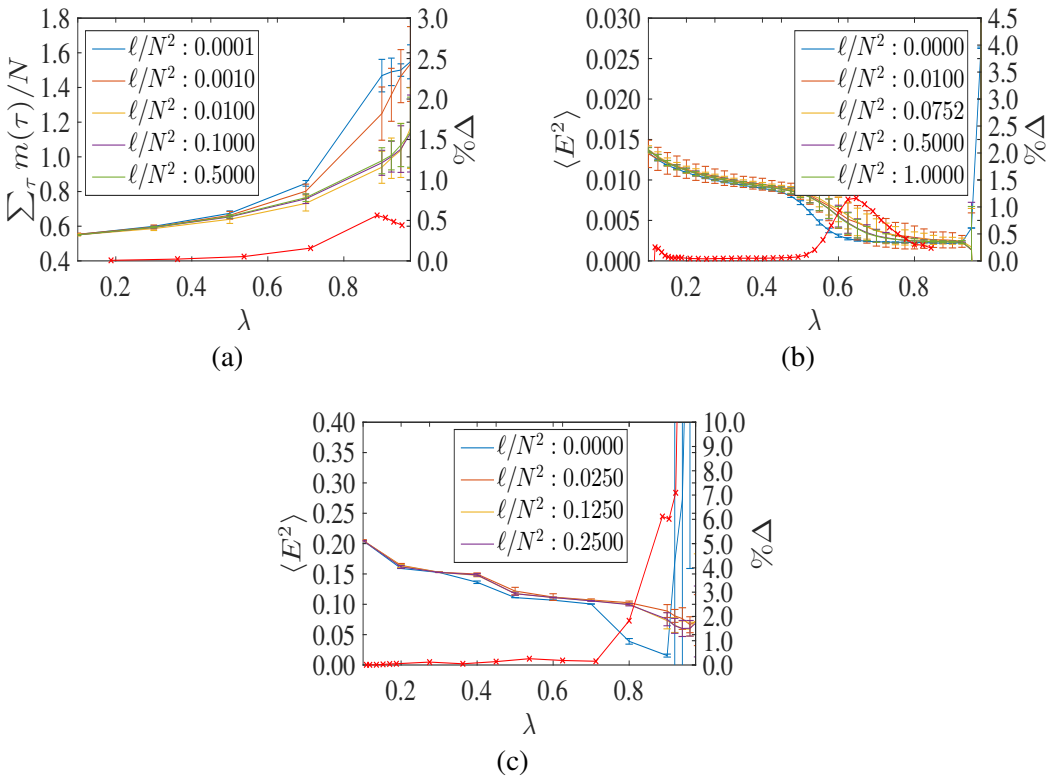


Figure 3.6: Total memory capacity (a), NARMA10 error (b), and Mackey-Glass prediction error (c) as a function of λ and increasing network irregularity. The best results in all cases are for completely regular networks.

values of the weight matrix are equidistant points on the perimeter of a circle with radius $|\lambda|$ on the complex plane. In this case all eigenvalues have identical magnitude, which means that the information decays at uniform rate along all dimensions. For a random network, the eigenvalues fill the complex plane uniformly and therefore each dimension has a different decay rate. Because information decays at different rates along different dimensions the signal loses coherence and therefore the readout layer cannot reconstruct the desired signal perfectly.

3.7 Memory and Computation in RC and Related Models

An early neural model for time series processing was the time-delay neural network [9]. This model was capable of creating an internal representation by integrating weighted delayed version of their inputs according to the backpropagation algorithm. However, it suffered from fixed memory depth, i.e., the number of delays has to be predefined. Recurrent neural networks were proposed to solve the problem of fixed memory depth [9]. However, their training requires backpropagation through time, which is not only costly but is unstable due to vanishing and exploding gradient problem [189]. Reservoir computing solves this issue by initializing a recurrent network as a reservoir in a stable state near the edge of chaos and training only a linear readout layer on the emergent short-term memory of the reservoir [5]. The linearity of the readout layer and the fixed reservoir make an efficient closed-form calculation of the readout weights possible. On the other hand, the number of previous inputs that can be accessed in the short-term memory is limited and depends on the number of nodes in the reservoir and the spectral radius of the reservoir. In this thesis, I showed that even the correlations accessible to the readout layer during task solving decay exponentially with the correlation lag τ. This means that despite the fast training, RC cannot use correlations at arbitrary timescale for a fixed system size and spectral radius. Moreover, for a fixed reservoir size and spectral radius, dependencies on previous inputs with lag will increase the values of the readout weights as an exponential of τ [80]. This will particularly limit the application of RC for physical devices where the weights are encoded using physical material. Increasing the spectral radius could increase the memory depth of the reservoir at the cost of approaching the critical threshold where the integrals of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ becomes divergent. Recently introduced architectures such as the LSTM [187] could, in principle, solve the memory limit problem. However, their training is computationally expensive and requires large-scale parallel processing, such as GPU clusters. In summary, RC could provide an efficient alternative for time series processing to time-delay networks and LSTM, if the correlation lags that are required for the task

are finite, but RC is not suitable for tasks with unknown correlation lags that could be arbitrarily long. In terms of the performance in task solving, if a mean-squared objective function is used and the model structure allows the required memory depth for the task, then time-delay networks, RC, and LSTM should result in the same error, provided that a proper schedule for the learning rate is used to ensure convergence to the solution in LSTM and time-delay networks.

3.8 Discussion

Although memory capacity of ESNs has been studied before, its learning and generalization ability in a task solving setup has not. Our derivation allows us to relate the memory capacity to task-solving performance for arbitrary ESNs and reason about their generalization. In empirical experiments with systems presented here, the training and testing are done with finite input sequences that are sampled independently for each experiment, so the statistics of the training and testing inputs vary according to a Gaussian distribution around their true values and one expects these estimates to approach their true values with increasing sample size. Hence, the mean-squared-error $\langle E^2 \rangle$, which is linear in the input and output statistics, is also distributed as a Gaussian for repeated experiments. By the law of large numbers, the difference between testing and training mean-squared-error tends to zero in the limit. This explains the ability of the system to generalize its computation from training to test samples.

The computational power of reservoir computing networks has been attributed to their memory capacity. While their memory properties have been studied under annealed approximation, previously no direct mathematical connection to their signal-processing performance had been made. We developed a mathematical framework to exactly calculate the memory capacity of RC systems and extended the framework to study their expected and worst-case errors on a given task in a supervised learning setup. Our framework gives

us an explanation for the different behavior observed in a delay line with an adaptive readout layer vs. reservoir computing and invalidates the claim that reservoir computing is "just" a regression on a memory. We have seen that the reservoir performs computation. In fact from the point of view of a linear readout layer, the reservoir in RC does not store raw input values, but stores the values based on how they are correlated with each other. Our result confirms previous studies that the upper bound for memory capacity for uncorrelated inputs is N. We further show that the memory capacity monotonically increases with correlation in the input. Intuitively, the output exploits the redundant structure of the inputs to retrieve longer sequences. Moreover, we generalize our derivation to task-solving performance. Our derivation helps us reason about the memory and performance of arbitrary systems directly in terms of their structure. We have seen that unlike the claim in [48], linear ESNs cannot have a λ-independent memory lower bound. We showed that networks with regular structure have a higher memory capacity but are very sensitive to slight changes in structure, while irregular networks are robust to variation in their structure.

Chapter 4

Conclusion

Computing with physical systems is a perennial topic of computer science research. Recent advances in materials science and nanotechnology have made this academic topic realizable. Meanwhile, the limitations in scaling current silicon technology have created commercial motivation to research possible alternatives to von Neumann-based computing. Research in unconventional computing paradigms must develop new ways of conceptualizing computation and characterizing its power. One way of using physical systems for computation is through their characterization as dynamical systems. This enables researchers to tap into one of the most prevalent sources of computation in nature, called the short-term memory. To use short-term memory an excitable system is driven with a temporal input. The dynamics of the system encodes the temporal input for a short period of time. A readout layer can extract this information and use it to generate a desired output. The coefficients, or weights, of the readout layer are computed with a closed-form linear regression, which is efficient. This approach was developed in the neural networks community (under the name of reservoir computing) as a way of modeling the prefrontal cortex and also a solution to avoid computationally expensive training of recurrent neural networks. Despite promising practical applications the theory of reservoir computing lacked mathematical development. Two prominent efforts to characterize reservoir computing

used mean-field approximation to compute the memory of the system. In this dissertation, I developed an exact mathematical framework to compute the memory of a standard model of reservoir computing, ESN. I generalized the framework to compute expected error as well as a worst-case error bound of ESN for generic temporal tasks. I validated this method by applying it to memory tasks for uncorrelated and correlated inputs, a chaotic prediction task, and a nonlinear autoregressive task. I further showed that the method can be insightful for reasoning about the performance of nonlinear ESN. I experimented with two types of nonlinearity: (1) nonlinear presynaptic integration and (2) nonlinear transfer function. I showed that, as expected according to the insights from the mathematical framework, even a small nonlinearity in the system gives the ESN almost the same power as a fully nonlinear ESN with a tanh transfer function.

This dissertation has contributed to understanding the properties of short-term memory in neuroscience and its application in machine learning and unconventional computing. While the current limitation of the work is that it only applies to linear systems, the contribution is still important and relevant to physical implementation of RC based on signal propagation through a linear reservoir, such as the photonic RC. Our framework makes it possible to exactly evaluate the optimal readout weights and therefore performance and memory properties of an arbitrary recurrent network, for arbitrary inputs and any given task, and gives an estimate for the worst-case error on the task. In addition, one can use the presented framework to exactly determine how memory and performance in the system change with respect to changes in the structure of the network. Moving beyond the current work, there remain crucial questions to be answered in RC. For example, what are the promising approaches to investigate memory and information processing capacity in nonlinear networks? Can such a framework give an explicit relation between the performance and the structure of the system? Will such a framework give us a way to build a network through a constructive or an adaptive algorithm to achieve minimum required performance for a given task? Is it feasible for RC to be applied to large-scale machine learning problems and what would be the real benefits of using RC in the field of big data and predictive

analysis? On a deeper theoretical level, one can ask what "short" in short-term memory in RC means. Clearly, the memory of past inputs decays over time, but RC particularly thrives in chaotic prediction problems, which are by definition characterized by long correlation lengths. Therefore, "short" must simply allude to time-scales that are not arbitrarily long, or in another word *finite* correlation length. In this case from the automata theory perspective, can we say that RC is equivalent to a finite state machine and can recognize all regular languages? Additionally, if one allows adaptive system size, will RC become akin to a nondeterministic Turing machine and able to recognize context-sensitive languages? Moreover, there are versions of RC that include feedback from the output to the reservoir. This induces a NARX-like architecture [219, 221], which has been shown to be Turing-universal. If such RCs are truly Turing universal, which is an open question, how can one train them to implement any given algorithm? Will such an algorithm help advance the current state-of-the-art in recurrent neural networks, which requires highly specialized architectures, such as gated networks [188] and LSTM [187], for the training algorithms to work properly. This dissertation shed light on some of the unknowns in the field of RC, such as expected performance on a given task, worst-case performance, and the effect of structure on the task performance. Although the formulation only applies to linear systems, in principle it can be used to describe the properties of physical reservoir computing, e.g., propagation of short coherence-length light through the reservoir medium and I hope that my results can help illuminate the path to answering some of these questions in the future.

T

Appendix A

A Mathematical Framework for Studying the ESN

My aim in this chapter is to use ESN with linear transfer function to develop a theoretical framework that allows us to form an expectation about the performance of RC for a desired computation. To demonstrate the power of this framework, I will apply it to the problem of memory curve characterization in ESN. The ESN is a simple and tractable model to study reservoir computing.

Here, I will use the memory capacity task as a simple example and derive an equation for the optimal readout weights for a one-dimensional reservoir based on the reservoir feedback weight, the input weight, and the input statistics. The one-dimensional example helps to understand the mechanics of the framework without having to deal with the complexities of matrix multiplications. I will then build on the one-dimensional example to derive the optimal readout weights for N-dimensional reservoirs. Generalizing the framework to N dimensions employs advanced linear algebra to get analytical solutions to matrix multiplications. In both one-dimensional and N-dimensional cases I have assumed an uncorrelated input signal for simplicity. I will compare the results of the analytical

calculations with numerical results for validation of my method. I hope that these two examples give the reader a good perspective for how the remaining results are derived. The remainder of the chapter will introduce the generic structure of the mathematical framework in abstract and succinct terms. I will start with the regression equation for the optimal readout weights for an arbitrary task and show it can be rewritten in terms of the structure of the reservoir and correlation structure of the input and output. I will then show that my derivation can be used not only to calculate the memory capacity, but also to derive an expected error for a given task. I will then show that the same framework can be used to derive a worst-case bound on the expected error for the task.

A.1 Analytical Solution to the Memory Curve: A One-Dimensional System

The method I developed to perform the calculations uses many steps and observations about the structure of the system and its dynamics. To demonstrate the framework, I will first explain how the calculations can be done in a one-dimensional system. This will permit the representation of the structure of the system as just scalars, which is much easier to work with and is sufficient as a pedagogical device. After developing our intuition about the logic of the derivation I will show the same process for an N dimensional reservoir in Section A.2.

A.1.1 Model

Let us review the description of the system and the memory curve. We put the time t as the subscript for aesthetics. The model is a linear ESN with a reservoir of one node. The reservoir connectivity is a feedback connection from the reservoir node to itself. This is denoted by a scalar Ω, where $|\Omega| < 1$. This is a necessary condition for the stability of

the system. A time-dependent scalar input signal u_t is fed to the reservoir using the input weight ω, also a scalar. The time-dependent reservoir states are denoted by the scalar $\mathbf{x}_t$. The time evolution of the reservoir $\mathbf{x}_t$ is governed by

$$x(t+1) = \Omega x(t) + \omega u(t), \tag{A.1}$$

and the output y_t is given by

$$y(t+1) = \Psi x(t+1), \tag{A.2}$$

where Ψ is a scalar calculated for a desired output $\widehat{y}_t$ by:

$$\Psi = \langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle^{-1} \langle \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}} \rangle. \tag{A.3}$$

Here $\mathbf{X}$ is a row vector whose columns are the states of the reservoir in time and the rows of $\widehat{\mathbf{Y}}^{\mathrm{T}}$ are the corresponding desired output at each time step. The memory function for the system is defined as the coefficient of determination between the output of the system and its τ past inputs:

$$MC_\tau = \frac{\mathrm{Cov}^2(u_{t-\tau}, y_t)}{\mathrm{Var}(u_{t-\tau})\mathrm{Var}(y_t)}, \tag{A.4}$$

where u_t is the input at time t, $u_{t-\tau}$ is the corresponding target output, and y_t is the output of the network given the optimal Ψ. The inputs u_t are drawn from identical and independent uniform distributions in the range $[-1, 1]$. The intuition behind this formula is to take the τ-delayed input $u_{t-\tau}$ and see how well the system can reconstruct it.

A.1.2 Computing $\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle$

To calculate the covariance component we first note that we can rewrite Equation A.1 explicitly in terms of the input history and initial condition of the system

$$x(t+1) = x_0 \sum_{i=0}^{t} \Omega^i + \sum_{i=0}^{t} \Omega^i \omega u_{t-i}. \tag{A.5}$$

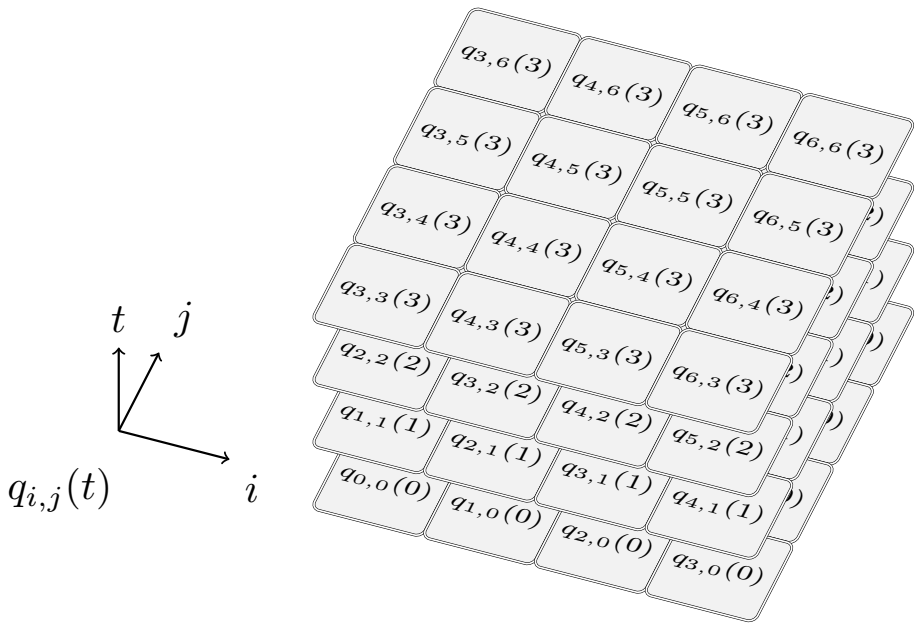


Figure A.1: The summation in Equation A.27 can be visualized as summing the terms $q_{i,j}(t)$ along three axis of i, j, and t.

However, since $|\Omega| < 1$ the contributions of the initial state will vanish over time and we are left with:

$$x(t+1) = \sum_{i=0}^{t} \Omega^i \omega u_{t-i}. \tag{A.6}$$

Now, we can write the covariance component as:

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \langle x^2(t) \rangle_{t=0}^{T} = \frac{1}{T} \sum_{t=0}^{T} x^2(t) \tag{A.7}$$

$$= \left\langle \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \Omega^{i+j} \omega^2 u_{t-i} u_{t-j} \right\rangle_{t=0}^{T} \tag{A.8}$$

$$= \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \Omega^{i+j} \omega^2 u_{t-i} u_{t-j} \tag{A.9}$$

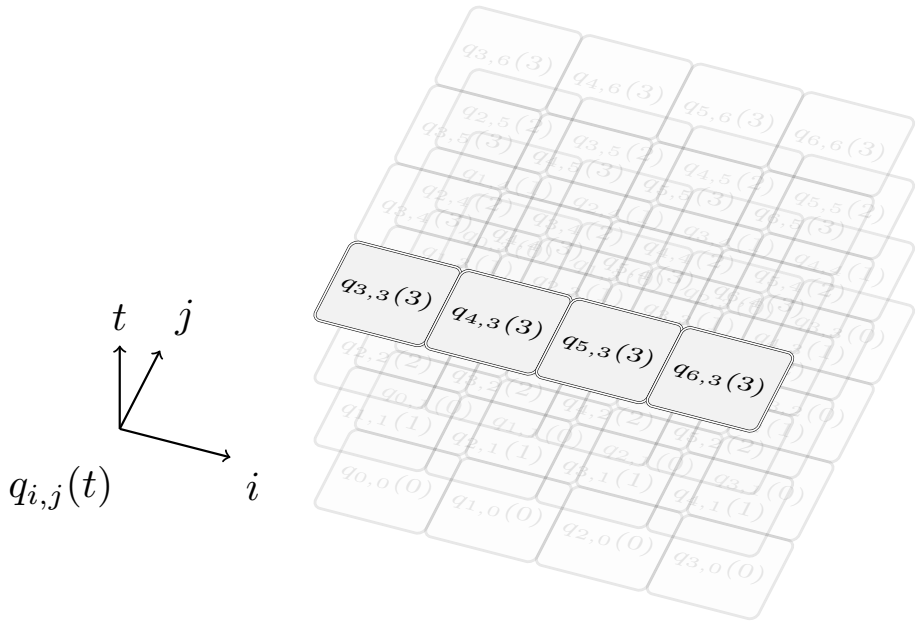


Figure A.2: The summation along the i axis.

To see how we can calculate Equation A.9 analytically it helps to visualize the formula as summing along three different axis of i, j, and t. Figure A.1 illustrates this. Let

$$q_{i,j}(t+1) = \Omega^{i+j}\omega^2 u_{t-i}u_{t-j}, \tag{A.10}$$

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \frac{1}{T}\sum_{t=0}^{T}\sum_{i=0}^{t-1}\sum_{j=0}^{t-1} q_{i,j}(t), T \to \infty. \tag{A.11}$$

Our task is to calculate the sum of $q_{i,j}(t)$ along the three axes of the cube. The sum over i and j is the sum of $q_{i,j}(t)$ in one horizontal layer, and then we sum over all the layers. Normally, we would be able to perform this sum using the power series identities along the i, j, and then t axis in order. However, in this case we face a complication. To see this, let us look at the sum along the i axis (Figure A.2). We now expand the values of $q_{i,j}(t)$ to see that they consist of a power sum, in which each term is multiplied by a constant and a stochastic part given by the input signal at the corresponding time step (Figure A.3). This stochasticity of each term prevents us from performing the sum along the i axis. Similarly we cannot perform this sum along the j axis. Note that this sum corresponds to the sum of

terms on a single layer of the cube, which corresponds to the instantaneous cross-product of the reservoir x_t^2.

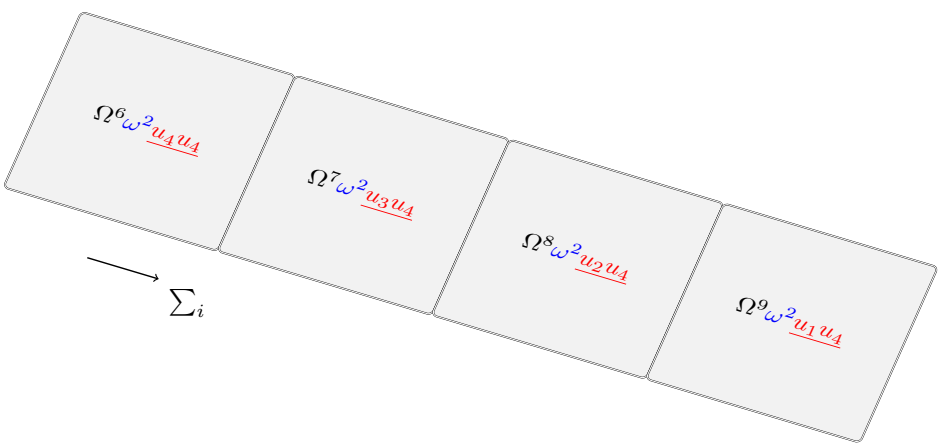


Figure A.3: Expanding the values along the i axis. The blue color are constants, the black shows the power series, and the red shows the stochastic factor.

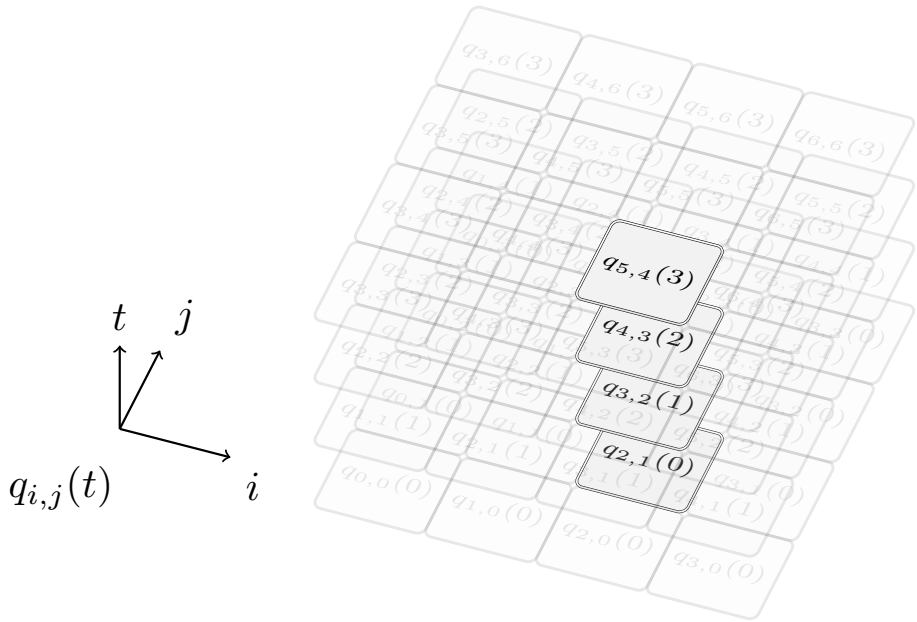


Figure A.4: The summation along the t axis.

However, if we look at a single column along the t axis (Figure A.4), we find that if we expand the values of $q_{i,j}(t)$, the terms of the power sum in the column consist of $u_{t-i}u_{t-j}$ multiplied by a constant factor (Figure A.5). Note that for each i and j this summation gives us the δ-delayed auto-covariance of the input signal. Therefore, we have found a trick to perform the sum vertically along each column of the cube. This leaves us to calculate the sum over i and j.

The algebraic representation of this trick is aligning the terms of the summation so we

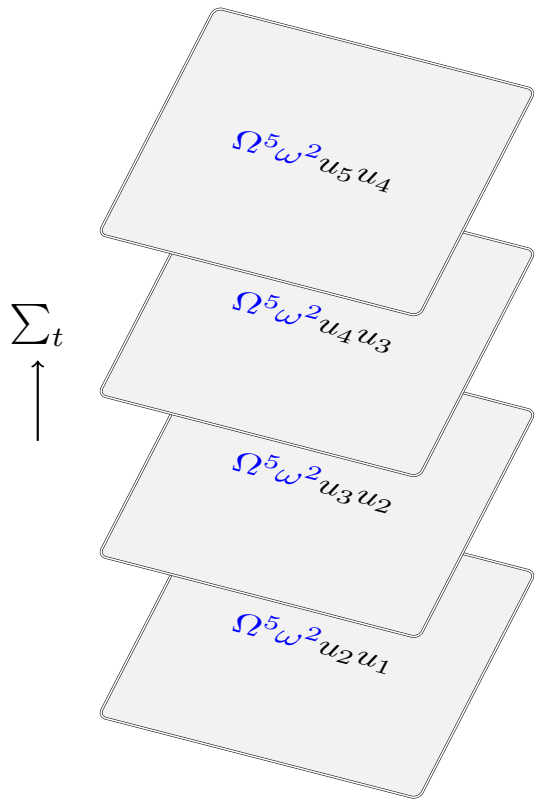


Figure A.5: Expanding the values along the t axis. The blue color are constants, the black shows the multiplication of input with a delayed version of itself.

have terms with equal powers of Ω on each column as follows:

$$\begin{aligned}
\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle &= \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \Omega^{i+j} \omega^2 u_{t-i} u_{t-j}, \\
&= \frac{1}{T} \ldots \Omega^{0+0} \omega^2 u_t u_t + \Omega^{0+1} \omega^2 u_t u_{t-1} + \ldots \\
&+ \ldots \Omega^{0+0} \omega^2 u_{t-1} u_{t-1} + \Omega^{0+1} \omega^2 u_{t-1} u_{t-2} + \ldots \\
&+ \ldots \Omega^{0+0} \omega^2 u_{t-2} u_{t-2} + \Omega^{0+1} \omega^2 u_{t-2} u_{t-3} + \ldots \\
&+ \ldots \Omega^{0+0} \omega^2 \underbrace{u_{t-3} u_{t-3}}_{\delta=0} + \Omega^{0+1} \omega^2 \underbrace{u_{t-3} u_{t-4}}_{\delta=1} + \ldots
\end{aligned}$$

We can then perform the sum column-wise, which is equivalent to performing the triple sum from outermost sum. I arranged this so that the columns have identical coefficients.

Also note that for $i > \theta$ we have $\Omega^i \to 0$. Therefore we can write:

$$\begin{aligned}\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle &= \ldots \Omega^{t+t} \omega^2 \langle u_t u_t \rangle + \Omega^{t+t-1} \omega^2 \langle u_t u_{t+1} \rangle + \ldots \\ &= \sum_{i=0}^{\theta} \sum_{j=0}^{\theta} \Omega^{i+j} \omega^2 \langle u_{t-i} u_{t-j} \rangle\end{aligned}$$

Remember that u_is are drawn from i.i.d. uniform distribution on the interval $[-1,1]$. The variance of this distribution will be $\langle u^2 \rangle = \frac{1}{3}$ and because of the independence condition the covariance of two variables will be zero. Therefore:

$$\langle u_{t-i} u_{t-j} \rangle = \begin{cases} 0 & , \text{ if } i \neq j \\ \langle u^2 \rangle & , \text{ if } i = j \end{cases} \tag{A.12}$$

Hence we can write:

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \langle u^2 \rangle \omega^2 \sum_{i=0}^{\theta} \Omega^{2i} \approx \frac{\langle u^2 \rangle \omega^2}{1 - \Omega^2} \tag{A.13}$$

Here we have shown that to calculate the covariance component, we only need to calculate the sum of the diagonal elements of the cube as illustrated in Figure A.6.

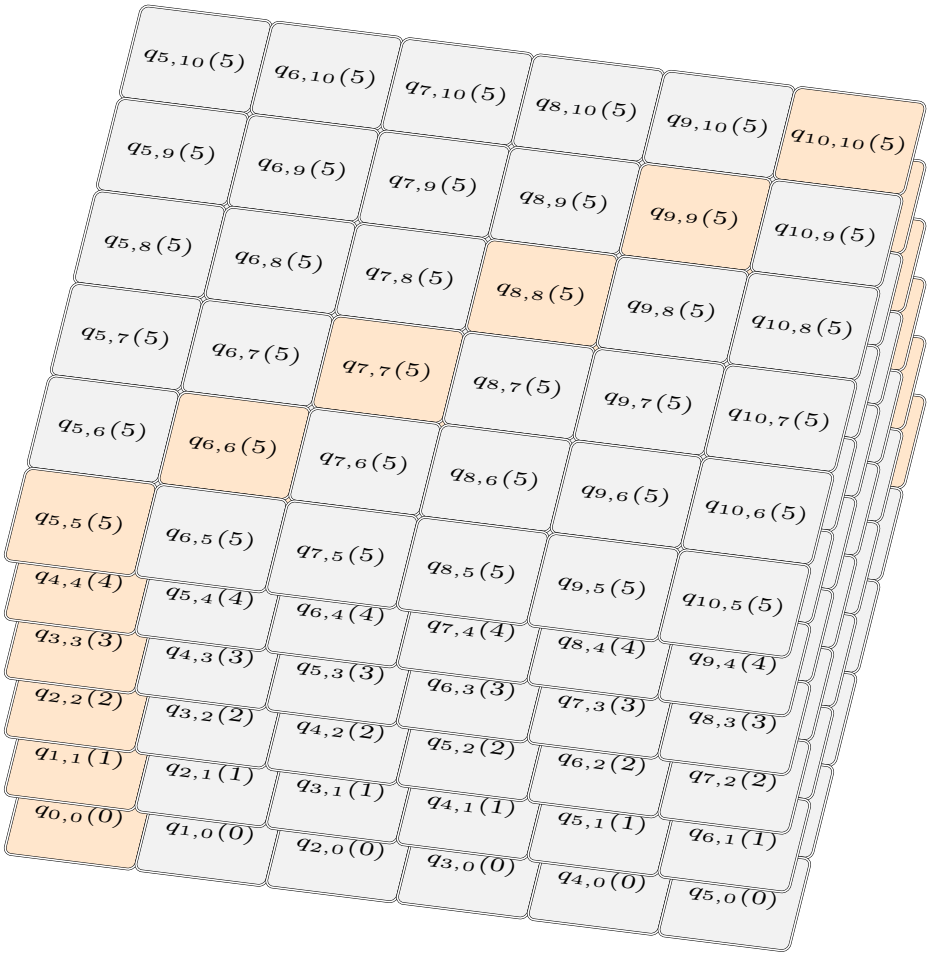


Figure A.6: For zero mean input signal drawn from i.i.d. uniform distribution, the covariance of the reservoir $\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle$ is given by the average of the diagonal terms on the cube.

A.1.3 Computing $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle$

The next step is to compute the projection component $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle$. First note that the desired output does not appear anywhere in the expression of $\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle$. This means that the covariance component is constant for a given ESN and input distribution. Now recall that the projection component is given by:

$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle = \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \Omega^i \omega u_{t-i} \widehat{y'}_t, \tag{A.14}$$

which means it depends on the desired output. To compute the memory function MC_τ, we have a different desired output for each τ, namely $u_{t-\tau}$. Therefore, we specify the projection component for each τ with a subscript $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau$. We can now write:

$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau = \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \Omega^i \omega u_{t-i} u_{t-\tau}, \tag{A.15}$$

Again we see that for $i > \theta$, $\Omega^i \to 0$ and that for all $i \neq \tau$ we have $\langle u_{t-i} u_{t-\tau} \rangle = 0$. It follows that:

$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau = \Omega^\tau \omega \langle u^2 \rangle. \tag{A.16}$$

A.2 Analytical Solution to the Memory Curve: An N-Dimensional System

In a high-dimensional system, the calculation of the memory curve follows the same logic as before. However, in this case one must be aware that the elements of calculations consist of matrices and vectors and therefore some of the simplification of the scalar case does not apply. Nevertheless, analytical calculation is still possible using a matrix identity that will help us perform the power sum. Let us review the model once again. The equations are adopted from the 1-dimensional case and repeated here for convenience.

A.2.1 Model

Here, the system is a linear ESN with a reservoir of N randomly interconnected nodes. The reservoir weight matrix is denoted by a $N \times N$ matrix Ω with spectral radius $\lambda^{max} < 1$. A time-dependent scalar input signal u_t is fed to the reservoir using the input weight vector ω. The time-dependent reservoir states are denoted by a column vector $\mathbf{x}(t)$. The time evolution of the reservoir state $\mathbf{x}(t)$ is governed by

$$\mathbf{x}(t+1) = \Omega\mathbf{x}(t) + \omega u(t), \tag{A.17}$$

and the output y_t is given by

$$y(t+1) = \Psi\mathbf{x}(t+1), \tag{A.18}$$

where Ψ is an N-dimensional column vector calculated for a desired output $\widehat{y}_t$ as:

$$\Psi = \langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle^{-1} \langle \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}} \rangle. \tag{A.19}$$

Here, the columns of $\mathbf{X}$ are the states of the reservoir in time $\mathbf{x}_t$ and the columns of $\widehat{\mathbf{Y}}$ are the corresponding desired output at each time step. Table A.1 provides a visual representation of all the matrices and their dimensionality to make it easier to follow the rest of the exposition.

The memory function for the system is defined as the coefficient of determination between the output of the system and its τ past inputs:

$$MC_\tau = \frac{\mathrm{Cov}^2(u_{t-\tau}, y_t)}{\mathrm{Var}(u_{t-\tau})\mathrm{Var}(y_t)}, \tag{A.20}$$

where u_t is the input at time t, $u_{t-\tau}$ is the corresponding target output, and y_t is the output of the network given the optimal Ψ. The inputs u_t are drawn from identical and independent uniform distributions in the range $[-1, 1]$.

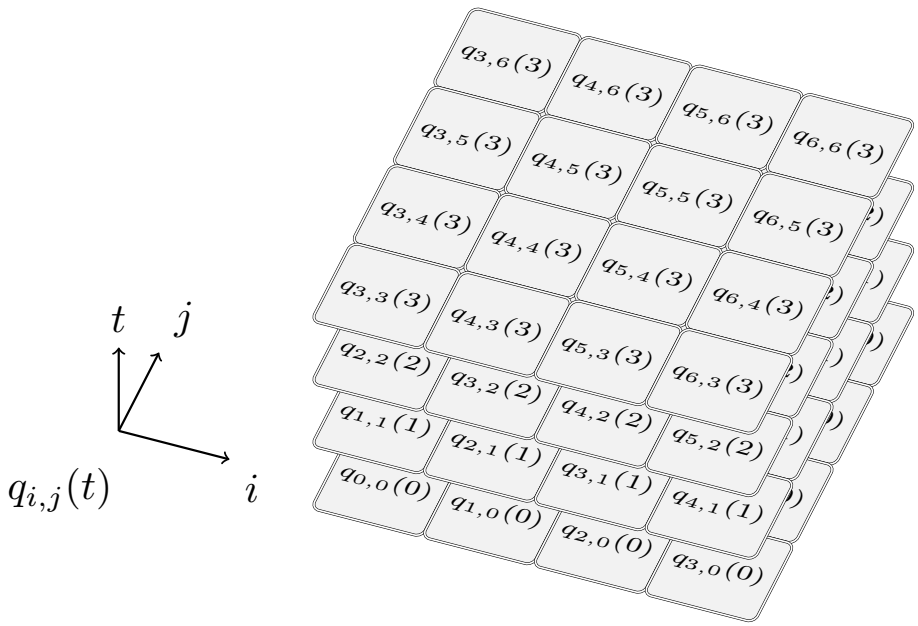


Figure A.7: The summation in Equation A.27 can be visualized as summing the terms $q_{i,j}(t)$ along three axis of i, j, and t.

Table A.1: A list of vectors and matrices used in the calculations along with their dimensionality. The shorthand notation $\mathbf{x}_t$ denotes the state vector and each of its elements $x_i(t)$ denote the state of the corresponding node i at time t. The vector ω is the input weight matrix, Ψ the output weight matrix, and $\widehat{\mathbf{Y}}^{\mathrm{T}}$ is the desired output vector elements of which correspond to $\widehat{Y}^{\mathrm{T}}$. Finally, Ω is the reservoir weight matrix and $\mathbf{X}$ is a $N \times T$ matrix whose columns correspond to the states of the reservoir at each time step.

$$\mathbf{x}_t = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_N(t) \end{bmatrix} \qquad \omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_N \end{bmatrix} \qquad \Psi = \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_N \end{bmatrix} \qquad \widehat{\mathbf{Y}}^{\mathrm{T}} = \begin{bmatrix} \widehat{y}_1 \\ \widehat{y}_2 \\ \vdots \\ \widehat{y}_T \end{bmatrix}$$

$$\Omega = \begin{bmatrix} \Omega_{(1,1)} & \Omega_{(1,2)} & \dots & \Omega_{(1,N)} \\ \Omega_{(2,1)} & \Omega_{(2,2)} & \dots & \Omega_{(2,N)} \\ \vdots & \vdots & \vdots & \vdots \\ \Omega_{(N,1)} & \Omega_{(N,2)} & \dots & \Omega_{(N,N)} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} X_{(1,1)} & X_{(1,2)} & \dots & X_{(1,T)} \\ X_{(2,1)} & X_{(2,2)} & \dots & X_{(2,T)} \\ \vdots & \vdots & \vdots & \vdots \\ X_{(N,1)} & X_{(N,2)} & \dots & X_{(N,T)} \end{bmatrix}$$

A.2.2 Computing $\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle$

To calculate the covariance component we first note that we can rewrite Equation A.17 explicitly in terms of the input history and initial condition of the system

$$\mathbf{x}(t+1) = x_0 \sum_{i=0}^{t} \Omega^i + \sum_{i=0}^{t} \Omega^i \omega u_{t-i}. \tag{A.21}$$

However, since the spectral radius of Ω, $\lambda_{max} < 1$ the contributions of the initial state will vanish over time and we will be left with:

$$\mathbf{x}(t+1) = \sum_{i=0}^{t} \Omega^i \omega u_{t-i}. \tag{A.22}$$

We can write the covariance component as:

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \langle \mathbf{x}(t) x^{\mathrm{T}}(t) \rangle_{t=0}^{T} = \frac{1}{T} \sum_{t=0}^{T} \mathbf{x}(t) x^{\mathrm{T}}(t) \tag{A.23}$$

$$= \left\langle \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \Omega^i \omega u_{t-i} u_{t-j}^{\mathrm{T}} \omega^{\mathrm{T}} \Omega^{\mathrm{T}j} \right\rangle_{t=0}^{T} \tag{A.24}$$

$$= \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \Omega^i \omega u_{t-i} u_{t-j}^{\mathrm{T}} \omega^{\mathrm{T}} \Omega^{\mathrm{T}j} \tag{A.25}$$

To see how we can calculate the Equation A.25 analytically it helps to visualize the formula as summing along three different axes of i, j, and t. Figure A.7 illustrates this. Let

$$q_{i,j}(t+1) = \Omega^i \omega u_{t-i} u_{t-j}^{\mathrm{T}} \omega^{\mathrm{T}} \Omega^{\mathrm{T}j}, \tag{A.26}$$

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} q_{i,j}(t), T \to \infty. \tag{A.27}$$

Our task is to calculate the sum of $q_{i,j}(t)$ along the three axes of the cube. The sum over i and j are the sum of $q_{i,j}(t)$ in one horizontal layer and then we sum over all the layers. Once again we notice that we cannot perform the summation along the i and j axes due to the stochastic factors(Figure A.8).

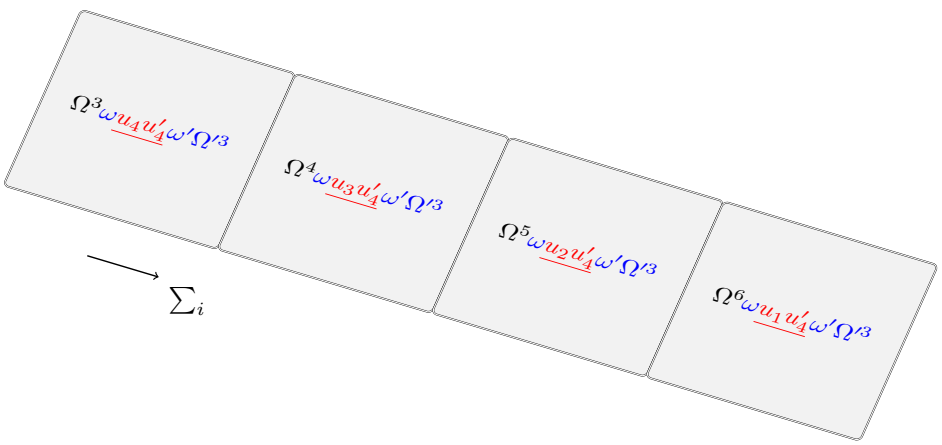


Figure A.8: Expanding the values along the i axis. The blue color are constants, the black shows the power series, and the red shows the stochastic factor.

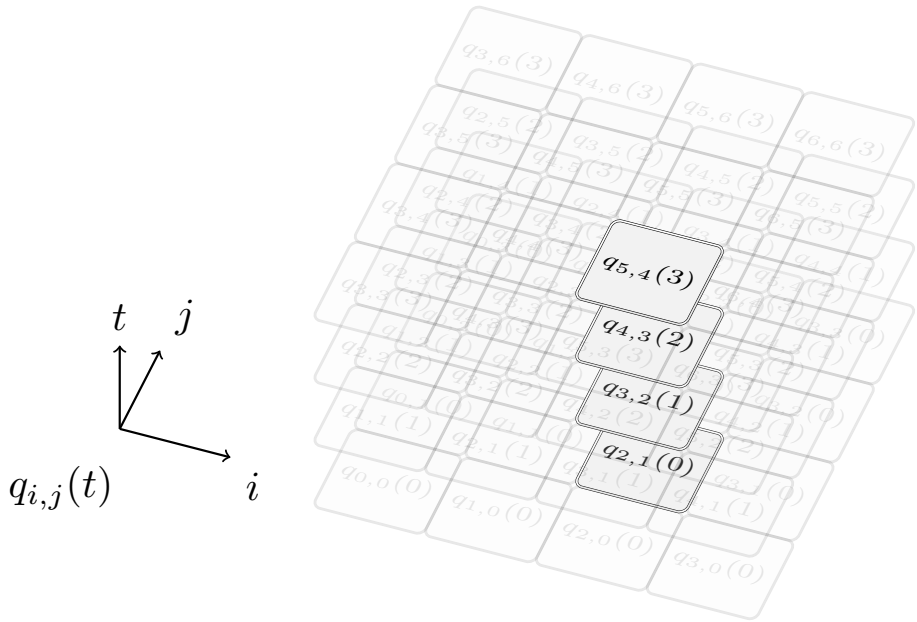


Figure A.9: The summation along the t axis.

However, if we look at a single column along the t axis (Figure A.9), we find that if we expand the values of $q_{i,j}(t)$ the terms of the power sum in the column consist of $u_{t-i}u_{t-j}$ squeezed from each side by constant factors (Figure A.10). Note that for each i and j this summations gives us the δ-delayed auto-covariance of the input signal. Therefore, we have found a trick to perform the vertical sum along the each column of the cube. This leaves us to calculate the sum over i and j.

The algebraic representation of this trick is aligning the terms of the summation so we

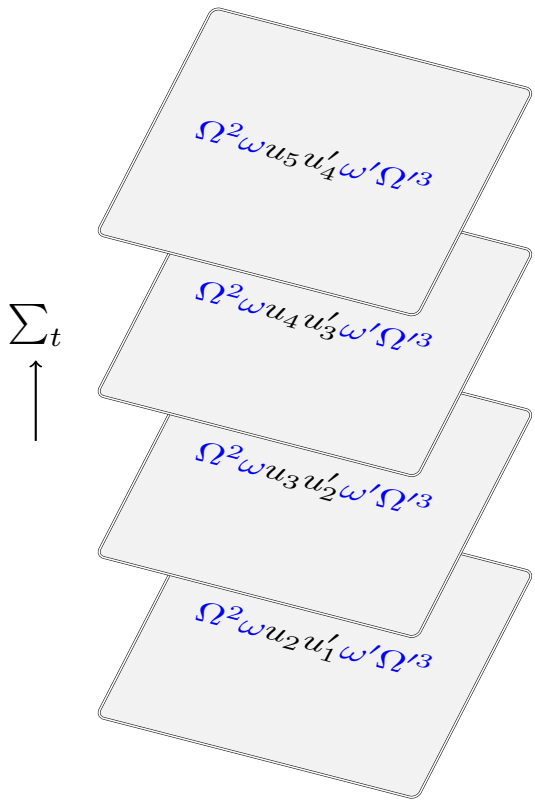


Figure A.10: Expanding the values along the t axis. The blue color are constants, the black shows the multiplication of input with a delayed version of itself.

have terms with equal powers of Ω on each column as follows:

$$\begin{aligned}
\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle &= \frac{1}{T}\sum_{t=0}^{T}\sum_{i=0}^{t-1}\sum_{j=0}^{t-1}\Omega^i u_{t-i}u_{t-j}^{\mathrm{T}}\Omega'^j, \\
&= \frac{1}{T}\ldots\Omega^0 u_t u_t^{\mathrm{T}}\Omega'^0 + \Omega^0 u_t u_{t-1}^{\mathrm{T}}\Omega'^1 + \ldots \\
&+ \ldots\Omega^0 u_{t-1}u_{t-1}^{\mathrm{T}}\Omega'^0 + \Omega^0 u_{t-1}u_{t-2}^{\mathrm{T}}\Omega'^1 + \ldots \\
&+ \ldots\Omega^0 u_{t-2}u_{t-2}^{\mathrm{T}}\Omega'^0 + \Omega^0 u_{t-2}u_{t-3}^{\mathrm{T}}\Omega'^1 + \ldots \\
&+ \ldots\Omega^0 \underbrace{u_{t-3}u_{t-3}^{\mathrm{T}}}_{\delta=0}\Omega^{\mathrm{T}0} + \Omega^0 \underbrace{u_{t-3}u_{t-4}^{\mathrm{T}}}_{\delta=1}\Omega^{\mathrm{T}1} + \ldots
\end{aligned}$$

We can then perform the sum columnwise, which is equivalent to performing the triple sum from outermost sum. I arranged this so that the columns have identical coefficients to

the left and right the $u_i u_j$ term. Also note that for $i > \theta$ we have $\Omega^i \to 0$. Therefore we can write:

$$\begin{aligned}\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle &= \ldots \Omega^t \langle u_t u_t^{\mathrm{T}} \rangle \Omega^{\mathrm{T}t} + \Omega^t \langle u_t u_{t+1}^{\mathrm{T}} \rangle \Omega^{\mathrm{T}t-1} + \ldots \\ &= \sum_{i=0}^{\theta} \sum_{j=0}^{\theta} \Omega^i \omega \langle u_{t-i} u_{t-j} \rangle \omega^{\mathrm{T}} \Omega^{\mathrm{T}j} \end{aligned} \tag{A.28}$$

Remember that u_is are drawn from i.i.d. uniform distribution on the interval $[-1,1]$. The variance of this distribution will be $\langle u^2 \rangle = \frac{1}{3}$ and because of the independence condition the covariance of two variables will be zero and therefore:

$$\langle u_{t-i} u_{t-j} \rangle = \begin{cases} 0 & , \text{ if } i \neq j \\ \langle u^2 \rangle & , \text{ if } i = j \end{cases} \tag{A.29}$$

And therefore we can write:

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \langle u^2 \rangle \sum_{i=0}^{\theta} \Omega^i \omega \omega^{\mathrm{T}} \Omega^{\mathrm{T}i} \tag{A.30}$$

This sum still cannot be performed because of the special matrix form. We have to perform a trick to be able to do this sum. Now let $\Lambda = \bar{\omega}\bar{\omega}\mathrm{T}$, and $\Omega = UDU^{-1}$ the eigen decomposition of Ω, and $\bar{\omega} = U^{-1}\omega$. Also let the vector d be the diagonal elements of D (the eigenvalues of Ω).

We use the fact that for diagonal matrices A and B, and vectors a and b holding the diagonal elements of A and B respectively and an arbitrary matrix C we can write:

$$ACB = C \circ AI^{\circ}B = C \circ ab^{\mathrm{T}} \tag{A.31}$$

Here $\circ$ is the Hadamard product and I° is the identity matrix with respect to the Hadamard product. Then we can write:

$$\Omega^i \omega \omega^{\mathrm{T}} \Omega^{\mathrm{T}j} = \Omega^i \Lambda \Omega^{\mathrm{T}j} = \Lambda \circ d^i d^{\mathrm{T}j} = \Lambda \circ D^i I^{\circ} D^j \tag{A.32}$$

And therefore:

$$\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle = \langle u^2 \rangle U \Lambda \circ \left(\sum_{i=0}^{\theta} (dd^{\mathrm{T}})^i \right) U^{\mathrm{T}} \tag{A.33}$$

$$\approx \langle u^2 \rangle U B U^{\mathrm{T}}, \tag{A.34}$$

where B is an $N \times N$ matrix whose elements are $B_{i,j} = \Lambda_{i,j}(1 - d_i d_j)^{-1}$.

A.2.3 Computing $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle$

The next step is to compute the projection component $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle$. First note that the desired output does not appear anywhere in the expression of $\langle \mathbf{X}\mathbf{X}^{\mathrm{T}} \rangle$. This means the covariance components is constant for a given ESN and input distribution. Now recall that the projection component is given by:

$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle = \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \Omega^i \omega u_{t-i} \widehat{y}_{t-i}^{\mathrm{T}}, \tag{A.35}$$

which means it depends on the desired output. To compute the memory function MC_τ, for each τ we have a different output, namely $u_{t-\tau}$. Therefore we specify the projection component for each τ with a subscript $\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau$. We can now write:

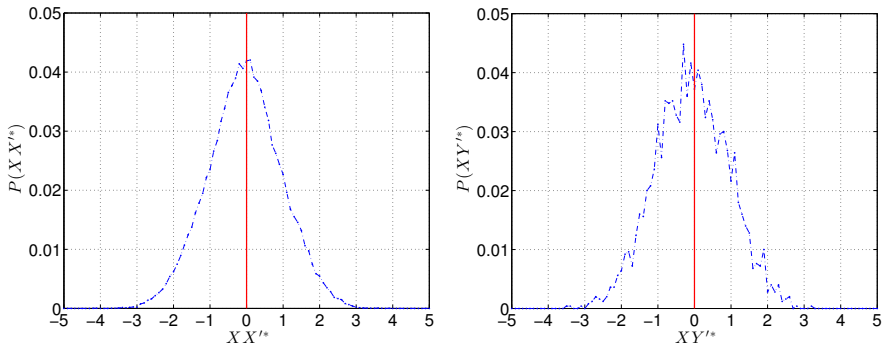
$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau = \frac{1}{T} \sum_{t=0}^{T} \sum_{i=0}^{t-1} \Omega^i \omega u_{t-i} u_{t-\tau}, \tag{A.36}$$

Again we see that for $i > \theta$, Ω^iT0 and that for all $i \neq \tau$ we have $\langle u_{t-i} u_{t-\tau} \rangle = 0$. It follows that:

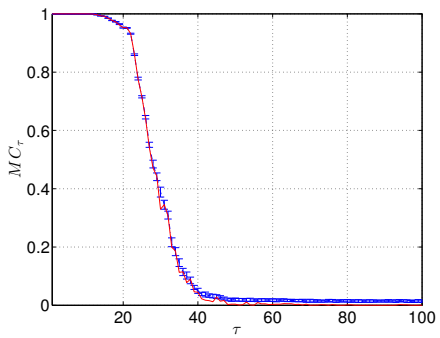
$$\langle \mathbf{X}\mathbf{Y}^{\mathrm{T}} \rangle_\tau = \Omega^\tau \omega \langle u^2 \rangle. \tag{A.37}$$

A.3 Validating the Method

In this section we calculate the memory capacity of a given ESN using our analytical solution and compare it with numerical estimations. For the purpose of demonstration, we create ten $N \times N$ reservoir weight matrices Ω and corresponding $N \times 1$ input weight matrix ω by sampling a zero-mean normal distribution with standard deviation of 1. We then rescale the weight matrix to have a spectral radius of $\lambda^* = \lambda$. For each $\{\Omega, \omega\}$ pair we run the system with an input stream of length 5000. We discard the first 2000 reservoir states and use the rest to calculate MC_τ, and repeat this experiment 10 times to calculate the average τ-delay memory capacity $\overline{MC}_\tau$. As we will see, the variance in our result is



(a) distribution of $\mathbf{XX}'^*$ (b) distribution of $\mathbf{XY}'^*$



(c) Memory curve MC_τ

Figure A.11: Agreement between analytical and simulation results for $\langle \mathbf{XX}^{\mathrm{T}} \rangle$ (a), $\langle \mathbf{XY}^{\mathrm{T}} \rangle$ (b) and the memory curve MC_τ (c).

low enough for 10 runs to give us a reliable average behavior. We choose $1 \leq \tau \leq 100$, and try the experiment with $N \in \{25, 50, 75, 100\}$ and $\lambda \in \{0.1, 0.50, 0.95\}$.

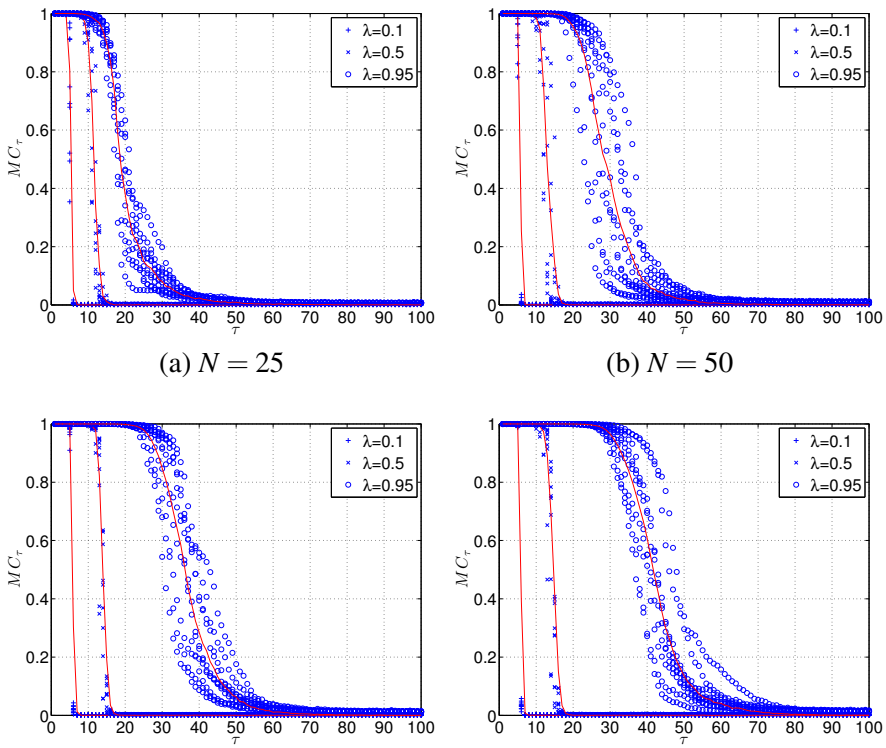
Figure A.11a and Figure A.11b illustrate the probability distribution of our simulated results for the entries of $\langle \mathbf{XX}^{\mathrm{T}} \rangle$ and $\langle \mathbf{XY}^{\mathrm{T}} \rangle$ for a sample ESN with $N = 50$ nodes and spectral radius $\lambda = 0.95$. We drove the ESN with 20 different input time series and for each input time series we calculated the matrices $\langle \mathbf{XX}^{\mathrm{T}} \rangle$, $\langle \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}} \rangle$. To look at all the entries of $\langle \mathbf{XX}^{\mathrm{T}} \rangle$ at the same time we create a dataset $\mathbf{XX}^{\mathrm{T}*}$ by shifting and rescaling each entry of $\langle \mathbf{XX}^{\mathrm{T}} \rangle$ with the corresponding analytical values so all entries map onto a zero-mean normal distribution. As expected there is no skewness in the result, suggesting that all values follow a normal distribution centered at the analytical calculations for each entry. Simi-

larly for Figure A.11b, we create a dataset $\mathbf{XY}^{\mathrm{T}*}$ by shifting and rescaling each entry of $\langle \mathbf{XY}^{\mathrm{T}} \rangle$ with the corresponding analytical values to observe that all values follow a normal distribution centered at the analytical values with no skewness.

Figure A.11c shows the complete memory curve MC_τ for the sample ESN. Our analytical results (solid red line) are in good agreement with simulation results (solid blue line). Note that our results are exact calculations and not approximation, therefore the analytical MC_τ curve also replicates the fluctuations for various values of MC_τ that are *signatures* of a particular instantiation of the ESN model.

Next, we analyze the accuracy of our analytical results with respect to changes in the reservoir size N and its spectral radius λ. Figure A.12 shows the result of this analysis, and reveals two interesting trends for accuracy and memory behavior for different N and λ. For all N and λ the analytical calculation of MC_τ agrees very well with the numerical simulation. However, as we approach $\lambda = 1$, the variance in the simulation result increases during the phase transition from $MC_\tau = 1$ to $MC_\tau = 0$, likely because the reservoir approaches the onset of chaos, i.e., $\lambda = 1$.

The behavior of the memory function also shows interesting behavior. For small $N < 50$, the transition from high to low MC_τ occurs very close to $\tau = N$, as expected from the fundamental limit $MC \leq N$. However, as the reservoir size grows, the position of the transition in MC_τ diverges from N. Note that our analytical calculation is equivalent to using infinite size training data for calculating the output weights, therefore divergence of the actual memory capacity from the bound N cannot be attributed to finite training size. Determining the reason for this discrepancy requires a more careful analysis of the memory function. We will present this analysis in Section 3.6.



(a) $N = 25$ (b) $N = 50$

(c) $N = 75$ (d) $N = 100$

Figure A.12: Sensitivity of the analytical (red solid lines) and simulation results (data markers) for the memory curve MC_τ to changes in the system size N and the spectral radius λ. The data were generated from 20 systems each driven with 20 different input streams. For all N and λ values, the analytical and simulated results are in good agreement. However, as the spectral radius approaches $\lambda = 1$ the variance of the simulated results increases, suggesting that the system is approaching the chaotic dynamical phase.

A.4 The General Formula for Optimal Readout Weights

Having developed a mental model to think about ESNs in mathematical terms and validating our approach for a simple task of memory capacity we now put together all the pieces in a succinct formula that gives a general expression for optimal readout weights of the reservoir.

Consider a discrete-time network of N nodes. The network weight matrix Ω is $N \times N$

with spectral radius $\lambda < 1$. A time-dependent scalar input signal u_t is fed to the network using the input weight vector ω. The evolution of the network state $\mathbf{x}_t$ and the output y_t is governed by

$$\mathbf{x}_{t+1} = \Omega \mathbf{x}_t + \omega u_t, \text{ and} \tag{A.38}$$

$$y_{t+1} = \Psi \mathbf{x}_{t+1}, \tag{A.39}$$

where

$$\Psi = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1} \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}} \tag{A.40}$$

is an N-dimensional column vector calculated for a desired output $\widehat{y}_t$.

Here, each column of $\mathbf{X}$ is the state of the network at time $\mathbf{x}_t$ and each column of $\widehat{\mathbf{Y}}^{\mathrm{T}}$ is the corresponding desired output at each time step. In practice it is sometimes necessary to use Tikhonov regularization to calculate the readout weights

$$\Psi = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \gamma^2 \mathbf{I}\right)^{-1} \mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}}, \tag{A.41}$$

where γ is a regularization factor that needs to be adjusted depending on Ω, ω, and u_t [206].

To derive our general formula for optimal readout weights we can simply take Equations A.30 and A.35 and rewrite them in more general and concise way. As a result calculating Ψ for a given problem requires the following input-output-dependent evaluations (Section A.4):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \sum_{i,j=0}^{\infty} \Omega^i \omega R_{uu}(i-j) \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^j, \text{ and} \tag{A.42}$$

$$\mathbf{X}\mathbf{Y}^{\mathrm{T}} = \sum_{i=0}^{\infty} \Omega^i \omega R_{u\widehat{y}}(i), \tag{A.43}$$

where $R_{uu}(i-j) = \langle u_t u_{t-(i-j)} \rangle$ and $R_{u\widehat{y}}(i-j) = \langle u_t \widehat{y}_{t-(i-j)} \rangle$ are the autocorrelation of the input and the cross-correlation of the input and target output. This may also be expressed

more generally in terms of the power spectrum of the input and the target:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \frac{1}{2T}\int_{-T}^{T} \Omega_{+}^{-1}\omega S_{uu}(f)\omega^{\mathrm{T}}\Omega_{-}^{-1}df, \tag{A.44}$$

$$\mathbf{X}\mathbf{Y}^{\mathrm{T}} = \frac{1}{2T}\int_{-T}^{T} \Omega_{+}^{-1}\omega S_{u\widehat{y}}(f)e^{if\tau}df. \tag{A.45}$$

where $\Omega_{+} = \left(\mathbf{I} - e^{if}\Omega\right)$ and $\Omega_{-} = \left(\mathbf{I} - e^{-if}\Omega\right)$, $S_{uu}(f)$ is the power spectral density of the input, and $S_{u\widehat{y}}(f)$ is the cross-spectral density of the input and the target output.

The performance can be evaluated by the mean-squared-error (MSE) as follows:

$$\langle E^2\rangle = \left\langle \left(\widehat{y}(t) - y(t)\right)^2 \right\rangle = \widehat{\mathbf{Y}}\widehat{\mathbf{Y}}^{\mathrm{T}} - \widehat{\mathbf{Y}}\mathbf{Y} \tag{A.46}$$

$$= \widehat{\mathbf{Y}}\widehat{\mathbf{Y}}^{\mathrm{T}} - \widehat{\mathbf{Y}}\mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1}\mathbf{X}\widehat{\mathbf{Y}}^{\mathrm{T}}. \tag{A.47}$$

The MSE gives us a distribution-independent upper bound on the instantaneous-squared-error through the application of Markov inequality [223]:

$$P\left[\left(\widehat{y}(t) - y(t)\right)^2 \geq a\right] \leq \frac{\langle E^2\rangle}{a}. \tag{A.48}$$

Intuitively, this means that the probability of error at time t being larger than a constant a is always less than or equal to the average error divided by a. The existence of a worst-case bound is important for engineering applications of RC.

A.5 Memory Under Correlated Input

In this section we derive the equation for memory capacity of ESN (Equation A.39) under correlated input with an exponential autocorrelation function, i.e., $R_{uu}(\tau) = e^{-\alpha\tau}$. The total memory of the system is given by the following summation over the memory function [80]:

$$\sum_{\tau} m(\tau) = \mathrm{Tr}((\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1} \sum_{\tau=0}^{\infty} (\mathbf{X}\mathbf{Y}^{\mathrm{T}})_{\tau} (\mathbf{Y}\mathbf{X}^{\mathrm{T}})_{\tau}). \tag{A.49}$$

where $\mathbf{Y}$ is the input with lag τ, $u_{t-\tau}$.

Computing $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ requires the evaluation of:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \sum_{i,j=0}^{\infty} \Omega^{i} \omega R_{uu}(i-j) \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^{j}. \tag{A.50}$$

This assumes an even correlation function, i.e., $R_{uu}(i-j) = R_{uu}(j-i)$. For numerical computation it is more convenient to perform the calculation as follows:

$$\begin{aligned} \mathbf{X}\mathbf{X}^{\mathrm{T}} &= \mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j} + \mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \leq j} - \mathbf{X}\mathbf{X}^{\mathrm{T}}_{i=j} \\ &= \mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j} + \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j}\right)^{\mathrm{T}} - \mathbf{X}\mathbf{X}^{\mathrm{T}}_{i=j}, \end{aligned} \tag{A.51}$$

where $\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j}$ is a partial sum of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ satisfying $i \geq j$, $\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \leq j} = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j}\right)^{\mathrm{T}}$ is a partial sum of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ satisfying $i \leq j$, and $\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i=j}$ is a partial sum of $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ satisfying $i = j$, which is double counted and must be subtracted. We can substitute $\tau = |i-j|$ and evaluate $\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i \geq j}$ and $\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i=j}$ as follows:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i\geq j} = \sum_{i,\tau=0}^{\infty} \Omega^i \omega R_{uu}(\tau) \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^{i+\tau} \tag{A.52}$$

$$= \sum_{i,\tau=0}^{\infty} \Omega^i \omega e^{-\alpha\tau} \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^{i+\tau} \tag{A.53}$$

$$= \sum_{i=0}^{\infty} \Omega^i \omega \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^i \sum_{\tau=0}^{\infty} (e^{-\alpha} \Omega^{\mathrm{T}})^{\tau} \tag{A.54}$$

$$= \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}} (\mathbf{I} - e^{-\alpha} \Omega^{\mathrm{T}})^{-1}, \tag{A.55}$$

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}_{i=j} = \sum_{i=0}^{\infty} \Omega^i \omega R_{uu}(0) \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^i \tag{A.56}$$

$$= \sum_{i=0}^{\infty} \Omega^i \omega \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^i \tag{A.57}$$

$$= \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}. \tag{A.58}$$

Here $\mathbf{B}$ is an $N \times N$ matrix whose elements are $B_{i,j} = \Lambda_{i,j}(1 - d_i d_j)^{-1}$. Here the trick is that $\bar{\omega} = \mathbf{U}^{-1}\omega$ takes the input to the basis of the connection matrix Ω allowing the dynamics to be described by the powers of the eigenvalues of Ω, i.e., $\mathbf{D}$. Since $\mathbf{D}$ is symmetric we can use the matrix identity $\mathbf{D}\Lambda\mathbf{D} = \Lambda \circ \mathbf{d}\mathbf{d}^{\mathrm{T}}$, where $\mathbf{d}$ is the main diagonal of $\mathbf{D}$. Summing over the powers of $\mathbf{D}$ gives us $\sum_{i=0}^{\infty} \Omega^i \omega \omega^{\mathrm{T}} (\Omega^{\mathrm{T}})^i = \mathbf{U}\mathbf{B}\mathbf{U}^{\mathrm{T}}$.

The covariance of the network states and the expected output is given by:

$$\mathbf{X}\mathbf{Y}^{\mathrm{T}}_{\tau} = \sum_i \Omega^i \omega R(|i - \tau|) = \sum_i \Omega^i \omega e^{-\alpha|i-\tau|}. \tag{A.59}$$

By plugging these two components into the memory equation we can directly calculate the memory capacity without simulating the ESN.

References

[1] J. W. Lawson and D. H. Wolpert, *Adaptive programming of unconventional nano-architectures*, Journal of Computational and Theoretical Nanoscience **3** (2006), 272–279.

[2] J. Tour, W. Van Zandt, C. Husband, S. Husband, L. Wilson, P. Franzon, and D. Nackashi, *Nanocell logic gates for molecular computing*, Nanotechnology, IEEE Transactions on **1** (2002), 100–109.

[3] S. Forrest, *Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks*, MIT Press, Cambridge, MA, USA, 1991.

[4] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, J. T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss, *Amorphous computing*, Communications of the ACM **43** (2000), 74–82.

[5] H. Jaeger and H. Haas, *Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication*, Science **304** (2004), 78–80.

[6] F. Wyffels and B. Schrauwen, *A comparative study of reservoir computing strategies for monthly time series prediction*, Neurocomputing **73** (2010), 1958–1964.

[7] S. Wang, X.-J. Yang, , and C.-J. Wei, *Harnessing nonlinearity by sigmoid-wavelet hybrid echo state networks (swhesn)*, in The 6th World Congress on Intelligent Control and Automation (WCICA 2006), Vol. 1, June 2006, pp. 3014–3018.

[8] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, New York, NY, USA, 2004.

[9] S. Haykin, *Neural Networks and Learning Machines (3rd Edition)*, Pearson Education, Inc., New York, NY, 2009.

[10] M. Fiers, B. Maes, and P. Bienstman, *Dynamics of coupled cavities for optical reservoir computing*, in Proceedings of the 2009 Annual Symposium of the IEEE Photonics Benelux Chapter, (S. Beri, P. Tassin, G. Craggs, X. Leijtens, and J. Danckaert, eds.) VUB Press, 2009, pp. 129–132.

[11] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, *Information processing using a single dynamical node as complex system*, Nature Communications **2** (2011), 468.

[12] H. O. Sillin, R. Aguilera, H.-H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski, *A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing*, Nanotechnology **24** (2013), 384004.

[13] J. Burger and C. Teuscher, *Volatile memristive devices as short-term memory in a neuromorphic learning architecture*, in Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 2014) IEEE Press, 2014, pp. 104–109.

[14] A. Goudarzi, M. R. Lakin, D. Stefanovic, and C. Teuscher, *A model for variation- and fault-tolerant digital logic using self-assembled nanowire architectures*, in Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH) IEEE Press, 2014, pp. 116–121.

[15] A. Goudarzi, M. Lakin, and D. Stefanovic, *Reservoir computing approach to robust computation using unreliable nanoscale networks*, in Unconventional Computation and Natural Computation, (O. H. Ibarra, L. Kari, and S. Kopecki, eds.), Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 164–176.

[16] M. Haselman and S. Hauck, *The future of integrated circuits: A survey of nanoelectronics*, Proceedings of the IEEE **98** (2010), 11–38.

[17] V. V. Zhirnov and R. K. Cavin III, *Microsystems for Bioelectronics, Second Edition: Scaling and Performance Limits*, Micro and Nano Technologies, William Andrew, 2015.

[18] J. P. Crutchfield, W. L. Ditto, and S. Sinha, *Introduction to focus issue: Intrinsic and designed computation: Information processing in dynamical systems—beyond the digital hegemony*, Chaos: An Interdisciplinary Journal of Nonlinear Science **20** (2010), 037101.

[19] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams, *Nanoscale molecular-switch crossbar circuits*, Nanotechnology **14** (2003), 462.

[20] G. Snider, *Computing with hysteretic resistor crossbars*, Applied Physics A **80** (2005), 1165–1172.

[21] P. Xu, S.-H. Jeon, H.-T. Chen, H. Luo, G. Zou, Q. Jia, M. Anghel, C. Teuscher, D. J. Williams, B. Zhang, X. Han, and H.-L. Wang, *Facile synthesis and electrical properties of silver wires through chemical reduction by polyaniline*, The Journal of Physical Chemistry C **114** (2010), 22147–22154.

[22] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, M. Aono, and J. K. Gimzewski, *Emergent criticality in complex Turing B-type atomic switch networks*, Advanced Materials **24** (2012), 286–293.

[23] Semiconductor Industry Association, *International technology roadmap for semiconductors (ITRS) 2.0, beyond CMOS,* `https://www.dropbox.com/sh/3jfh5fq634b5yqu/AACxgvNR8zhcTYt2cp2nj0W0a/6_2015%20ITRS%202.0%20Beyond%20CMOS.pdf?dl=0` Technical report, 2015.

[24] W. Maass, T. Natschläger, and H. Markram, *Real-time computing without stable states: a new framework for neural computation based on perturbations*, Neural computation **14** (2002), 2531–60.

[25] X.-J. Wang, *Synaptic reverberation underlying mnemonic persistent activity*, Trends in Neurosciences **24** (2001), 455–463.

[26] J. Fuster and G. Alexander, *Neuron activity related to short-term memory*, Science **173** (1971), 652–654.

[27] K. Kubota and H. Niki, *Prefrontal cortical unit activity and delayed alternation performance in monkeys*, Journal of Neurophysiology **34** (1971), 337–347.

[28] S. Funahashi, C. J. Bruce, and P. S. Goldman-Rakic, *Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex*, Journal of Neurophysiology **61**.

[29] G. Rainer, W. F. Asaad, and E. K. Miller, *Memory fields of neurons in the primate prefrontal cortex*, Proceedings of the National Academy of Sciences **95** (1998), 15008–15013.

[30] R. Romo, C. D. Brody, A. Hernandez, and L. Lemus, *Neuronal correlates of parametric working memory in the prefrontal cortex*, Nature **399** (1999), 470–473.

[31] G. Major and D. Tank, *Persistent neural activity: prevalence and mechanisms*, Current Opinion in Neurobiology **14** (2004), 675 – 684.

[32] C. D. Brody, R. Romo, and A. Kepecs, *Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations*, Current Opinion in Neurobiology **13** (2003), 204 – 211.

[33] P. Dominey, M. Arbib, and J.-P. Joseph, *A model of corticostriatal plasticity for learning oculomotor associations and sequences*, Journal of Cognitive Neuroscience **7** (1995), 311–336.

[34] X. Hinaut and P. F. Dominey, *Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing*, PLoS ONE **8** (2013), e52946.

[35] H. Jaeger *The "echo state" approach to analysing and training recurrent neural networks* Technical Report GMD Rep. 148, St. Augustin: German National Research Center for Information Technology, 2001.

[36] B. Schrauwen, D. Verstraeten, and J. V. Campenhout, *An overview of reservoir computing: theory, applications and implementations*, in Proceedings of the 15th European Symposium on Artificial Neural Networks, 2007, pp. 471–482.

[37] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, *An experimental unification of reservoir computing methods*, Neural Networks **20** (2007), 391–403.

[38] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, *Optoelectronic reservoir computing*, Scientific Reports **2** (2012), 1–6.

[39] H. Jaeger, *Adaptive nonlinear system identification with echo state networks*, in Advances in Neural Information Processing Systems 14, 2002, pp. 593–600.

[40] S. Dasgupta, F. Wörgötter, and P. Manoonpong, *Information theoretic self-organised adaptation in reservoirs for temporal memory tasks*, in Engineering Applications of Neural Networks, (C. Jayne, S. Yue, and L. Iliadis, eds.), Vol. 311 of *Communications in Computer and Information Science*, Springer Berlin Heidelberg, 2012, pp. 31–40.

[41] L. Chudy and I. Farkas, *Prediction of chaotic time-series using dynamic cell structures and local linear models*, Neural Network World **8** (1998), 481–489.

[42] J. McNames, J. A. K. Suykens, and J. Vandewalle, *Winning entry of the ku leuven time-series prediction competition*, International Journal of Bifurcation and Chaos **9** (1999), 1485–1500.

[43] F. Gers, D. Eck, and J. F. Schmidhuber *Applying LSTM to time series predictable through time-window approaches* Technical Report IDSIA-22-00, IDSIA/USI-SUPSI, 2000.

[44] H. Jaeger *Short term memory in echo state networks* Technical Report GMD Report 152, GMD-Forschungszentrum Informationstechnik, 2002.

[45] H. Jaeger *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach* Technical Report GMD Report 159, German National Research Center for Information Technology, St. Augustin-Germany, 2002.

[46] J. Butcher, D. Verstraeten, B. Schrauwen, C. Day, and P. Haycock, *Reservoir computing and extreme learning machines for non-linear time-series data analysis*, Neural Networks **38** (2013), 76–89.

[47] R. Penrose, *A generalized inverse for matrices*, Mathematical Proceedings of the Cambridge Philosophical Society **51** (1955), 406–413.

[48] A. Rodan and P. Tiňo, *Minimum complexity echo state network*, Neural Networks, IEEE Transactions on **22** (2011), 131–144.

[49] A. Atiya and A. Parlos, *New results on recurrent network training: unifying the algorithms and accelerating convergence*, Neural Networks, IEEE Transactions on **11** (2000), 697–709.

[50] J. Steil, *Backpropagation-decorrelation: online recurrent learning with O(N) complexity*, in Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, Vol. 2, 2004, pp. 843–848 vol.2.

[51] S. Ding, H. Zhao, Y. Zhang, X. Xu, and R. Nie, *Extreme learning machine: algorithm, theory and applications*, Artificial Intelligence Review **25** (2013), 1–13.

[52] M. Hermans and B. Schrauwen, *Recurrent kernel machines: Computing with infinite echo state networks*, Neural Computation **24** (2011), 104–133.

[53] M. Buehner and P. Young, *A tighter bound for the echo state property*, Neural Networks, IEEE Transactions on **17** (2006), 820–824.

[54] M. C. Ozturk, D. Xu, and J. C. Príncipe, *Analysis and design of echo state networks*, Neural Computation **19** (2007), 111–138.

[55] J. Boedecker, O. Obst, N. M. Mayer, and M. Asada, *Initialization and self-organized optimization of recurrent neural network connectivity*, HFSP Journal **3** (2009), 340–349.

[56] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, *Improving reservoirs using intrinsic plasticity*, Neurocomputing **71** (2008), 1159–1171.

[57] J. Larsen, *A generalization error estimate for nonlinear systems*, in Neural Networks for Signal Processing, 1992. II., Proceedings of the 1992 IEEE Workshop on, 1992, pp. 29–38.

[58] J. Larsen, L. Hansen, C. Svarer, and M. Ohlsson, *Design and regularization of neural networks: the optimal use of a validation set*, in Neural Networks for Signal Processing, 1996. VI. Proceedings of the 1996 IEEE Workshop on, 1996, pp. 62–71.

[59] J. Larsen and L. Hansen, *Empirical generalization assessment of neural network models*, in Neural Networks for Signal Processing, 1995. V. Proceedings of the 1995 IEEE Workshop on, 1995, pp. 30–39.

[60] J. Moody, *Prediction risk and architecture selection for neural networks*, in From Statistics to Neural Networks, (V. Cherkassky, J. Friedman, and H. Wechsler, eds.), Vol. 136 of *NATO ASI Series*, Springer Berlin Heidelberg, 1994, pp. 147–165.

[61] S. A. Kauffman, *Coevolution to the edge of chaos: Coupled fitness landscapes, poised states, and coevolutionary avalanches*, Journal of Theoretical Biology **149** (1991), 467–505.

[62] C. Langton, *Computation at the edge of chaos: Phase transitions and emergent computation*, Physica D **42** (1990), 12–37.

[63] D. R. Chialvo, *Critical brain networks*, Physica A: Statistical Mechanics and its Applications **340** (2004), 756–765.

[64] T. Rohlf and S. Bornholdt, *Criticality in random threshold networks: annealed approximation and beyond*, Physica A: Statistical Mechanics and its Applications **310** (2002), 245–259.

[65] T. Rohlf, N. Gulbahce, and C. Teuscher, *Damage spreading and criticality in finite random dynamical networks*, Physical Review Letters **99** (2007), 248701.

[66] A. Levina, J. M. Herrmann, and T. Geisel, *Dynamical synapses causing self-organized criticality in neural networks*, Nature Physics **3** (2007), 857–860.

[67] A. Goudarzi, C. Teuscher, N. Gulbahce, and T. Rohlf, *Emergent criticality through adaptive information processing in Boolean networks*, Physical Review Letters **108** (2012), 128702.

[68] J. Boedecker, O. Obst, J. Lizier, N. Mayer, and M. Asada, *Information processing in echo state networks at the edge of chaos*, Theory in Biosciences **131** (2012), 205–2013.

[69] L. de Arcangelis and H. J. Herrmann, *Learning as a phenomenon occurring in a critical state*, Proceedings of the National Academy of Sciences **107** (2010), 3977–81.

[70] O. Kinouchi and M. Copelli, *Optimal dynamical range of excitable networks at criticality*, Nature Physics **2** (2006), 348–351.

[71] L. Büsing, B. Schrauwen, and R. Legenstein, *Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons.*, Neural Computation **22** (2010), 1272–1311.

[72] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, *Memory versus non-linearity in reservoirs*, in Neural Networks (IJCNN), The 2010 International Joint Conference on, July 2010, pp. 1–8.

[73] M. Kaiser and C. C. Hilgetag, *Development of multi-cluster cortical networks by time windows for spatial growth*, Neurocomputing **70** (2007), 1829–1832.

[74] ———, *Modelling the development of cortical systems networks*, Neurocomputing **58-60** (2004), 297–302.

[75] Q. Song and Z. Feng, *Effects of connectivity structure of complex echo state network on its prediction performance for nonlinear time series*, Neurocomputing **73** (2010), 2177–2185.

[76] A. Medina, I. Matta, and J. Byers, *On the origin of power laws in internet topologies*, SIGCOMM Computer Communication Review **30** (2000), 18–28.

[77] Z. Deng and Y. Zhang, *Collective behavior of a small-world recurrent neural system with scale-free distribution*, Neural Networks, IEEE Transactions on **18** (2007), 1364–1375.

[78] ———, *Complex systems modeling using scale-free highly-clustered echo state network*, in Neural Networks, 2006. IJCNN '06. International Joint Conference on, 2006, pp. 3128–3135.

[79] S. Ganguli, D. Huh, and H. Sompolinsky, *Memory traces in dynamical systems*, Proceedings of the National Academy of Sciences **105** (2008), 18970–18975.

[80] O. L. White, D. D. Lee, and H. Sompolinsky, *Short-term memory in orthogonal neural networks*, Physical Review Letters **92** (2004), 148102.

[81] V. Vapnik, *The nature of statistical learning theory*, Springer, New York, NY, 2000.

[82] L. Valiant, *A theory of the learnable*, Communications of the ACM **27** (1984), 1134–1142.

[83] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological review **65** (1958), 386–408.

[84] A. B. J. Novikoff, *On convergence proofs on perceptrons*, Symposium on the Mathematical Theory of Automata **12** (1962), 615–622.

[85] W. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biology **5** (1943), 115–133.

[86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, in Parallel Distributed Processing, (D. E. Rumelhart and J. L. McClelland, eds.), Vol. 1, MIT Press, Cambridge, MA, 1986, Chap. 8, pp. 318–362.

[87] J.-C. Delvenne, *What is a universal computing machine?*, Applied Mathematics and Computation **215** (2009), 1368–1374.

[88] C. Moore, *Unpredictability and undecidability in dynamical systems*, Physical Review Letters **64** (1990), 2354–2357.

[89] C. Moore, *Generalized shifts: unpredictability and undecidability in dynamical systems*, Nonlinearity **4** (1991), 199.

[90] P. Koiran and C. Moore, *Closed-form analytic maps in one and two dimensions can simulate universal Turing machines*, Theoretical Computer Science **210** (1999), 217–223.

[91] P. Koiran, M. Cosnard, and M. Garzon, *Computability with low-dimensional dynamical systems*, Theoretical Computer Science **132** (1994), 113–128.

[92] M. S. Branicky, *Universal computation and other capabilities of hybrid and continuous dynamical systems*, Theoretical Computer Science **138** (1995), 67–100.

[93] O. Bournez and M. Cosnard, *On the computational power of dynamical systems and hybrid systems*, Theoretical Computer Science **168** (1996), 417–459.

[94] H. Siegelmann, *Computation beyond the Turing limit*, Science **238** (1995), 632–637.

[95] M. Ercsey-Ravasz and Z. Toroczkai, *Optimization hardness as transient chaos in an analog approach to constraint satisfaction*, Nature Physics **7** (2011), 966–970.

[96] K. Murali, S. Sinha, W. L. Ditto, and A. R. Bulsara, *Reliable logic circuit elements that exploit nonlinearity in the presence of a noise floor*, Physical Review Letters **102** (2009), 104101.

[97] K. Murali, A. Miliotis, W. L. Ditto, and S. Sinha, *Logic from nonlinear dynamical evolution*, Physics Letters A **373** (2009), 1346–1351.

[98] S. Sinha and W. L. Ditto, *Dynamics based computation*, Physical Review Letters **81** (1998), 2156–2159.

[99] M. Zanin, D. Papo, and S. Boccaletti, *Computing with complex-valued networks of phase oscillators*, EPL (Europhysics Letters) **102** (2013), 40007.

[100] M. Zanin, D. Papo, I. Sendiña Nadal, and S. Boccaletti, *Computation as an emergent feature of adaptive synchronization*, Physical Review E **84** (2011), 060102.

[101] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro, *Programmable and autonomous computing machine made of biomolecules*, Nature **414** (2001), 430–434.

[102] M. Kahan, B. Gil, R. Adar, and E. Shapiro, *Towards molecular computers that operate in a biological environment*, Physica D: Nonlinear Phenomena **237** (2008), 1165–1172.

[103] C.-H. Lu, B. Willner, and I. Willner, *DNA nanotechnology: From sensing and DNA machines to drug-delivery systems*, ACS Nano **7** (2013), 8320–8332.

[104] N. C. Seeman, J. M. Rosenberg, and A. Rich, *Sequence-specific recognition of double helical nucleic acids by proteins*, Proceedings of the National Academy of Sciences **73** (1976), 804–808.

[105] J. H. Chen and N. C. Seeman, *Synthesis from DNA of a molecule with the connectivity of a cube*, Nature **350** (1991), 631–633.

[106] W. B. Sherman and N. C. Seeman, *A precisely controlled DNA biped walking device*, Nano Letters **4** (2004), 1203–1207.

[107] N. C. Seeman and P. S. Lukeman, *Nucleic acid nanostructures: bottom-up control of geometry on the nanoscale*, Reports on Progress in Physics **68** (2005), 237–270.

[108] N. C. Seeman, *From genes to machines: DNA nanomechanical devices*, Trends in Biochemical Sciences **30** (2005), 119–125.

[109] D. Soloveichik, G. Seelig, and E. Winfree, *DNA as a universal substrate for chemical kinetics*, Proceedings of the National Academy of Sciences **107** (2010), 5393–5398.

[110] D. Y. Zhang and G. Seelig, *Dynamic DNA nanotechnology using strand-displacement reactions*, Nature Chemistry **3** (2011), 103–113.

[111] E. Winfree, *Algorithmic self-assembly of DNA*, PhD thesis, California Institute of Technology, June 1998.

[112] D. Soloveichik and E. Winfree, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), 1544–1569.

[113] E. Winfree, *Self-healing tile sets*, in Nanotechnology: Science and Computation, (J. Chen, N. Jonoska, and G. Rozenberg, eds.), Natural Computing Series, Springer Berlin Heidelberg, 2006, pp. 55–78.

[114] E. Winfree and R. Bekbolatov, *Proofreading tile sets: Error correction for algorithmic self-assembly*, in DNA Computing, (J. Chen and J. Reif, eds.), Vol. 2943 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2004, pp. 126–144.

[115] S. Summers, *Reducing tile complexity for the self-assembly of scaled shapes through temperature programming*, Algorithmica **63** (2012), 117–136.

[116] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*, Nature **407** (2000), 493–496.

[117] L. Adleman, *Molecular computation of solutions to combinatorial problems*, Science **266** (1994), 1021–1024.

[118] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman, *Solution of a 20-variable 3-SAT problem on a DNA computer*, Science **296** (2002), 499–502.

[119] M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic, *Deoxyribozyme-based logic gates*, Journal of the American Chemical Society **124** (2002), 3555–3561.

[120] L. Qian and E. Winfree, *A simple DNA gate motif for synthesizing large-scale circuits*, Journal of The Royal Society Interface **8** (2011), 1281–1297.

[121] ———, *Scaling up digital circuit computation with DNA strand displacement cascades*, Science **332** (2011), 1196–1201.

[122] R. Pei, E. Matamoros, M. Liu, D. Stefanovic, and M. N. Stojanovic, *Training a molecular automaton to play a game*, Nature Nanotechnology **5** (2010), 773–777.

[123] M. N. Stojanovic and D. Stefanovic, *A deoxyribozyme-based molecular automaton.*, Nature Biotechnology **21** (2003), 1069.

[124] Y. Benenson, *Biomolecular computing systems: principles, progress and potential*, Nature Reviews Genetics **13** (2012), 455–468.

[125] T. A. P. F. Doll, S. Raman, R. Dey, and P. Burkhard, *Nanoscale assemblies and their biomedical applications*, Journal of the Royal Society Interface **10** (2013), 20120740.

[126] S. Lee, K. Saito, H.-R. Lee, M. J. Lee, Y. Shibasaki, Y. Oishi, and B.-S. Kim, *Hyperbranched double hydrophilic block copolymer micelles of poly(ethylene oxide) and polyglycerol for pH-responsive drug delivery*, Biomacromolecules **13** (2012), 1190–1196.

[127] S. M. Douglas, I. Bachelet, and G. M. Church, *A logic-gated nanorobot for targeted transport of molecular payloads*, Science **335** (2012), 831–834.

[128] C. J. Delebecque, A. B. Lindner, P. A. Silver, and F. A. Aldaye, *Organization of intracellular reactions with rationally designed RNA assemblies*, Science **333** (2011), 470–474.

[129] Y. Krishnan and M. Bathe, *Designer nucleic acids to probe and program the cell*, Trends in Cell Biology **22** (2012), 624–633.

[130] C. Bancroft, T. Bowler, B. Bloom, and C. T. Clelland, *Long-term storage of information in DNA*, Science **293** (2001), 1763–1765.

[131] J. H. Reif, T. H. LaBean, M. Pirrung, V. S. Rana, B. Guo, C. Kingsford, and G. S. Wickham, *Experimental construction of very large scale DNA databases with associative search capability*, in DNA Computing, (N. Jonoska and N. C. Seeman, eds.), Vol. 2340 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2002, pp. 231–247.

[132] J. Reif, M. Hauser, M. Pirrung, and T. LaBean, *Application of biomolecular computing to medical science: A biomolecular database system for storage, processing, and retrieval of genetic information and material*, in Complex Systems Science in Biomedicine, (T. Deisboeck and J. Kresh, eds.), Topics in Biomedical Engineering International Book Series, Springer US, 2006, pp. 701–735.

[133] G. M. Church, Y. Gao, and S. Kosuri, *Next-generation digital information storage in DNA*, Science **337** (2012), 1628.

[134] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, *Towards practical, high-capacity, low-maintenance information storage in synthesized DNA*, Nature **494** (2013), 77–80.

[135] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences **79** (1982), 2554–2558.

[136] A. Hjelmfelt, E. D. Weinberger, and J. Ross, *Chemical implementation of neural networks and Turing machines*, Proceedings of the National Academy of Sciences **88** (1991), 10983–10987.

[137] A. Hjelmfelt, F. W. Schneider, and J. Ross, *Pattern recognition in coupled chemical kinetic systems*, Science **260** (1993), 335–337.

[138] J.-P. Laplante, M. Pemberton, A. Hjelmfelt, and J. Ross, *Experiments on pattern recognition by chemical kinetics*, The Journal of Physical Chemistry **99** (1995), 10063–10065.

[139] A. P. Mills Jr., B. Yurke, and P. M. Platzman, *Article for analog vector algebra computation*, Biosystems **52** (1999), 175–180.

[140] A. P. Mills Jr., M. Turberfield, A. J. Turberfield, B. Yurke, and P. M. Platzman, *Experimental aspects of DNA neural network computation.*, Soft Computing **5** (2001), 10–18.

[141] L. Qian, E. Winfree, and J. Bruck, *Neural network computation with DNA strand displacement cascades*, Nature **475** (2011), 368–372.

[142] H. Jiang, M. Riedel, and K. Parhi, *Digital signal processing with molecular reactions*, Design Test of Computers, IEEE **29** (2012), 21–31.

[143] T. Sohka, R. A. Heins, R. M. Phelan, J. M. Greisler, C. A. Townsend, and M. Ostermeier, *An externally tunable bacterial band-pass filter*, Proceedings of the National Academy of Sciences **106** (2009), 10135–10140.

[144] M. N. Win and C. D. Smolke, *Higher-order cellular information processing with synthetic RNA devices*, Science **322** (2008), 456–460.

[145] H. Jiang, S. A. Salehi, M. D. Riedel, and K. K. Parhi, *Discrete-time signal processing with DNA*, ACS Synthetic Biology **2** (2013), 245–254.

[146] H. Jiao, Y. Zhong, L. Zhang, and P. Li, *Unsupervised remote sensing image classification using an artificial DNA computing*, in Natural Computation (ICNC), 2011 Seventh International Conference on, Vol. 3, 2011, pp. 1341–1345.

[147] H. W. H. van Roekel, L. H. H. Meijer, S. Masroor, Z. C. Félix Garza, A. Estèvez-Torres, Y. Rondelez, A. Zagaris, M. A. Peletier, P. A. J. Hilbers, and T. F. A. de Greef, *Automated design of programmable enzyme-driven DNA circuits*, ACS Synthetic Biology **4** (2015), 735–745.

[148] K. Montagne, R. Plasson, Y. Sakai, T. Fujii, and Y. Rondelez, *Programming an in vitro DNA oscillator using a molecular networking strategy*, Molecular Systems Biology **7** (2011), 466–466.

[149] A. Baccouche, K. Montagne, A. Padirac, T. Fujii, and Y. Rondelez, *Dynamic DNA-toolbox reaction circuits: A walkthrough*, Methods **67** (2014), 234–249.

[150] B. Yordanov, J. Kim, R. L. Petersen, A. Shudy, V. V. Kulkarni, and A. Phillips, *Computational design of nucleic acid feedback control circuits*, ACS Synthetic Biology **3** (2014), 600–616.

[151] M. R. Lakin, A. Minnich, T. Lane, and D. Stefanovic, *Towards a biomolecular learning machine*, in Unconventional Computation and Natural Computation 2012, (J. Durand-Lose and N. Jonoska, eds.), Vol. 7445 of *Lecture Notes in Computer Science* Springer-Verlag, 2012, pp. 152–163.

[152] P. Banda, C. Teuscher, and D. Stefanovic, *Training an asymmetric signal perceptron through reinforcement in an artificial chemistry*, Journal of The Royal Society Interface **11** (2014), 20131100.

[153] S. Modi, C. Nizak, S. Surana, S. Halder, and Y. Krishnan, *Two DNA nanomachines map pH changes along intersecting endocytic pathways inside the same cell*, Nature Nanotechnology **8** (2013), 459–467.

[154] K.-A. Yang, M. Barbu, M. Halim, P. Pallavi, B. Kim, D. M. Kolpashchikov, S. Pecic, S. Taylor, T. S. Worgall, and M. N. Stojanovic, *Recognition and sensing of low-epitope targets via ternary complexes with oligonucleotides and synthetic receptors*, Nature Chemistry **6** (2014), 1003–1008.

[155] S. Beyer, W. Dittmer, and F. Simmel, *Design variations for an aptamer-based DNA nanodevice.*, Journal of Biomedical Nanotechnology **1** (2005), 96–101.

[156] S. Beyer and F. C. Simmel, *A modular DNA signal translator for the controlled release of a protein by an aptamer*, Nucleic Acids Research **34** (2006), 1581–1587.

[157] E. Shapiro and B. Gil, *RNA computing in a living cell*, Science **322** (2008), 387–388.

[158] R. P. Feynman *Engineering and science*, 1960.

[159] J. R. Heath and M. A. Ratner, *Molecular electronics.*, Physics Today **56** (2003), 43.

[160] J. Xiang, W. Lu, Y. Hu, Y. Wu, H. Yan, and C. M. Lieber, *Ge/Si nanowire heterostructures as high-performance field-effect transistors*, Nature **441** (2006), 489–493.

[161] J. E. Green, J. Wook Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu, Y. Shik Shin, H.-R. Tseng, J. F. Stoddart, and J. R. Heath, *A 160-kilobit molecular electronic memory patterned at 1011 bits per square centimetre*, Nature **445** (2007), 414–417.

[162] K. Terabe, T. Hasegawa, T. Nakayama, and M. Aono, *Quantized conductance atomic switch*, Nature **433** (2005), 47–50.

[163] J. Yao, H. Yan, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber, *Nanowire nanocomputer as a finite-state machine*, Proceedings of the National Academy of Sciences **111** (2014), 2431–2435.

[164] A. Schmid and Y. Leblebici, *A modular approach for reliable nanoelectronic and very-deep submicron circuit design based on analog neural network principles*, in Nanotechnology, 2003. IEEE-NANO 2003. 2003 Third IEEE Conference on, Vol. 2, Aug 2003, pp. 647–650 vol. 2.

[165] J. Dai, L. Wang, and F. Jain, *Analysis of defect tolerance in molecular crossbar electronics*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **17** (2009), 529–540.

[166] L. Žaloudek and L. Sekanina, *Cellular automata-based systems with fault-tolerance*, Natural Computing **11** (2012), 673–685.

[167] A. H. Tran, S. Yanushkevich, S. Lyshevski, and V. Shmerko, *Design of neuromorphic logic networks and fault-tolerant computing*, in Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on, Aug 2011, pp. 457–462.

[168] W. Zhang and N.-J. Wu, *CMOL-based cellular neural networks and parallel processor for future image processing*, in Nanotechnology, 2008. NANO '08. 8th IEEE Conference on, Aug 2008, pp. 737–740.

[169] M. Anghel, C. Teuscher, and H.-L. Wang, *Adaptive learning in random linear nanoscale networks*, in Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on, Aug 2011, pp. 445–450.

[170] T. Szkopek, V. P. Roychowdhury, D. A. Antoniadis, and J. N. Damoulakis, *Physical fault tolerance of nanoelectronics*, Physical Review Letters **106** (2011), 176801.

[171] C. Koch and I. Segev, *The role of single neurons in information processing*, Nature Neuroscience **3** (2000), 1171–1177.

[172] J. C. Magee, *Dendritic integration of excitatory synaptic input*, Nature Reviews Neuroscience **1** (2000), 181–190.

[173] M. Margulis and C.-M. Tang, *Temporal integration can readily switch between sublinear and supralinear summation*, Journal of Neurophysiology **79** (1998), 2809–2813.

[174] R. D. Cazé, M. Humphries, and B. Gutkin, *Passive dendrites enable single neurons to compute linearly non-separable functions*, PLoS Comput Biol **9** (2013), e1002867.

[175] C. L. Giles and T. Maxwell, *Learning, invariance, and generalization in high-order neural networks*, Applied Optics **26** (1987), 4972–4978.

[176] R. Durbin and D. E. Rumelhart, *Product units: A computationally powerful and biologically plausible extension to backpropagation networks*, Neural Computation **1** (1989), 133–142.

[177] C. J. McAdams and J. H. R. Maunsell, *Effects of attention on orientation-tuning functions of single neurons in macaque cortical area V4*, The Journal of Neuroscience **19** (1999), 431–441.

[178] R. Andersen, G. Essick, and R. Siegel, *Encoding of spatial location by posterior parietal neurons*, Science **230** (1985), 456–458.

[179] S. Treue and J. H. R. Maunsell, *Effects of attention on the processing of motion in macaque middle temporal and medial superior temporal visual cortical areas*, The Journal of Neuroscience **19** (1999), 7591–7602.

[180] F. Gabbiani, H. G. Krapp, C. Koch, and G. Laurent, *Multiplicative computation in a visual neuron sensitive to looming*, Nature **420** (2002), 320–324.

[181] G. Geiger, *Optomotor responses of the fly* musca domestica *to transient stimuli of edges and stripes*, Kybernetik **16** (1974), 37–43.

[182] K. Götz, *The optomotor equilibrium of the drosophila navigation system*, Journal of comparative physiology **99** (1975), 187–210.

[183] J. L. Peña and M. Konishi, *Auditory spatial receptive fields created by multiplication*, Science **292** (2001), 249–252.

[184] Y. Shin and J. Ghosh, *Ridge polynomial networks*, Neural Networks, IEEE Transactions on **6** (1995), 610–622.

[185] T. Toyoizumi, *Nearly extensive sequential memory lifetime achieved by coupled nonlinear neurons*, Neural Computation **24** (2012), 2678–2699.

[186] S. Ganguli and H. Sompolinsky, *Short-term memory in neuronal networks through dynamical compressed sensing*, in Advances in Neural Information Processing Systems 23, (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), 2010, pp. 667–675.

[187] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation **9** (1997), 1735–1780.

[188] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Gated feedback recurrent neural networks*, in Volume 37: Proceedings of The 32nd International Conference on Machine Learning, Vol. 37, 2015.

[189] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, JMLR W & CP **28** (2013), 1310–1318.

[190] J. Mertens and I. Sutskever, *Learning recurrent neutral networks with hessian-free optimization*, in Proceedings of the 28^{th} International Conference on Machine Learning, Vellevue, WA, USA, 2011.

[191] A. Graves, G. Wayne, and I. Danihelka, *Neural turing machines*, arXiv:1410.5401 [cs.NE].

[192] A. V. Pinheiro, D. Han, W. M. Shih, and H. Yan, *Challenges and opportunities for structural dna nanotechnology*, Nat Nano **6** (2011), 763–772.

[193] A. Goudarzi, M. R. Lakin, and D. Stefanovic, *DNA reservoir computing: A novel molecular computing approach*, in DNA Computing and Molecular Programming, (D. Soloveichik and B. Yurke, eds.), Vol. 8141 of *Lecture Notes in Computer Science*, Springer International Publishing, 2013, pp. 76–89.

[194] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, *Hierarchical composition of memristive networks for real-time computing*, in Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on, Boston, Ma, July 8-10 2015, pp. 33–38.

[195] M. Hermans, M. Burm, T. Van Vaerenbergh, J. Dambre, and P. Bienstman, *Trainable hardware for dynamical computing using error backpropagation through physical media*, Nat Commun **6**.

[196] C. W. Brown, III, M. R. Lakin, E. K. Horwitz, M. L. Fanning, H. E. West, D. Stefanovic, and S. W. Graves, *Signal propagation in multi-layer DNAzyme cascades using structured chimeric substrates*, Angewandte Chemie International Edition **53** (2014), 7183–7187.

[197] C. W. Brown, III, M. R. Lakin, D. Stefanovic, and S. W. Graves, *Catalytic molecular logic devices by DNAzyme displacement*, ChemBioChem **15** (2014), 950–954.

[198] D. Snyder, A. Goudarzi, and C. Teuscher, *Computational capabilities of random automata networks for reservoir computing*, Physical Review E **87** (2013), 042808.

[199] A. Goudarzi and D. Stefanovic, *Towards a calculus of echo state networks*, Procedia Computer Science **41** (2014), 176–181.

[200] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher, *Computational capacity and energy consumption of complex resistive switch networks*, AIMS Materials Science **2** (2015), 530–545.

[201] A. Goudarzi, A. Shabani, and D. Stefanovic, *Exploring transfer function nonlinearity in echo state networks*, in Computational Intelligence for Security and Defense Applications (CISDA), 2015 IEEE Symposium on, May 2015, pp. 1–8.

[202] A. Goudarzi, A. Shabani, and D. Stefanovic, *Product reservoir computing: Time-series computation with multiplicative neurons*, in 2015 International Joint Conference on Neural Networks (IJCNN), 2015.

[203] A. Goudarzi, D. Arnold, D. Stefanovic, K. B. Ferreira, and G. Feldman, *A principled approach to HPC event monitoring*, in Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale, FTXS '15, ACM, New York, NY, USA, 2015, pp. 3–10.

[204] M. Stella, C. S. Andreazzi, S. Selakovic, A. Goudarzi, and A. Antonioni, *Parasite spreading in spatial ecological multiplex networks*, arXiv:1602.06785 **[q-bio.QM]** (2016), Under review.

[205] A. Goudarzi, S. Marzen, P. Banda, G. Feldman, M. R. Lakin, C. Teuscher, and D. Stefanovic, *Memory and information processing in recurrent neural networks*, arXiv:1604.06929v1 **[cs.NE]** (2016), Under revision.

[206] M. Lukoševičius, H. Jaeger, and B. Schrauwen, *Reservoir computing trends*, KI - Künstliche Intelligenz **26** (2012), 365–371.

[207] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, *Dynamics, computation, and the "edge of chaos": A re-examination*, in Complexity: Metaphors, Models, and Reality. Reading, (G. Cowan, D. Pines, and D. Melzner, eds.), Addison-Wesley, Reading, MA, 1994, pp. 497–513.

[208] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, *Geometry from a time series*, Phys. Rev. Lett. **45** (1980), 712–716.

[209] J. P. Crutchfield and K. Young, *Inferring statistical complexity*, Phys. Rev. Lett. **63** (1989), 105–108.

[210] P. V. Yoshua Bengio, Aaron Courville, *Representation learning: A review and new perspectives*, arXiv:1206.5538 [cs.LG].

[211] Y. Bengio, P. Simard, and P. Frasconi, *Learning long-term dependencies with gradient descent is difficult*, Trans. Neur. Netw. **5** (1994), 157–166.

[212] H. Jaeger *Long short-term memory in echo state networks: Details of a simulation study* Technical Report 27, School of Engineering, Jacobs University, 2012.

[213] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, *Information processing capacity of dynamical systems*, Scientific Reports **2**.

[214] M. Hermans and B. Schrauwen, *Memory in linear recurrent neural networks in continuous time*, Neural Networks **23** (2010), 341 – 355.

[215] M. V. Srinivasan, S. B. Laughlin, and A. Dubs, *Predictive coding: A fresh view of inhibition in the retina*, Proceedings of the Royal Society of London B: Biological Sciences **216** (1982), 427–459.

[216] Q.-Y. Hao, R. Jiang, M.-B. Hu, B. Jia, and W.-X. Wang, *Exponential decay of spatial correlation in driven diffusive system: A universal feature of macroscopic homogeneous state*, Scientific Reports **6** (2016), 19652 EP –.

[217] J. D. Murray, A. Bernacchia, D. J. Freedman, R. Romo, J. D. Wallis, X. Cai, C. Padoa-Schioppa, T. Pasternak, H. Seo, D. Lee, and X.-J. Wang, *A hierarchy of intrinsic timescales across primate cortex*, Nat Neurosci **17** (2014), 1661–1663.

[218] M. C. Mackey and L. Glass, *Oscillation and chaos in physiological control systems*, Science **197** (1977), 287–289.

[219] H. Siegelmann, B. Horne, and C. Giles, *Computational capabilities of recurrent narx neural networks*, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **27** (1997), 208–215.

[220] H. Siegelmann and E. Sontag, *On the computational power of neural nets*, Journal of Computer and System Sciences **50** (1995), 132 – 150.

[221] B. Horne, H. Siegelmann, and C. Giles, *What narx networks can compute*, in SOFSEM '95: Theory and Practice of Informatics, (M. Bartosek, J. Staudek, and J. Wiedermann, eds.), Vol. 1012 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1995, pp. 95–102.

[222] J. Šíma and P. Orponen, *General-purpose computation with neural networks: A survey of complexity theoretic results*, Neural Computation **15** (2003), 2727–2778.

[223] A. Dasgupta, *Probability for Statistics and Machine Learning: Fundamentals and Advanced Topics*, Springer-Verlag, New York, NY, 2011.