Theory and Modeling of Complex Nonlinear Delay Dynamics Applied to Neuromorphic Computing

Bogdan Penkovsky

SPIM

Thèse de Doctorat

école doctorale sciences pour l'ingénieur et microtechniques

UNIVERSITÉ DE FRANCHE-COMTÉ

Theory and Modeling of Complex Nonlinear Delay Dynamics Applied to Neuromorphic Computing

Théorie et Modélisation de la Complexité des Dynamiques Non Linéaires à Retard, Application au Calcul Neuromorphique

Bogdan Penkovsky

THÈSE présentée par

Bogdan Penkovsky

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne Franche-Comté

Spécialité : **Optique et Photonique**

Theory and Modeling of Complex Nonlinear Delay Dynamics Applied to Neuromorphic Computing

Théorie et Modélisation de la Complexité des Dynamiques Non Linéaires à Retard, Application au Calcul Neuromorphique

Soutenue publiquement le 21 juin 2017 devant le Jury composé de :

Maria-Pilar Bernal	Président du Jury	Directeur de recherche CNRS, COMUE Université Bourgogne Franche-Comté
Damien Querlioz	Rapporteur	Chargé de recherche HDR CNRS, Université Paris 11
Guy Van der Sande	Rapporteur	Professeur, Vrije Universiteit Brussel
Daniel Brunner	Examinateur	Chargé de recherche CNRS, COMUE Université Bourgogne Franche-Comté
Christian Gamrat	Examinateur	Chercheur, Université Paris Saclay
Laurent Larger	Examinateur	Professeur des Universités, COMUE Université Bourgogne Franche-Comté
Yuri Maistrenko	Examinateur	Professeur, National Academy of Sciences Kyiv
Miguel Soriano	Examinateur	Associate professor, IFISC Palma de Mallorca

Abstract

The thesis develops a novel approach to design of a *reservoir computer*, one of the challenges of modern Science and Technology. It consists of two parts, both connected by the correspondence between optoelectronic delayed-feedback systems and spatio-temporal nonlinear dynamics. In the first part (Chapters 1 and 2), this correspondence is used in a fundamental perspective, studying self-organized patterns known as *chimera states*, discovered for the first time in purely temporal systems. Study of chimera states may shed light on mechanisms occurring in many structurally similar high-dimensional systems such as neural systems or power grids. In the second part (Chapters 3 and 4), the same spatio-temporal analogy is exploited from an applied perspective, designing and implementing a brain-inspired information processing device: a real-time digital *reservoir computer* is constructed in FPGA hardware. The implementation utilizes delay dynamics and realizes input as well as output layers for an autonomous cognitive computing system.

Keywords: Reservoir computing, Machine learning, Complex systems, Nonlinear delay dynamics, Chimera states, FPGA

Résumé

Cette thèse développe une nouvelle approche pour la conception d'un *reservoir computer*, l'un des défis de la science et de la technologie modernes. La thèse se compose de deux parties, toutes deux s'appuyant sur l'analogie entre les systèmes optoelectroniques à retard et les dynamiques spatio-temporelles non linéaires. Dans la première partie (Chapitres 1 et 2) cette analogie est utilisée dans une perspective fondamentale afin d'étudier les formes auto-organisées connues sous le nom *d'états Chimère*, mis en évidence une première fois comme une conséquence de ces travaux. Dans la deuxième partie (Chapitres 3 et 4) la même analogie est exploitée dans une perspective appliquée afin de concevoir et mettre en oeuvre un concept de traitement de l'information inspiré par le cerveau: un *reservoir computer* fonctionnant en temps réel est construit dans une puce FPGA, grâce à la mise en oeuvre d'une dynamique à retard et de ses couches d'entrée et de sortie, pour obtenir un système traitement d'information autonome intelligent.

Mots-clés: Reservoir computing, Apprentissage automatique, Systèmes complexes, Dynamique non linéaire à retard, États Chimère, FPGA

To Elena...

Contents

Introduction

Let us imagine for a minute the place of technology in the future. Today is a typical day in 2040 and you study a foreign language in a linguistic school. Suppose you study French. As usual, you arrive by a self-driving taxi. Today is your private lesson, so you will talk to a computer. The computer is multilingual and you speak French to enhance your verbal skills. You prefer to combine studying language with learning more about your hobby, photography. That is why you request articles about photography in the middle of the 20th century. You also specify to translate these diverse materials into French, corresponding to your proficiency. The computer replies taking into account your language level and explaining difficult idioms, meanwhile an interactive catalogue of analog photography articles is generated. You select the items of interest, and a book is composed on the fly, tailored to your educational needs. Then, it is sent to your personal cloud — the storage of all your files accessible from the Internet.

The described picture is not an utopia. A self-driving taxi is slower than the one with a human driver because of the safety regulations, and the computer cannot motivate like a real teacher. Therefore, in this future world, real taxi drivers and teachers, who are experts in their domains, still exist. However, a fine balance between the areas of humanity and technology has been established: technology is human-oriented. This is not possible if a self-driving taxi is unsafe or a robotic instructor cannot adapt to the needs of a visitor: accurate request comprehension, excellent suggestion heuristics, accessibility of the vocal and video interfaces intelligently tuned for each user and so on.

All of those requirements are complex technological challenges. To be market-competitive, self-driving cars should not only drive accurately, but also predict if there are vehicles and other objects that potentially may create any safety threats. That is exactly what a responsible human driver does! However, there exists no such machine able to act as a human driver. The apparent reason for this is that our machines are inherently not designed to behave as biological organisms. One of the scenarios to alleviate this shortcoming is by progressive improvement of existing designs, comparing them to the blueprints found in the nature.

The following thesis is meant to contribute towards overcoming the challenges of emerging "smart" technologies via a so-called *neuromorphic* approach. The neuromorphic systems are biological neural networks-inspired technologies that are ultimately called to deal with problems suboptimally solved by the machines invented in the 20th century, namely: the problems of fast and energy-efficient cognition, human interaction, robustness, etc. The journal Frontiers in Neuroscience[1] defines those systems as such that *"carry out robust and efficient neural computation using hardware implementations that operate in physical time. Typically they are event- or data-driven, they employ low-power, massively parallel hybrid analog/digital VLSI circuits, and they operate using the same physics of computation used by the nervous system."*

Built of billions of neurons and possessing trillions of interconnections, the human brain is an impressively complex organ. The brain operates at approximately 10^{18} FLOP, an order of magnitude more than the top supercomputer in 2016 Sunway TaihuLight[2] (10^{17} FLOP), yet consuming less than 0.0002% power of TaihuLight (20W out of 15MW). Not only the currently available technology is energy-inefficient but also, fundamentally limited: it is based on transistors and the minimal transistor size is estimated to be $5\,nm$; today the industry is already close to $11\,nm$. Finally, the present technology is not capable to address modern challenges efficiently because it is limited conceptually: the major part of computing attempts are to some extent restricted to programmed, designed computation, whereas natural phenomena (e.g. biological neural systems) exhibit so-called *intrinsic computation* [1]. The particularity of intrinsic computation is that the substrate and the computing mechanism are often the same thing. The main problem is, however, a lack of knowledge: how is information stored, how much information is encoded, how is it processed and later transmitted [1]. Answering any of those questions would be a great step towards new information processing architectures.

The computational approach we pursue in this work is known as *reservoir computing*: the computing substrate does not need to be specifically designed, but can be preexisting, even randomly generated. The major consequence — when comparing to popular computation methods — is the support of *in-materio* computing, i.e. physical dynamical systems can be involved directly as computation substrates. Moreover, memory and computation algorithm are inseparable, and in that way similar to biological neural systems [2]. Another characteristic feature of reservoir computing that brings it closer to the brain is highlighted in the recent findings showing that the brain might be "prewired" to solve certain tasks [3].

The origins of reservoir computing can be traced back to 1940–1950s when multiple computation paradigms were explored. The pioneering work of McCulloch and Pitts

[1]http://journal.frontiersin.org/journal/neuroscience/section/neuromorphic-engineering#about
[2]https://www.top500.org/system/178764

(1943) [4] laid ground for computations using simple processing units (neurons). A big step to modern neuroscience and also biologically-inspired information processing was the biological neuron's model by Hodgkin and Huxley (1952) [5]. In 1958, Rosenblatt presented the *perceptron* in his seminal paper [6], which in many aspects became the prototype of today's artificial neural networks. The limits of the perceptron were realized after Minsky and Papert have pointed out in 1969 that a single-layer architecture is not enough to build a universal approximating machine [7]. This was a devastating blow to the neural networks community, which led to many discussions. After years of investigations, the universal approximating capabilities of multi-layer neural networks were formally proved by Cybenko (1989) [8]. Another significant milestone was Kohonen's work (1982) [9] describing self-organizing maps, systems learning without a teacher. The next step forward was achieved with invention of *recurrent neural networks* learning by Pineda (1987) [10]. Pineda highlighted three crucial aspects of such artificial networks, making them close to the biological neural systems: a high number of degrees of freedom, nonlinearity, and dissipation nature; all three are the pillars of neuromorphic information processing methods, including the present work. The approach to recurrent neural networks that now is called reservoir computing, was independently developed in 2000-s by Maass *et al.* [11], Jaeger [12], and Steil [13]. Finally, the grounds for our work utilizing a nonlinear time-delay system were laid by Appeltant *et al.* (2011) [14].

In this work, we study the class of nonlinear delayed-feedback systems. The dynamical behavior of those systems evolves in the infinite-dimensional phase space. Such time-delayed systems may exhibit complex behaviors [15, 16], thus becoming an object of interest for our reservoir computing paradigm. It turns out that in many aspects delay dynamical systems are analogous to the ones found in spatially-extended systems [17, 18, 19], strengthening the link between delay dynamics and neural networks. The complexity of nonlinear delay systems grows with the length of the delay [15], increasing system's dimensionality. High dimensionality is also a property of the brain, a gigantic network of neurons, and the easiest way to literally build such a large network would be by finding a clever method to take advantage of the complexity provided by nonlinear delay dynamics. The limits of delay systems for information processing still have to be explored. Therefore, those systems have to be studied fundamentally — both inside and outside of the information processing context.

Structure of the thesis

In Chapter 1, we describe the basic properties of nonlinear delay systems and develop an analogy between time-delay and spatially-extended systems, such as networks of oscillators. We use delay dynamics as an example to introduce driven systems, the ones with external inputs; then we describe how the information is processed in driven systems such as neural networks. An application connecting both delay dynamics and neural networks is reservoir computing, an alternative computation paradigm.

Using the network analogy from Chapter 1, in Chapter 2 we perform numerical and experimental exploration of a recently discovered in nonlinear delayed-feedback systems phenomena, known as *chimera states*. The term originates from symmetrical networks of coupled oscillators, where chimera states are marking spontaneous symmetry breaking resulting in coherent-incoherent motions [20, 21]. Chimera states may arise in such real-world networks as power grids and networks of neurons in the human heart, breaking the synchrony and leading to failure of the system [22].

In Chapter 3, we study reservoir computing applications implemented with nonlinear delay dynamics, where the delay acts as a memory that stores a network of virtual nodes. The goal is to find an optimal configuration that can be readily realized in a form of digital hardware. Our study is applied to two kinds of problems: prediction and classification tasks. The prediction of time-series aims at several steps-ahead estimation of the (future) time-series signal. The task is complicated since any chaotic system is sensitive to the slightest perturbations, making any long-term prediction inaccurate. The speech recognition task is aimed at comprehension of the human speech by a machine. One of the known technological difficulties is speech recognition in noisy environments. We address this challenge with reservoir computing performing isolated spoken digits recognition on signals with artificially added noise.

The paradigm of nonlinear delay dynamics permits different physical realizations: in analog, digital, or hybrid electronics, optics and optoelectronics, etc. In Chapter 4, we build a stand-alone FPGA-based all-digital neuromorphic demonstrator, capable of prediction and classification tasks. The demonstrator leverages the time-multiplexing technique to build networks of 1000 virtual nodes working in real-time. Currently implemented system features the bandwidth up to 12 Mb/s with estimated power consumption of 0.3 W. Since the reservoir computer implementation is completely described with an electronic circuit-specific hardware description language, a high-speed dedicated neuromorphic chip can be ultimately manufactured.

Main contributions provided by this work

Below are highlighted the main outcomes of present work, including both fundamental and applied research. The fundamental research results in the first observation of chimera states in a purely temporal system. We underline the main ingredients required for chimera states in delay dynamical systems: non-local connections provided by the integral term, and an asymmetrically shaped nonlinear function supporting two attractors (fixed-point and chaotic). We demonstrate the multistability in such a delay system where different chimera solutions coexist depending only on the initial conditions. We support the numerical studies with experimental evidence.

The applied research aims at electronic reservoir computing demonstrator in the framework of the BIPhoProc (Brain-Inspired Photonic Processor) project, utilizing the same dynamical system as for chimera states demonstration. We study the possibility to replace continuous-time differential equations with their discrete-time versions, highly suitable for digital implementation. Motivated by hardware efficiency, we perform several steps of optimization, searching the best-performing dynamics model and the simplest nonlinear functions. For that purpose, we employ simultaneous parameter optimization techniques based on *genetic algorithms*. Finally, we implement a digital demonstrator and run a series of benchmark tests to validate our approach. Using the discrete-time delay dynamics approximation and the nonlinearity simplified to piecewise linear functions, we still obtain prediction and classification results comparable to the state of the art. Furthermore, we obtain improved prediction accuracy using the model with the integral term.

This work was supported financially by the region of Franche-Comté, the Labex ACTION program (Contract No. ANR-11-LABX-0001-01), and by the BiPhoProc ANR project (ANR-14-OHRI-0002-02).

Publications

1. *L. Larger, B. Penkovsky, Y. Maistrenko* Laser chimeras as a paradigm for multistable patterns in complex systems. *Nature Communications.* 6:7752 (2015).
2. *Y. Maistrenko, B. Penkovsky, M. Rosenblum* Solitary state at the edge of synchrony in ensembles with attractive and repulsive interactions. *Phys. Rev. E* 89:060901 (2014).
3. *L. Larger, B. Penkovsky, Y. Maistrenko* Virtual chimera states for delayed-feedback systems. *Phys. Rev. Lett.* 111:054103 (2013).

Conference presentations and work dissemination

1. Oral presentation "Chimera States in Nonlinear Systems with Delayed Feedback" at "Complex patterns on networks" mini-symposium, Dynamics Days Europe, Szeged, Hungary, June 6, 2017.
2. Oral presentation "Ma thèse en 180 secondes" ("Three minute thesis") at the regional final of the University of Bourgogne Franche-Comté, Dijon, April 4, 2017.
3. Oral presentation "Towards a Brain-Inspired Computer With a Delay Dynamics" at GDR BioComp, INSA Lyon, October 11, 2016.
4. Invited seminar "Laser Delay Dynamics For Information Processing" at TU Berlin, Germany, July 13, 2016.
5. Oral presentation "FPGA-Based Reservoir Computing" at FEMTO-ST, Besançon, July 1, 2016.
6. Oral presentation "Toward New General-Purpose Processor With Nonlinear Transient Computing" at Dynamics Days Europe, Corfu, Greece, June 9, 2016.
7. Poster "A New Architecture For a General-Purpose Microprocessor" at RNL, Paris, March 15, 2016.
8. Poster "A New Architecture For a General-Purpose Microprocessor" at Atelier SMYLE, Arc-et-Senans, September 17–18, 2015.
9. Poster "Chimera States In Laser Delay Dynamics" at NonLinear Summer School, Peyresq, France, August 21–28, 2015.
10. Poster "Chimera States In Laser Delay Dynamics" at "40 years of Kuramoto model", Dresden, Germany, July 27–28, 2015.
11. Poster "Chimera States In Laser Delay Dynamics" at Rencontre Nonlinéaire (RNL) 2015, Paris, March 15–16, 2015.
12. Oral presentation "Chimera States In Laser Delay Dynamics" at WIAS, Berlin, Germany, November 25, 2014.
13. Oral presentation "Chimera States In Laser Delay Dynamics" at NonLinear Summer School, Peyresq, August 23, 2014.
14. Poster "Nonlinear Delay Optoelectronic Oscillator Exhibiting Laminar-Turbulent Transition" at the colloquium on Subcritical Transition to Turbulence, Cargese, Corsica, May 7, 2014.

Chapter 1

Time-delay systems and networks

1.1 Nonlinear delayed-feedback systems

Qu'est-ce que le passé, sinon du présent qui est en retard?
– Pierre Dac, French humorist

"Oh dear! I shall be too late!"

Time delays are ubiquitous, they are found in numerous real phenomena that develop on a finite speed. For instance, due to a finite propagation speed, it takes us around 48 minutes to send or receive a light signal from a Jupiter's satellite. Subsequently, delays play a crucial role in a satellite's remote control. Understanding the human brain, where the information is processed with respect to delayed propagation of chemical signals, is another grand challenge. In daily life, a traffic jam moves slowly as everyone drives in a chain of delays. Those delays appear because of a finite time that takes a driver to see and to respond to the movement ahead. Sometimes traffic jams become so slow that the victim drivers remind the White Rabbit[1] who was always late. Finally, everyone in a shower experienced a delayed feedback right on themselves. The water first being too cold, then becomes too hot. That is the delayed response to turning the tap in

[1]A personage from "Alice's Adventures in Wonderland" by Lewis Carroll. Illustration by John Tenniel.

the past, when the water was still perceived as cold. Hurry to switch back the tap, and the water becomes too cold again.

In this work, we consider a class of dynamical systems with nonlinear delayed feedback that can be mathematically represented by a delay differential equation (DDE) of the following type:

$$\tau\dot{x}(t) + x(t) = f\left(\beta, x(t-\tau_D)\right), \tag{1.1.1}$$

where $x(t) \in \mathbb{R}^1$ is a dynamical variable, $\dot{x}(t) \equiv dx(t)/dt$ is its derivative on time, τ is a linear decay time constant, τ_D is a time delay, $f : \mathbb{R}^1 \to \mathbb{R}^1$ is a nonlinear function, and β is a control parameter[2]. To describe the initial state of a system given by Eq. (1.1.1), a function defining the initial conditions between $t = -\tau_D$ and $t = 0$ is required. The number of oscillations with a time scale τ one can fit into the delay τ_D, is related to the number of degrees of freedom (or the complexity) for such delay dynamics. An illustration can be drawn from the example with a traffic jam. Here, the state of the system are individual velocities of the cars on the road (delay line), where each car adds to the overall complexity of the dynamics. A *phase space* is a multidimensional space in which all the states of a system are represented; every degree of freedom of a dynamical system is an axis in this space.

Due to the complexity caused by the high dimensionality of the phase space, nonlinear systems with time delay exhibit a rich variety of behaviors [15, 16]. Such systems are common in many research fields, finding applications in photonics [23, 24, 25, 26], biology [27, 28, 29], chaos theory [15, 24, 30], chaos cryptography and communications [23], and novel computational concepts, including processing and memory units [14, 31, 32, 33, 34, 35, 36, 37, 38].

Prominent example of Eq. (1.1.1) is the Ikeda DDE, which describes the behavior of a delay oscillator, illuminating transition to the chaos in optics [39]:

$$\tau\dot{x}(t) + x(t) = \beta \sin^2\left(x(t-\tau_D) + \Phi_0\right), \tag{1.1.2}$$

where β is a delayed feedback gain, Φ_0 is a constant phase offset, and the linear decay time τ is assumed to be small with respect to the delay time τ_D, i.e. $\tau \ll \tau_D$. In the literature, such DDEs are referred to as *long delay* systems.

Another famous example of systems governed by Eq. (1.1.1) is the Mackey-Glass DDE, with a nonlinear function $f(x) = \beta x/\left(1 + x^{10}\right)$ [27]. Originally describing blood cell concentration dynamics, Mackey-Glass equation has been extensively studied for effects of the nonlinear delayed feedback.

[2]For the sake of simplicity, sometimes we will omit the parameter β in $f(x) \equiv f(\beta, x)$.

1.1.1 Route to chaos in DDE

To understand very qualitatively the way complex behavior can emerge from DDE described by Eq. (1.1.1), a so-called adiabatic approximation can be introduced. When the parameter τ is small, i.e. let $\tau \to 0$, then the dynamics of Eq. (1.1.1) reduces to a difference equation, i.e. without derivative:

$$x(t) = f\left(x(t - \tau_D)\right). \tag{1.1.3}$$

Now, the behavior is fully determined by the nonlinear map f. Without loss of generality, we can rewrite Eq. (1.1.3) as a recurrence relation:

$$x_n = f(x_{n-1}), \tag{1.1.4}$$

with discrete time $n = 0, 1, \ldots$. The nonlinear function f is usually assumed to be unimodal, i.e. it has one maximum. The simplest example that illustrates the route to chaos in the difference equation Eq. (1.1.4) is the logistic map $f(\lambda,x)$ [40]:

$$f(\lambda, x) = \lambda x(1 - x), \quad x \in [0, 1] \tag{1.1.5}$$

where $\lambda > 0$ is a control parameter. Depending on λ, the discrete dynamical system governed by Eq. (1.1.5) either stays in a regime of equilibrium, exhibits periodic oscillations, or demonstrates chaotic, aperiodic behavior. When $1 < \lambda < 3$, the dynamics is trivial: the system rests in a fixed point state $x^* = 1 - 1/\lambda$, $x^* \equiv x_n = x_{n-1}$ (Fig. 1.1.1, top). However, as soon as $\lambda > 3$, the fixed point solution becomes unstable, and instead, one can observe period-2 oscillation cycle. Crossing another bifurcation threshold $\lambda \simeq 3.449$, the next period doubling bifurcation occurs, and period-4 cycle arises in the dynamical system Eq. (1.1.4). At further increase of λ, period doubling bifurcations continue to occur, producing cycles of period 2^m, until at $\lambda_\infty \simeq 3.570$ the dynamics becomes chaotic. The rate of the period doubling bifurcations is known as the universal *Feigenbaum's constant* δ_F (1978) [41]:

$$\delta_F = \lim_{m\to\infty} \frac{\lambda_m - \lambda_{m-1}}{\lambda_{m+1} - \lambda_m} \simeq 4.669. \tag{1.1.6}$$

Since δ_F holds for smooth unimodal f with a non-degenerated maximum, i.e. $f'' \neq 0$, we can take another, sinusoidal function which fulfills this condition $f'' = 2\lambda \cos(2x)|_{x=\pi/2} = -2\lambda \neq 0$:

$$x_n = \lambda \sin^2(x_{n-1}). \tag{1.1.7}$$

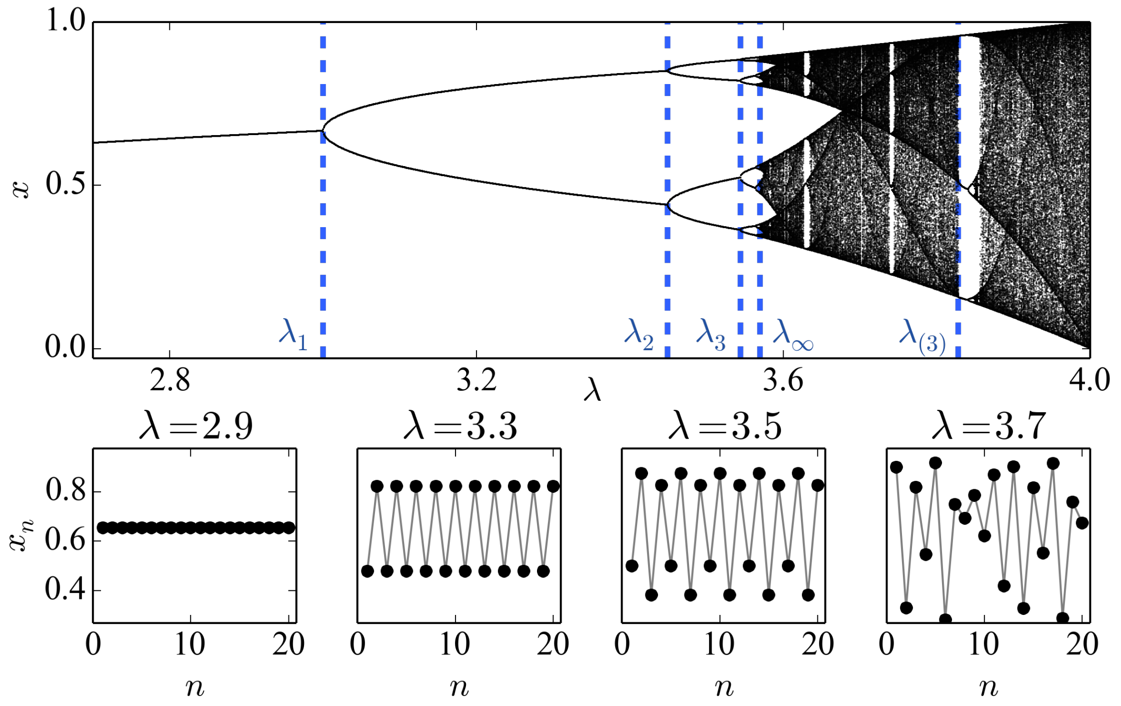

Figure 1.1.1: Upper: Bifurcation diagram of logistic map $x_n = \lambda x_{n-1}(1 - x_{n-1})$ with the control parameter λ. Successive iterates of the map are plotted after transients have died out. Vertical dashed lines show bifurcation values of $\lambda_1 = 3, \lambda_2 \simeq 3.449, \lambda_3 \simeq 3.544, \lambda_\infty \simeq 3.57, \lambda_{(3)} \simeq 3.828$ that precede windows of period-2, period-4, period-8, chaotic motion, and period-3 window, respectively. **Lower**: Examples of fixed point, period-2, period-4, and chaotic regimes.

The bifurcation diagram of this difference equation is depicted in Fig. 1.1.2 (a), where one can observe a period doubling cascade which eventually leads to chaotic behavior. The order of bifurcations is exactly the same as for the logistic map described by Eq. (1.1.5). It appears that for any continuous nonlinear function f periodic orbits in Eq. (1.1.4) occur in accordance with the *Sharkovsky ordering* (1964) [42] of all the periods:

$$2^0 \prec 2^1 \prec 2^2 \ldots 2^m \cdot 5 \prec 2^m \cdot 3 \ldots 2^2 \cdot 5 \prec 2^2 \cdot 3 \ldots 2^1 \cdot 5 \prec 2^1 \cdot 3 \ldots 7 \prec 5 \prec 3. \quad (1.1.8)$$

Theorem. *(Sharkovsky) If a continuous one-dimensional map f has an orbit of period p and $q \prec p$ in ordering given by Eq. (1.1.8), it also has an orbit of period q.*

For instance, if one can observe period $p = 3$, then period $q = 5$ can be also observed since $5 \prec 3$. The same holds true for $q = 7$ since $7 \prec 3$, and so on. As a

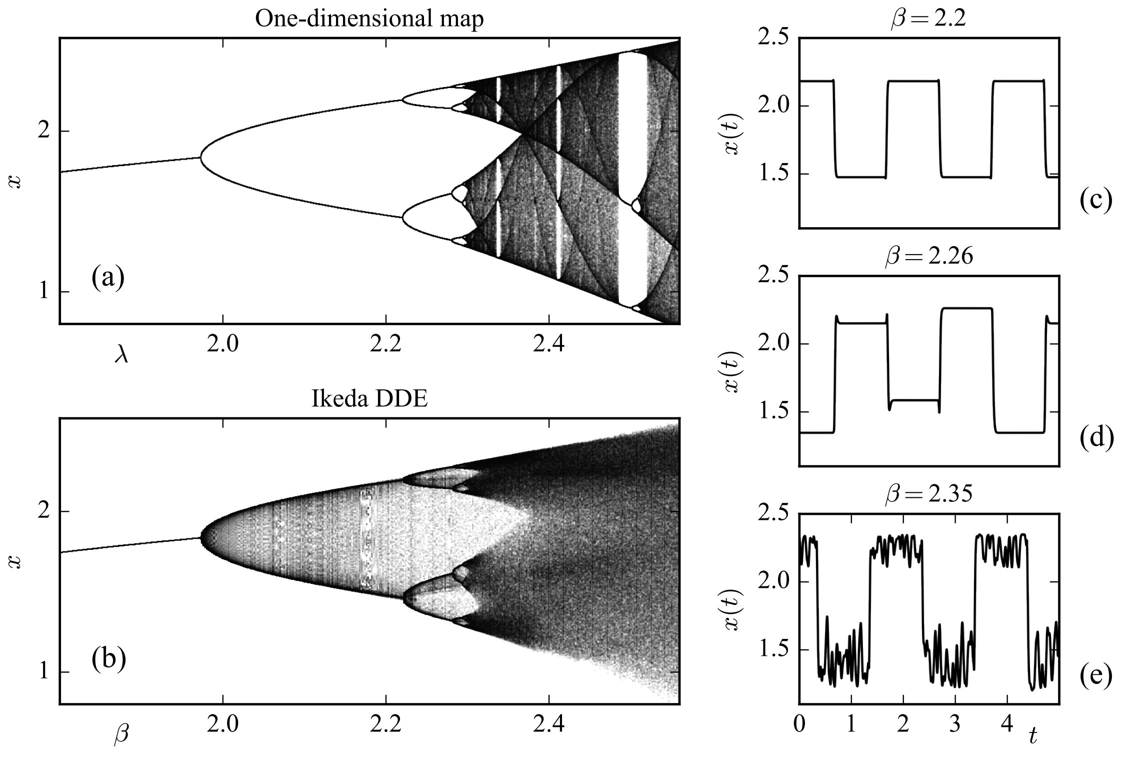

Figure 1.1.2: Left: (a) Bifurcation cascade of difference equation $x_n = \lambda \sin^2(x_{n-1})$ depending on the control parameter λ; (b) Bifurcations in a continuous-time Ikeda DDE dynamics $0.01\dot{x}(t) + x(t) = \beta \sin^2(x(t-1))$ depending on the parameter β. Semi-transparent dots denote the asymptotic trajectories x(t) over a single time delay interval after a transient of 3000 delays. **Right**: Typical regimes in the Ikeda DDE: (c) period-2 limit cycle, (d) period-4 limit cycle, (e) chaotic motion.

consequence, observing a period-3 cycle leads to all other integer periods. Therefore, observing period-3 window of periodicity indicates that the discrete-time system can exhibit chaotic motion, i.e. "Period Three Implies Chaos" Li, Yorke (1975) [43]. Thus, sinusoidal map given by Eq. (1.1.7) follows the same bifurcation scenario as the logistic map Eq. (1.1.5).

Now, going back from the iterated map Eq. (1.1.7) to the original Ikeda DDE, the derivative term $\tau\dot{x}(t)$ with nonzero τ has to be introduced:

$$\tau\dot{x}(t) + x(t) = \beta \sin^2 (x(t - \tau_D)), \tag{1.1.9}$$

where τ_D is a delay, $0 < \tau \ll \tau_D$ is a small parameter, and β is an amplification coefficient of f. Therefore, Ikeda DDE Eq. (1.1.9) may be considered as a singular perturbation of the difference equation Eq. (1.1.7).

The bifurcation diagram of the class of systems described by Eq. (1.1.9) is depicted in Fig. 1.1.2 (b). There, the bifurcations are computed with respect to the control parameter β. Other parameters $\tau = 0.01$ and $\tau_D = 1$ are set to be constant. As it can be noticed, bifurcation values of β in the Ikeda DDE Eq. (1.1.9) does agree with the corresponding one-dimensional map Eq. (1.1.7). One can observe a cascade of period doubling bifurcations of limit cycles where the new cycles occur with increase of the control amplification parameter β. As in the case of a one-dimensional map, the bifurcations lead to chaos. Examples of period-2 and period-4 limit cycles and chaotic motion can be seen on the right side of Fig. 1.1.2. However, one observes alternating plateaus of the delay length, instead of the alternating values of x_n in the one-dimensional iterated map Eq. (1.1.7). An essential peculiarity of the time delay model, which differs DDE Eq. (1.1.9) from difference equation Eq. (1.1.7), is the absence of the windows of periodicity in the chaotic parameter range $\beta > \beta_\infty \simeq 2.29$. This is the consequence of the fact that Eq. (1.1.9) is one-dimensional, whereas DDE Eq. (1.1.7) is infinite dimensional. The two models do not exhibit the same number of degrees of freedom, or equivalently the same dimensionality. Therefore, Eq. (1.1.9) cannot be accurately approximated by setting $\tau = 0$.

1.1.2 Interpretation of Ikeda DDEs

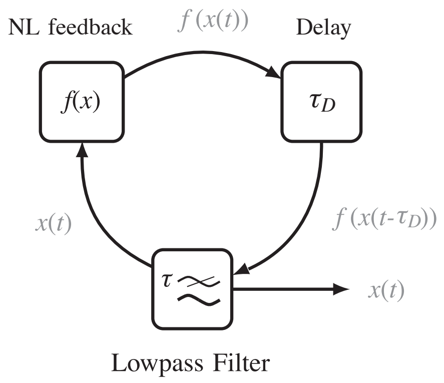

Figure 1.1.3: Oscillator with nonlinear delayed feedback realizing the Ikeda DDE.

In physical systems, a lowpass filter is responsible for the differential term: the dynamical variable x can never change infinitely fast as it is the case in the iterated map approach. Therefore, the Ikeda DDE is implemented in an architecture comprising three principal elements: a delay, a nonlinear function f, and a lowpass filter (Fig. 1.1.3). The signal $x(t)$ is nonlinearly transformed by a nonlinear function $f(x) = \beta \sin^2(x + \Phi_0)$ and is delayed by time τ_D, thus, producing a delayed signal $\beta \sin^2(x(t - \tau_D) + \Phi_0)$. This signal is consecutively filtered, limiting the system's fastest response to a dirac-perturbation to the characteristic response time τ. Thus, nonlinear delay dynamics is realized as a feedback loop, in which a linear filter provides the argument for the nonlinear function f, which in turn then serves as the filter's input. This iterative relationship is also schematically illustrated in Fig. 1.1.3. Such architecture allows for observing the route to chaos in the Ikeda DDE experimentally.

The system in Fig. 1.1.3 includes a lowpass filter, which is described in the Fourier domain as

$$H(\omega) = \frac{G_0}{(1 + i\omega\tau)} = \frac{I_F(\omega)}{I_D(\omega)}, \tag{1.1.10}$$

where G_0 is the filter's gain, $I_F(\omega) = \mathrm{FT}\{i_F(t)\}$ and $I_D(\omega) = \mathrm{FT}\{f[i_D(t - \tau_D)]\} = \mathrm{FT}\left\{\sin^2[i_D(t - \tau_D) + \Phi_0]\right\}$ are the response and the input to the filter, respectively. By FT we denote the Fourier transform, which decomposes a function of time into the frequencies it consists of.

Converting Eq. (1.1.10) from the Fourier into the time domain, first we rewrite the equation as:

$$(1 + i\omega\tau)I_F(\omega) = G_0 I_D(\omega). \tag{1.1.11}$$

By expanding the brackets on the left side and making use of the identity $\mathrm{FT}^{-1}(i\omega X(\omega)) = \frac{d}{dt}\mathrm{FT}^{-1}(x(t))$, i.e. multiplication by $i\omega$ in the Fourier domain is equivalent to differentiation in the time domain, the Ikeda differential equation becomes:

$$i_F(t) + \tau \frac{di_F}{dt}(t) = G_0 \sin^2[i_D(t - \tau_D) + \Phi_0], \tag{1.1.12}$$

where $i_F(\omega) = \mathrm{FT}^{-1}\{I_F(\omega)\}$ and $\sin^2[i_D(t - \tau_D) + \Phi_0] = \mathrm{FT}^{-1}\{I_D(\omega)\}$; FT^{-1} is the inverse Fourier transform. Then, Eq. (1.1.12) in a normalized form, often used for numerical analysis, reads as:

$$\varepsilon \dot{x}(s) + x(s) = \beta \sin^2[x(s - 1) + \Phi_0] \tag{1.1.13}$$

where $s := t/\tau_D$ is the normalized time and $\varepsilon := \tau/\tau_D$ is a normalized parameter representing differential time characteristics of the dynamics. Typically, $\varepsilon \ll 1$ and the value $1/\varepsilon$ characterizes the linear "memory" of the differential dynamics. On the other hand, the nonlinear function f is responsible for the difference dynamics (Eq. (1.1.4)) and its nonlinear contribution. Thus, the system's complexity is determined by the interaction of the differential and difference parts of the DDE Eq. (1.1.13). Fixing ε, control parameters are the feedback gain β and the feedback offset phase Φ_0. Control parameter β is responsible for the nonlinear function's f stretching and Φ_0 determines the horizontal shift of f.

The global impact of events described by DDEs locally in time, might be difficult to understand. To gain better comprehension of the DDE behavior, a signal processing approach can be employed. Equation (1.1.13), which provides only local differential dynamics of the Ikeda model, can be alternatively rewritten as a global convolution product

$$x(t) = \int_{-\infty}^{t} h(t-\xi) \cdot f\left[x(\xi-1)\right] d\xi, \tag{1.1.14}$$

where $f(x)$ is a nonlinear function and $h(t)$ is the impulse response function (Fig. 1.1.4) related to the filtering of the process, i.e. the linear, differential part of Eq. (1.1.13). Function $h(t)$ is an inverse Fourier transform of the filter's transfer function $H(\omega)$. This impulse response $h(t)$ is responsible for local interactions within the DDE. Equation (1.1.14) has no differential term, instead it explicitly provides information how both the delayed events $x(\xi-1)$ and "local" events $h(t-\xi)$ contribute to the solution $x(t)$.

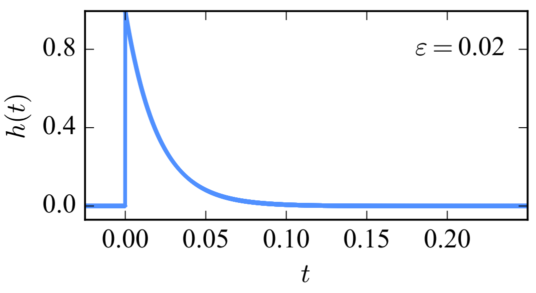

Figure 1.1.4: Impulse response function $h(t)$ of DDE Eq. (1.1.13), $\varepsilon = 0.02$.

1.1.3 Space-time representation of DDEs

Reminding the example with a traffic jam, one can imagine a delay interval as a section of a road with many cars. Taking another example, with a delayed hot water in a shower, one realizes that the delay line is a tube full of water. The common aspect in both cases is that the delay interval has a *spatial* dimension. This gives us an intuitive motivation for considering any delay interval as spatial. Indeed, if one tries to study the dynamics of Eq. (1.1.13), one needs knowledge of the state within the initial interval of $[-1; 0]$. To reveal patterns in the DDE's global behavior, it is useful to represent the information given by variable x as a (discrete time) iteration of a nonlinear operator defined on the delay interval. The delay interval is then referred to as *virtual space* and the dynamics in Eq. (1.1.13) can be visualized using a two-dimensional space-time representation, originally proposed by Arecchi *et al.* (1992) [17, 18, 19] (Fig. 1.1.5).

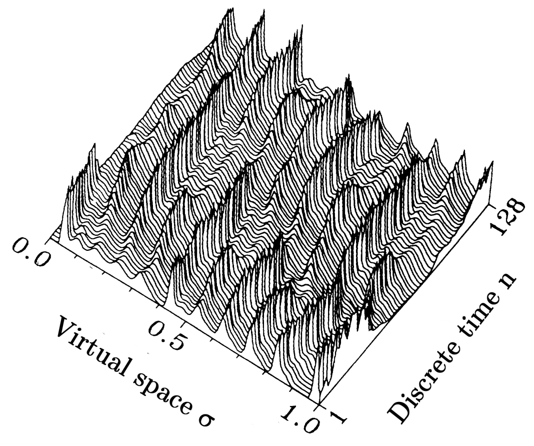

Figure 1.1.5: Space-time plot: a two-dimensional DDE representation, image adapted from [17]. Here, σ denotes virtual space within a delay interval and n is discrete time.

The virtual space can be understood as a state of a virtual network. Hence, the dynamics related to the interactions within the network lies on a significantly longer timescale than τ_D. The space-time visualization scheme of otherwise one-dimensional time traces allows observation of the pattern formation within the virtual space of the delay interval. Therefore, a convenient mean for dynamics monitoring on a longer scale is provided.

The virtual space of DDEs is often materialized in experiments. Using an optical delay line (e.g. a spool of optical fiber), the state of delay line is a "physical" space. An example with a discrete virtual space $[0, N-1]$ is an electronic delay line of depth N. Such a delay line can be realized by a first in-first out (FIFO) memory buffer oftentimes employed to emulate a delay in electronic/optoelectronic systems. Then, the state of the FIFO delay line completely changes in a discrete time $n := N/f_{CLK}$ where f_{CLK} is the FIFO's clock frequency.

1.1.4 DDE as a time-multiplexed circular network

To reveal patterns in the DDE's global behavior we used the space-time representation. Each delay interval τ_D was described as containing a "frozen" state, a *snapshot* of a virtual spatial dimension. This description can be formalized with the help of impulse response h. For that, we interpret h as a local coupling between spatially distributed nonlinear oscillators. Motivated by the spatio-temporal representation, such a network would be of circular geometry.

To interpret the Ikeda DDE as a discrete time evolution of a functional trajectory defined over the normalized time interval $[0, 1+\gamma]$, $\gamma = O(\varepsilon\beta)$, the relation between DDE solutions and the virtual space-time representation is defined as:

$$x_\sigma(n) = x(s),\ \sigma \in [0, 1+\gamma],\ (1+\gamma)\,n + \sigma = s,\ n = 0, 1, 2, \ldots \tag{1.1.15}$$

where each element of $x_\sigma(n)$ is the state of a virtual network of oscillators at discrete time n and $\sigma \in [0, 1+\gamma]$, $\gamma \ll 1$ is a new spatial coordinate defined through $x(s)$ in the formula (1.1.15).

With the identity $(1+\gamma)\,n + \sigma = s$, Eq. (1.1.14) can be written as iterated functional sequences

$$x_\sigma(n) = \int_{-\infty}^{n(1+\gamma)+\sigma} h\left(n(1+\gamma) + \sigma - \xi\right) \cdot f\left(x(\xi - 1)\right) d\xi,\ \sigma \in [0, 1+\gamma],\ n = 0, 1, \ldots \tag{1.1.16}$$

Changing discrete variable $n' = n/(1+\gamma)$, Eq. (1.1.16) is rewritten as:

$$x_\sigma(n') = \int_{-\infty}^{n'+\sigma} h\left(n' + \sigma - \xi\right) \cdot f\left[x(\xi - 1)\right] d\xi. \tag{1.1.17}$$

Therefore, the temporal dynamics of DDE Eq. (1.1.13) can be interpreted as a functional sequence $G_{n'}$, $n' = 1, 2, \ldots$, where $G_{n'} : g_{n'-1} \to g_{n'}$ is a nonlinear integral operator mapping a function given at $[0, 1]$ onto itself, i.e.:

$$x_\sigma(n') = x_\sigma(n' - 1) + I_\sigma(n'), \sigma \in [0,1] \tag{1.1.18}$$

where $x_\sigma(n' - 1) = \int_{-\infty}^{(n'-1)+\sigma} h\left(t - \xi\right) \cdot f\left[x(\xi - 1)\right] d\xi$ and

$I_\sigma(n') = \int_{(n'-1)+\sigma}^{n'+\sigma} h\left(t - \xi\right) \cdot f\left[x(\xi - 1)\right] d\xi$, $t = n' + \sigma$.

Substituting variable $\xi' = \xi - n'$, the integral $I_\sigma(n')$ is rewritten in a simpler form:

$$I_\sigma(n') = \int_{\sigma-1}^{\sigma} h\left(\sigma - \xi'\right) \cdot f\left[x_{\xi'}(n'-1)\right] d\xi'. \tag{1.1.19}$$

Equation (1.1.18) describes a spatially-extended system which can be solved, e.g. by the Euler integration scheme, see [17]. Consequently, I_σ corresponds to a nonlinear "spatial" coupling in the network of oscillators x_σ, continuously distributed within the interval $\sigma \in [0,1]$ and evolving in discrete time n'. From equations (1.1.18) and (1.1.19) it can be seen, therefore, that each "node" $x_\sigma(n')$ is dependent (a) on its own state one iteration before $x_\sigma(n'-1)$ and (b) on the state of the neighboring nodes through the impulse response function h.

For instance, in the Ikeda DDE with

$$f\left[x_{\xi'}(n'-1)\right] = f\left[x(\xi', n'-1)\right] = \sin\left[x(\xi', n'-1) + \Phi_0\right],$$

the discrete-time evolution of Eq. (1.1.18) obeys the form of:

$$x_\sigma(n') = x_\sigma(n'-1) + \int_{\sigma-1}^{\sigma} h(\sigma - \xi') \cdot \sin\left[x_{\xi'}(n'-1) + \Phi_0\right] d\xi'. \tag{1.1.20}$$

The system described by Eq. (1.1.20) can already be considered as a circular network. This structure is revealed with a comparison to a ring of phase oscillators [20, 21]:

$$\frac{\partial}{\partial t}\varphi(x,t) = \omega - \int_{-\pi}^{\pi} G(x-x') \sin\left(\varphi(x,t) - \varphi(x',t) + \alpha\right) dx', \tag{1.1.21}$$

where x is the position in space of an oscillator, whose phase is φ, ω is the natural frequency constant, and α is the phase constant. The term $\varphi(x,t) - \varphi(x',t)$ describes the oscillators' phase difference, while $G(y)$ is the distance-dependent coupling kernel. Derived from the complex Ginzburg-Landau equation and known under the name of Kuramoto-Sakaguchi model, Eq. (1.1.21) is a phase model commonly utilized to describe collective synchronization in large ensembles of coupled oscillators. Equation (1.1.21) is able to capture such distinctive solutions as synchrony and chaos.

Choosing an appropriate co-rotating frame, the natural frequency ω can be set to zero. Applying $\alpha' := \alpha - \pi$, Eq. (1.1.21) becomes:

$$\frac{\partial}{\partial t}\varphi(x,t) = \int_{-\pi}^{\pi} G(x-x') \sin\left(\varphi(x,t) - \varphi(x',t) + \alpha'\right) dx', \tag{1.1.22}$$

and when making use of Euler's discretization with time step $\Delta t = 1$:

$$\frac{\partial}{\partial t}\varphi(x,t) = \frac{\varphi(x,t+1) - \varphi(x,t)}{(t+1) - t} = \varphi(x,t+1) - \varphi(x,t). \tag{1.1.23}$$

Therefore, Eq. (1.1.22) can be read as:

$$\varphi(x,t+1) = \varphi(x,t) + \int_{-\pi}^{\pi} G(x-x') \sin\left(\varphi(x,t) - \varphi(x',t) + \alpha'\right) dx', \tag{1.1.24}$$

or equivalently

$$\varphi(x,t) = \varphi(x,t-1) + \int_{-\pi}^{\pi} G(x-x') \sin\left(\varphi(x,t-1) - \varphi(x',t-1) + \alpha'\right) dx'. \tag{1.1.25}$$

Coming back to Eq. (1.1.20), the variable $x(\sigma, n')$ depends on spatial position σ and discrete time n', as does the phase $\varphi(x,t)$ in Eq. (1.1.25); the term $h(\sigma - \xi')$ (Fig. 1.1.4) plays the role comparable to the coupling kernel $G(x-x')$; finally the constant phase shift Φ_0 is equivalent to α. However, instead of the phase difference $\varphi(x,t-1) - \varphi(x',t-1)$ coupling in Eq. (1.1.25), there is an amplitude self-coupling $x(\xi', n'-1)$ in Eq. (1.1.20).

The state of the network x_σ is stored in the time delay interval $[0;1]$. This interval corresponds to a linear, circular memory, which for example can readily be implemented using an optical fiber. In that way, dynamical variable $x(t)$ can be regarded as *time-multiplexing* of a network with circular connectivity structure (Fig. 1.1.6).

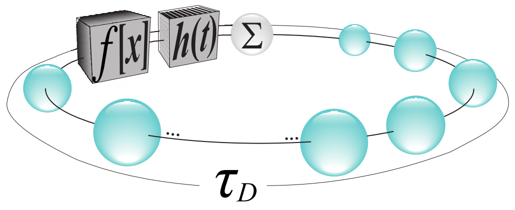

Figure 1.1.6: The Ikeda DDE as a convolution product between an impulse response function $h(t)$ and a nonlinear function $f(x)$ having the delayed variable $x(t-1)$ as its argument.

When a network of oscillators contains *discrete* nodes as elementary units (e.g. when dealing with a digital delay line), then space is discrete and Eq. (1.1.18) can be rewritten as:

$$x[dn+s] = x_s[n] = x_s[n-1] + \sum_{k=s-d}^{s-1} h[s-k] \cdot f(x[k-d]), s = 1, 2 \ldots d, \qquad (1.1.26)$$

where square brackets $[\cdot]$ denote discretization in time, d is the number of nodes in the virtual network, i.e. in the discrete delay line.

1.1.5 Driven systems

Until now, only so-called *autonomous* DDE systems were considered, i.e. without external input. However, in the real world there exist numerous non-isolated systems. Systems with external input are called *driven* systems. Modifying Eq. (1.1.1) by including external input signal $u(t)$, we obtain a driven DDE (in a normalized form):

$$\varepsilon\dot{x}(t) + x(t) = f\left(x(t-1) + u(t)\right), \qquad (1.1.27)$$

where $f(x)$ is a nonlinear transformation. Equation (1.1.27) is schematically represented in Fig. 1.1.7.

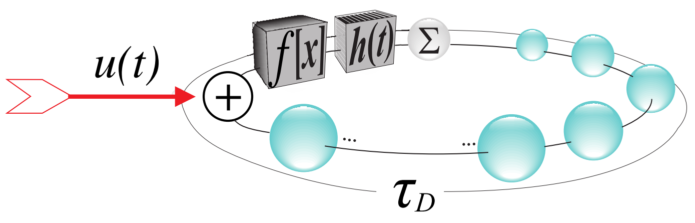

Figure 1.1.7: A driven DDE, with an external information input $u(t)$.

The drive signal $u(t)$ can be interpreted as a way to inject in the initial condition, i.e. a waveform defined on a time interval of a duration corresponding to the delay. If $u(t)$ is such a waveform on $[-\tau_D; 0]$, and zero everywhere else, we have a conceptual way on how to inject a specific initial condition. After such an initial condition, one then typically observes the dynamics in an autonomous way. After some transient, the autonomous dynamics converge toward an asymptotic motion, which is typical one corresponding to the many different possible solutions along the bifurcation diagram (a steady state, a periodic motion, or a chaotic motion, depending on the parameter values set for the system).

In the case of a non-zero $u(t)$ over the full time axis, there is continuous injection of a signal, and consequently the system can not reach an asymptotic state. It is forced to a continuously transient motion, because of the presence of the signal $u(t)$, which is then said to trigger a transient trajectory in the phase space of the delay dynamics. Such a transient trajectory will be later used for information processing according to a brain-inspired technique known as *reservoir computing.*

Driven systems are common for other applications. For example, in the case of a *system identification* [44], the system under analysis is driven by a sinusoidal signal $u(t)$. Another example is the optical information processing demonstrated in [45] for chaos-based encryption application. Information processing systems are driven, in fact, by an external information input signal $u(t)$. For instance, voice recognition systems are driven by a speech waveform $u(t)$, self-driving cars are driven by environment signals $\mathbf{u}(t)$ and so on. In the most general context, the utilization of such driven networks of nonlinear nodes for information processing defines *artificial neural networks*, a prominent subject in machine learning field.

1.2 Artificial neural networks

What I cannot create, I do not understand.
– Richard Feynman, Nobel Prize winner in Physics (1965)

Despite the fact that computers are ubiquitous today, their working principle is far from biological. The human brain, which consumes as little as 20 W of power [46], is able to recognize familiar faces in presence of noise (e.g. poor lighting conditions) in fractions of seconds. However, this task is tough even for the fastest modern computers. This capability poses one of the central problems of biologically inspired hardware: how can information be processed more efficiently.

Another significant problem is to understand the brain from biological view point. It is known that the human brain consists of hundred billions of neurons, which are estimated to have thousand trillions of interconnections. However, little is known about how does that biological network operate. Understanding of this intricate organization might be crucial for medicine and related fields. So-called constructive approach has a goal to approximate the brain, or at least a network of neurons. Building a system analogous to the brain could help to shed some light onto the organ, which is still treated mostly as a blackbox.

1.2.1 Approximation of neural behavior

Though the anatomy of neural cells is known (Fig. 1.2.1, upper), modeling collective cell dynamics is an open question. A biologically justified model of a neuron was introduced by Hodgkin and Huxley in 1952 [5] (Nobel Prize 1963). The model is a four-dimensional nonlinear system of ordinary differential equations (ODEs). However, computationally simulating a network of such neurons is extremely hard. Should one try to compute, for instance, the dynamics of 10^9 neurons (only 1% of estimated total neurons in the human brain), the model of Hodgkin-Huxley becomes impractical because of the amount required resources, such as processing power, memory, and storage. It is worth mentioning this computational problem represents one of the global challenges of natural sciences and is addressed by the Human Brain Project[3] (1 billion €).

A much simpler neuronal model was proposed by Izhikevich [47]. The model is a two-dimensional system of ODEs:

$$\begin{aligned} \dot{v} &= 0.04v^2 + 5v + 140 - u + I, \\ \dot{u} &= a(bv - u), \end{aligned} \tag{1.2.1}$$

where v and u are dimensionless variables, characterizing the membrane potential of the neuron and a membrane recovery variable, respectively. a and b are dimensionless parameters describing the timescale of the recovery variable u and the sensitivity of the variable u to the fluctuations of membrane potential v, respectively. External forcing I represents synaptic currents, making Eq. (1.2.1) a driven system (Section 1.1.5). Being significantly simpler compared to the original Hodgkin-Huxley model, system of equations (1.2.1) is still able to capture such typical neural regimes as *spiking*, *chattering*, and *bursting*. Thus, it can be used to model biologically-plausible neural behavior. However, even computation with this simplified model is still a complicated engineering task when dealing with billions of neurons and, therefore, is not yet feasible.

To achieve computational efficiency for biologically-inspired information processing, one may need to make a further step towards system simplification. Instead of individual neurons, modeling a collective behavior of neuronal network can be attempted. Here *artificial neural networks*, referred further in the text as neural networks (NNs), are coming into the play.

Figure 1.2.1 illustrates the idea behind the transition from a biological to a very simplified mathematical, formal model of the neuron. The biological neuron (Fig. 1.2.1, upper) has a cell body (soma) that receives electrical impulses from other neurons

[3]http://www.humanbrainproject.eu

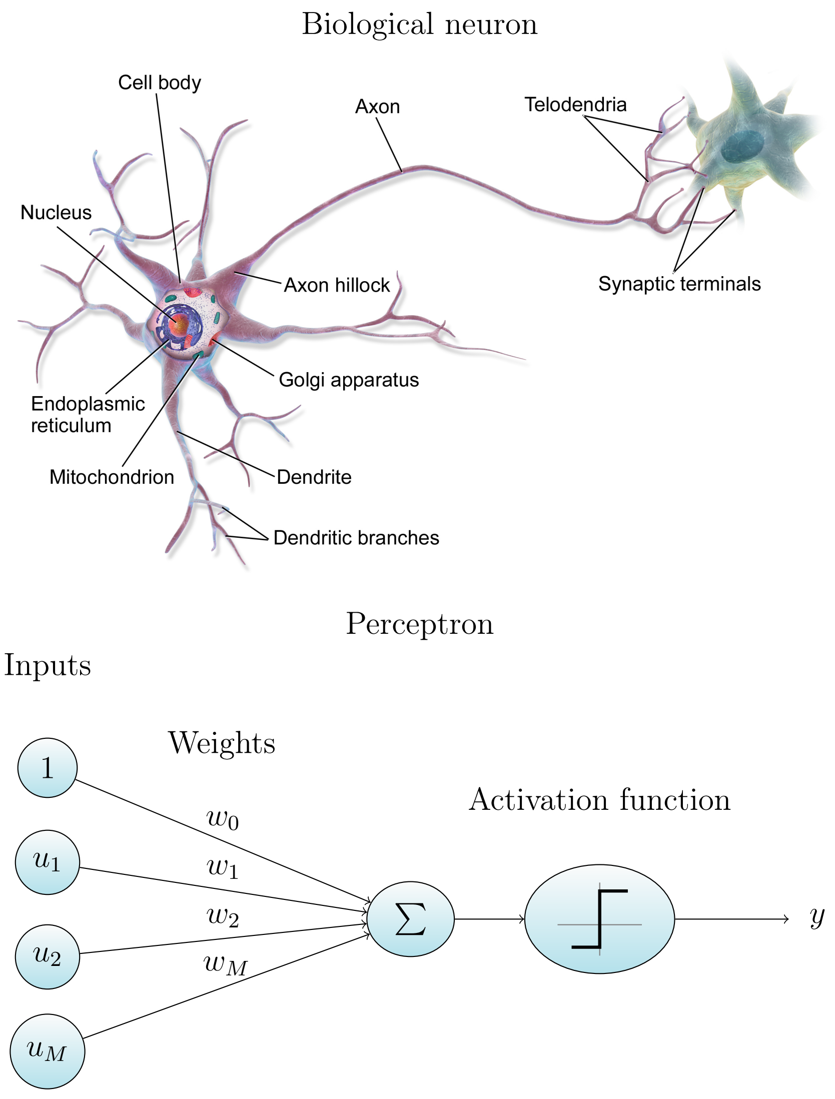

Figure 1.2.1: Simplification of the neuron's architecture for information processing. **Upper**: The biological neuron's model. Image source: [48]. **Lower**: *The perceptron*, a formal neuron's model. The input vector $\mathbf{u} = (1, u_1, u_2, \ldots u_M)$ is weighted and nonlinearly transformed via the activation function, creating a scalar output value y.

through dendrites. The soma performs non-linear processing over the input signals. If the total input is higher than a certain threshold, then an output impulse is produced [49]. This impulse is then transmitted through an axon, the long projection of the neuronal cell that conducts electrical impulses away from the cell body. In contrast to the biological neuron, the formal neuron called *perceptron* [6] (Fig. 1.2.1, lower) captures only the essential of the neuron's features: input connections, nonlinear transformation, and the output. Similarly to the neuron, the perceptron performs a thresholded transformation over the received information. As it can be seen from Fig. 1.2.1, perceptron is an extremely refined version of its biological counterpart and we stress that it is a purely mathematical concept used for the purpose of information processing.

A single-layer perceptron is mathematically described as:

$$y = f\left(\sum_{i=0}^{M} w_i u_i\right), \tag{1.2.2}$$

where $\mathbf{u} = (1, u_1, u_2, \dots u_M)$ is an input vector (Fig. 1.2.1, lower), $\mathbf{w} = (w_0, w_1, w_2, \dots w_M)$ is a weights row-vector, and f is a nonlinear *activation function*, e.g. step function, sigmoid, etc. Note, expression $\sum_{i=0}^{M} w_i u_i$ is the inner product between the vectors $\mathbf{u}$ and $\mathbf{w}$, where the first term $w_0 \cdot 1$ in Eq. (1.2.2) is the input bias. Thus, using a dot product notation $(\cdot)$ for vectors, Eq. (1.2.2) can be rewritten as:

$$y = f\left(\mathbf{w} \cdot \mathbf{u}\right). \tag{1.2.3}$$

Comparing Eq. (1.2.3) with physical models of the neuron, such as Eq. (1.2.1), we point out that there are no time-dependent variables involved in the model anymore. Thus, perceptron is a mathematical concept, a memoryless function, which maps the input vector $\mathbf{x}$ into the output y. It is necessary to highlight that even though there is a nonlinear activation function, the perceptron acts as a *linear* separator, a separation hyperplane in a n-dimensional phase space.

1.2.2 Feedforward neural networks

Not every problem is linear. In Fig. 1.2.2, there is an example of inputs which cannot be separated using a straight line, i.e. by a hyperplane in a 2-dimensional space. The input vectors $\mathbf{u} = (u_1, u_2)$ and the respective target results y of binary operation, known as XOR (eXclusive OR), are given by Table 1.1.

Therefore, linear separators such as perceptron cannot solve the XOR problem. To solve it, an additional layer of artificial neurons has to be introduced (Fig. 1.2.3). Now, the computation has to be done in two stages. In the first stage, the hidden layer $\mathbf{x}$ is a vector calculated as:

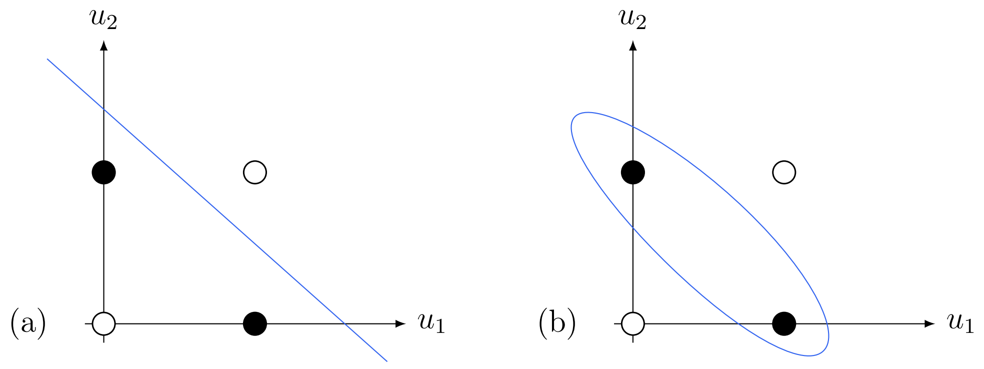

Figure 1.2.2: (a) The XOR operator cannot be resolved by a separation with a straight line (a hyperplane in 2D space). (b) Only transition to a higher-dimensional space, which is equivalent to drawing a curve in 2D space, may allow separation between classes. Coding: filled dots – 1, hollow dots – 0.

u_1	u_2	XOR(u_1, u_2)
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.1: The XOR binary function

$$\mathbf{x} = f_1\left(W^I\mathbf{u}\right), \tag{1.2.4}$$

where $\mathbf{u} = (u_1, \dots u_M)$ and $\mathbf{x} = (x_1, \dots x_N)$ are vectors. Matrix $W^I \in \mathbb{R}^{N\times(M+1)}$ is the input weights matrix, thus the hidden layer consists of multiple perceptrons given by row-vectors of W^I. The result of matrix-vector product $W^I\mathbf{u}$ is a vector, which is transformed by the perceptron's transfer function f_1. The function f_1 is applied element-wise, i.e. $f_1(\mathbf{x}) = (f(x_1), \dots f(x_N))$. Note that if $N = 1$, Eq. (1.2.4) is reduced to Eq. (1.2.3). In the second and final stage, the output of the network is calculated as a perceptron which receives the values from the hidden layer:

$$\mathbf{y} = f_2\left(W^R\mathbf{x}\right), \tag{1.2.5}$$

where $W^R \in \mathbb{R}^{K\times(N+1)}$ is the readout matrix, K is the output dimension (in Fig. 1.2.3 $K = 1$), and f_2 is another transfer function, which is often the identity. For brevity, we will assume $M := M' + 1$ and $N := N' + 1$, thus omitting bias inputs when describing $W^I \in \mathbb{R}^{N\times M}$ and $W^R \in \mathbb{R}^{K\times N}$, respectively.

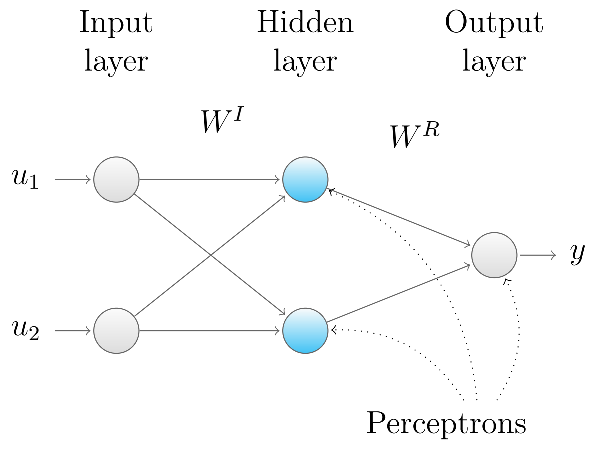

Figure 1.2.3: A feedforward neural network (FNN) solving the XOR problem. In this example, the hidden layer consists of two perceptrons $\mathbf{x} = (x_1, x_2)$, each of which receives two variable inputs u_1 and u_2, plus a constant input (not marked). The output layer y is a perceptron with two inputs x_1 and x_2 coming from the previous (hidden) layer, plus a constant input (not marked). The connection weights are given by the matrices W^I and W^R. Note that the network is not unique, i.e. there exist many possible configurations of W^I and W^R solving the same problem.

The computational process within a NN can be well illustrated by solving the XOR problem. Let the input and the readout matrices be

$$W^I = \begin{pmatrix} -10 & 20 & 20 \\ 30 & -20 & -20 \end{pmatrix}, W^R = \begin{pmatrix} -3 & 2 & 2 \end{pmatrix}, \tag{1.2.6}$$

and let the transfer functions be the following:

$$f_1 = \begin{cases} 1, & if \quad x > 0, \\ 0, & if \quad x \leq 0, \end{cases} \qquad f_2(x) = x. \tag{1.2.7}$$

Here f_2 is the identity, thus, the answer is a linear combination of the hidden layer's values. Consider an input $u_1 = 1$, $u_2 = 0$. Then, the hidden layer is computed as:

$$\begin{aligned} x = & \ f\begin{pmatrix} -10 + 20 \cdot u_1 + 20 \cdot u_2 \\ 30 - 20 \cdot u_1 - 20 \cdot u_2 \end{pmatrix} \\ = & \ f\begin{pmatrix} -10 + 20 \cdot 1 + 20 \cdot 0 \\ 30 - 20 \cdot 1 - 20 \cdot 0 \end{pmatrix} = \begin{pmatrix} f(10) \\ f(10) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned} \tag{1.2.8}$$

An answer is obtained, using the state of the hidden layer $\mathbf{x}$:

$$y = -3 + 2 \cdot x_1 + 2 \cdot x_2 = -3 + 2 \cdot 1 + 2 \cdot 1 = 1. \tag{1.2.9}$$

The other combinations of the input $\mathbf{u} = (u_1, u_2)$ can be verified in a similar fashion.

Artificial neural networks given by equations (1.2.4) and (1.2.5) are called *feedforward neural networks* (FNNs). FNNs with a single hidden layer of neurons have been proven [8, 50] to be able to approximate an arbitrary continuous function y_{target} on compact subsets of Euclidean space.

Theorem. *If $y_{target}(\mathbf{u}) \in C(I^M)$, where I^M is the M-dimensional hypercube $[0,1]^M$, $C(I^M)$ is the space of continuous functions, and f is a nonconstant, bounded, and monotonically-increasing continuous function, such that*

$$y(\mathbf{u}) = W^R f\left(W^I \mathbf{u}\right).$$

Then for each $\varepsilon > 0$ and $\mathbf{u} \in I^M$, there exist $W^I \in \mathbb{R}^{N \times M}$ and $W^R \in \mathbb{R}^{K \times N}$, such that $\|y(\mathbf{u}) - y_{target}(\mathbf{u})\| < \varepsilon$.

The *universal approximation theorem* [8] stated above not only provides formal requirements for FNNs but also implies that FNNs are a powerful mean of computation. Consisting of multiple perceptrons, on the other hand, FNNs can be regarded as a semblance of collective behavior in biological neuronal systems. The transition from biologically-justified to biologically-inspired architecture was motivated by computational efficiency to fit the existing technology.

1.2.3 Recurrent neural networks

FNN architecture provides a one-directional information processing flow[4]. Recurrent neural networks (RNNs), on the other hand, represent a class of NNs, the connection topology of which allows cycles. Such cycles are not present in FNNs, and RNNs can therefore be regarded as their extension. A schematic illustration is given in Fig. 1.2.4.

In RNNs, the hidden layer in the middle is typically referred to as the *recurrent layer*. As illustrated by the non-exclusively forward-directed connections in Fig. 1.2.4, the recurrent layer has internal connections which include recurrent loops. Because of the

[4]Hence the name, *feedforward* neural networks.

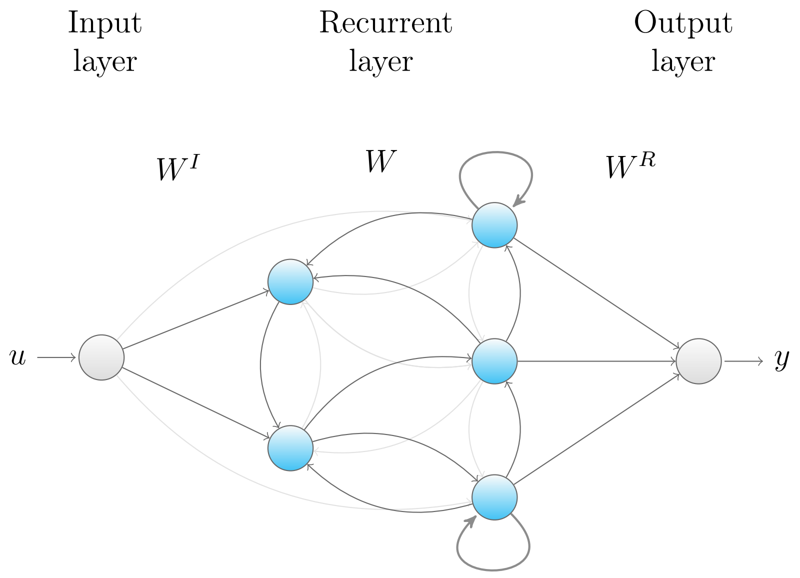

Figure 1.2.4: Schematic of information processing flow in a recurrent neural network (RNN). The recurrent layer, in the middle, retains the information and, therefore, the output of RNN depends on the previous inputs.

presence of recurrent loops, RNNs provide a distant analogy with the biological brain [46].

Mathematically, RNNs can be expressed as:

$$\mathbf{x}(n) = f\left(W^I \mathbf{u}(n) + W\mathbf{x}(n-1)\right),\ n = 1, \dots, T, \tag{1.2.10}$$

where matrix $W^I \in \mathbb{R}^{N\times M}$ is the input map, $W \in \mathbb{R}^{N\times N}$ is the map of the previous state of the recurrent layer, and $\mathbf{x}(n)$ is the *internal state* of a network at discrete time n. The result of computation $\mathbf{y}(n)$ is obtained as

$$\mathbf{y}(n) = W^R \mathbf{x}(n), \tag{1.2.11}$$

where matrix $W^R \in \mathbb{R}^{K\times N}$ is a readout map. As it can be seen from Eq. (1.2.10), the internal state of the RNN $\mathbf{x}(n)$ depends not only on current inputs, but also on the previous inputs. Such dependence on the previous inputs is the reason why RNNs are more complex objects comparing to FNNs, where information flows unidirectionally from input to output.

While FNNs are universal function approximators, RNNs can be regarded as *algorithms*. The main difference is that functions (or maps) do not have memory (they are *stateless*) while algorithms do. It has been shown, RNNs are universal approximators of dynamical systems [51]. Moreover, it has been shown that they are Turing equivalent [52]. Thus, RNNs constitute a more powerful class of NNs than FNNs. RNNs can be applied for such problems as system identification, inverse system identification, filtering, prediction, pattern classification, associative memory, data compression [53].

1.2.4 Neural networks training

1.2.4.1 Supervised vs unsupervised learning

The most striking idea behind feedforward NNs is a construction of computing network that can be reused for a wide range of computational problems. Tasks such as spam[5] detection, handwriting recognition, face recognition, time series forecasting and many others, can be regarded as an unknown function y_{target} approximation problems. For instance, in the case of spam detection, the function $y_{\text{target}}(\mathbf{u})$ has to return one of two values, true or false, depending if the argument $\mathbf{u}$ is a spam or not. In this example, $\mathbf{u}$ is a vector of characters representing an email message. In the case of handwriting recognition, the function $y_{\text{target}}(\mathbf{u})$ has to return a character (or a string of characters) depending on its input $\mathbf{u}$, which is a matrix of pixels representing an image. In the case of face recognition, $y_{\text{target}}(\mathbf{u})$ has to identify a face $\mathbf{u}$ in a database of known faces. In the last case, of time series forecasting, $y_{\text{target}}(\mathbf{u})$ has to predict the forthcoming (future) element(s) of time series $\mathbf{u}$.

Behind the training of the listed above examples, (a) there is a set of inputs $\{\mathbf{u}^{(m)}, m = 1, \ldots, N_{\text{total}}\}$, (b) there is a set of corresponding known outputs $\{y_{\text{target}}(\mathbf{u}^{(m)}), m = 1, \ldots, N_{\text{total}}\}$, and (c) the goal is to find a function $y(\mathbf{u})$, e.g. implemented by a NN. The function $y(\mathbf{u})$ is required to minimize a certain error measure $\|y(\mathbf{u}) - y_{\text{target}}(\mathbf{u})\|$. The tasks listed above, therefore, belong to a *supervised learning* class of problems. The goal of supervised training method is known as *generalization* of known outcomes. The present thesis is concentrated around problems solved with the help of supervised learning.

However, there are problems that do not have a predefined solution. Consider, for instance, a dataset of arbitrary data, in which there are hidden relations, e.g. clusters, to be uncovered. If no prior information is given about the number of clusters, their structure or size, then the system is trained with an *unsupervised learning* method. Kohonen networks [9] are an example of such self-organized NNs.

[5]Irrelevant or inappropriate email messages sent to a large number of recipients.

There also exists an intermediate approach known as a *reinforced learning* [54]. During the reinforcement learning process no explicit teacher signal is given. However, the trained system receives a reward upon successful completion of given task or penalty as a consequence of fail. For instance, reinforced learning is often used to teach intelligent agents (robots) solving motion problems, such as walking or object manipulations. There are no explicit instructions, how to balance the robot's body or position the limbs. However, upon completion of each training round, the agent receives a certain reward or "pleasure" function, depending on how close it was to the completion of given goal.

1.2.4.2 FNN training

Backpropagation is one of the most popular supervised learning methods used in FNNs training. The objective of the method is to find network weights W^I and W^R such that the summed square error is minimized

$$E^R = \frac{1}{2N_{\text{total}}} \sum_{m=1}^{N_{\text{total}}} \left\| y(\mathbf{u}^{(m)}) - y_{target}(\mathbf{u}^{(m)}) \right\|^2 .$$

Before the training, weights in W^I and W^R are initialized randomly, typically with small values. The backpropagation is performed iteratively. In each iteration, a *forward pass* given by formulas (1.2.4) and (1.2.5) is computed for each training sample. Then, during the *backward pass* weights are updated from the output to the input layer. This procedure allows the error to be incrementally decreased along the direction of the error gradient ∇E with respect to weights, i.e.

$$\nabla E = \sum_{m=1}^{N_{\text{total}}} \frac{\partial E^l(m)}{\partial w^l_{ij}} \tag{1.2.12}$$

according to the rule

$$\text{new } w^l_{ij} = w^l_{ij} - \alpha \nabla E, \tag{1.2.13}$$

where α is a (small) learning rate, l is the updated layer's index. This way, new readout weights $w^R_{ij} \in W^R,\ i = 1, \dots, K,\ j = 1, \dots, N$ and consequently new input weights $w^I_{ij} \in W^I,\ i = 1, \dots, N,\ j = 1, \dots, M$ are computed. The forward and backward passes are repeated again until a stop condition, e.g. training convergence. For more practical details about backpropagation algorithm see e.g. this tutorial [55].

Multiple modifications of the backpropagation technique exist, e.g. *incremental learning* where the weights are changed after each training sample individually, modifications with adaptive learning rate α, etc. The algorithm can also be extended to NNs with multiple hidden layers. However, there are several limitations of the backpropagation method family. One of the most important limitations is, like all gradient-descent methods, it is not guaranteed to find a global error minimum. Another major limitation is that training is often slowly converging. Lasting multiple iterations, it may require substantial computation resources to train a neural network.

1.2.4.3 RNN training

While RNNs have an algorithmic advantage over FNNs[6], their training is not as straightforward using error backpropagation. In addition to problems of local optima and slow convergence, the learning process is also driven through bifurcations of the dynamical system [53]. That may slow down or even disrupt the learning process.

Training an RNN is similar to training a FNN with a large number of hidden layers as we will explain in the following paragraphs. Fig. 1.2.5 (a) shows an example of an RNN. The input u_t together with the recurrent feedback x_t (blue arrow), is transformed by a nonlinear map f. Altogether, the input u_t, the feedback x_t, and the nonlinear map f constitute a recurrent layer producing the internal network state x_{t+1}. The resulting state x_{t+1} is the input to the next layer, where the readout map g finally returns the RNN's answer y_{t+1}.

By unfolding the RNN in time, one is able to obtain a FNN representation (Fig. 1.2.5 (b)). Unfolding for n steps is achieved by n replications of the recurrent layer. That results in a FNN with n virtual layers that also has n distinct inputs $u_t, u_{t+1}, \dots, u_{t+n-1}$. Therefore, the inner state x of the initial RNN is dependent on all the past inputs. That illustrates how RNNs implicitly store the information about the history of all the previous inputs.

To train such unfolded network, a gradient descent method can be used. This technique is called back-propagation through time (BPTT). During backpropagation training, gradients either increase or decrease at each time step. A consequence of the highly increased network's depth, is the creation of so-called *exploding* and, in other cases, so-called *vanishing* gradients [12]. Thus, BPTT becomes a non-trivial task requiring experimentation. Additional tricks (e.g. teacher forcing) are often required to achieve

[6]Here we have to remind that RNNs have the internal state (memory), while FNNs do not. Therefore, RNNs are equivalent to algorithms, while FNNs are memoryless maps. The presence of memory is a tremendous difference that makes RNNs closer to the real brain. Consequently, FNNs are a conceptually less complex class of NNs.

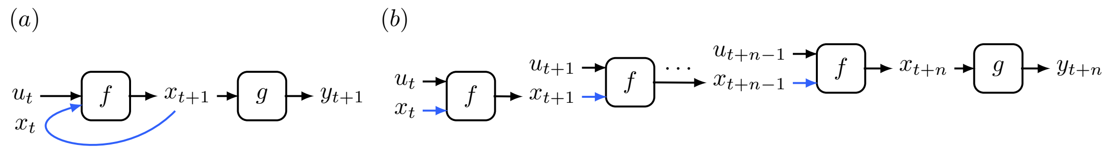

Figure 1.2.5: (a) A simple RNN where f is a nonlinear activation function and g is a readout map. (b) The same network, but unfolded in time for n time steps, which transforms it into an FNN with many layers.

convergence, see [53] for details. Finally, the BPTT procedure drastically increases computational costs for RNNs training, compared to FNNs with the same number of real layers.

Artificial neural networks have demonstrated potential in solving such difficult problems as object recognition [56], playing Go [57] and video games [58], skin cancer diagnosis [59] and others. However, all those algorithms are mostly implemented on general-purpose devices such as CPUs and GPUs[7]. On the other hand, implementation in dedicated hardware promises higher processing speeds and a much better energy efficiency. Because of inherit complex dynamics, physical systems realizing nonlinear delay dynamics are a viable candidate to implement neural networks directly in hardware [60, 61].

1.3 Applications of nonlinear delay dynamics

The diversity of dynamical behavior found in delay systems can be appreciated by the diversity of dynamical states found in their bifurcation diagram (see Fig. 1.3.1). Once translated into a discrete representation of dynamical networks, they share a strong analogy to dynamics widely exploited with artificial neural networks in the field of machine learning. All three regimes (fixed point, limit cycles, and chaotic regime) exhibited by DDEs found an application.

Chaotic regime can be used to hide an information signal in chaotic carrier. This idea is the basis of secure chaos communication where the broadband chaotic waveform replaces a classical sinusoidal carrier, bringing a ciphering (steganography) functionality to the transmitted information. Synchronization between chaos systems is the requirement for encryption/decryption of the signal [62]. Therefore, an identical pair of systems is required. Implementations of a delayed-feedback oscillator provide a major

[7]Central processing units and graphical processing units.

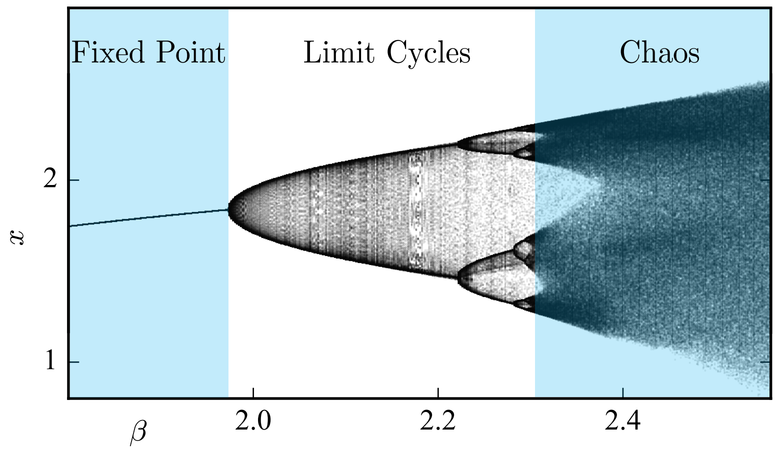

Figure 1.3.1: Bifurcations in the Ikeda DDE

improvement: such implementations significantly increase the phase space comparing to earlier demonstrations. The benefit of the approach is the increased difficulty to recover the encryption parameters. Another advantage of such systems is high-speed encoding achieving up to 10 Gb/s [45, 63].

Limit-cycle solutions of DDEs are also used for practical purposes. This concerns the microwave generation with very high spectral purity. The resulting ultrastable microwaves are beneficial for radar applications. The advantage of optoelectronic oscillators implementing the Ikeda dynamics over classical electromechanical resonator-based oscillators is in the ability to reach high oscillation frequencies while still featuring an extreme short term stability as required by radar systems. Moreover, the quality factor of optoelectronic oscillators is improving as the operating frequency increases [26, 64].

Systems with asymptotic *fixed point* solutions are getting higher attention in the nonlinear dynamics community due to the search for alternative information processing hardware [14]. Until now, decreasing transistors' sizes were able to speed-up the processors' clock rates. That enabled artificial neural networks, which demonstrated a greater potential in solving machine learning tasks unaccessible before [65, 66, 67, 56, 58, 57]. However, the process of miniaturization is fundamentally limited. The silicon industry is close to the limits. That may pause the development of artificial neural networks, relying on speed improvements of general-purpose electronics. A possible way to speed up those networks is to build specialized hardware natively supporting them. Reservoir computing (RC) is a promising RNN training approach that is based on separating creation of RNN from its training. This paradigm enables a rich variety of new computing hardware, including systems realizing DDEs. One of the main advantages of RC built on top of DDEs is the efficiency of delayed-feedback oscillator architecture.

Implemented in optics [32, 33, 34, 68], such devices can become a photonic alternative to present digital information processing instruments already facing the fundamental limits.

1.4 Conclusion

Introduced in the Chapter, nonlinear delayed-feedback systems are present in many physical processes [16]. In practice one may realize DDE of Ikeda type using delayed-feedback oscillator architecture. The architecture comprises three principal components: a nonlinear transformation, a delay, and a lowpass filter. Because of their complexity, delayed-feedback systems exhibit a rich variety of behaviors. In order to better understand the dynamics, we developed an analogy between DDEs and spatially-extended dynamical systems, approximated by a set of virtual oscillators placed on a temporal ring. That property along with inherent oscillator homogeneity achieved by the *time multiplexing* technique makes nonlinear delayed-feedback systems a tool of interest, e.g. to study synchronization phenomena in ensembles of coupled oscillators.

On the other hand, an analogy to spatially-extended systems connects dynamics of DDEs and those of recurrent neural networks (RNNs). RNNs are generalization of feedforward neural networks (FNNs), a simple model inspired by biological neuronal networks. Both RNNs and FNNs are used for information processing. However, due to increased complexity, RNNs are more difficult to train using methods that are typically applied to FNNs, e.g. gradient-descent. Among those difficulties are slow convergence, exploding and vanishing gradients, increases computational costs and others. Reservoir computing is an alternative RNN training method, solving the listed above problems. In addition, RNN allows computation directly by DDEs, in asymptotic fixed point regime. In its turn, DDEs give rise to new hardware [60, 61] implementing delayed-feedback oscillator architecture. Other applications of DDEs involve other dynamical regimes, such as limit-cycles (high spectral purity microwave generation [64]) and chaos (secure chaos communication [45, 63]).

Chapter 2

Chimera states in nonlinear delayed-feedback systems

Nonlinear delay dynamical systems modeled by delay differential equations (DDEs) can exhibit very complex behavior. Space-time representation allows to treat DDEs as homogenous networks of coupled oscillators. The self-organized patterns known as *chimera states* are characterized by spontaneous symmetry breaking in such homogenous networks. We describe the first experimental observation of chimera states in optoelectronics. The observed chimera patterns are highly robust, i.e. they can be found for a wide range of parameters and are stable with respect to the noise in the experimental setup.

2.1 Introduction

Until recently, there were considered only two extreme cases of synchronization in homogenous[1] networks of coupled oscillators: either synchrony (coherence) or chaos (incoherence). In [20] it was first demonstrated that both coherent and incoherent states can coexist. The network described by a phase equation Eq. (1.1.21) with oscillators continuously distributed in space (Section 1.1.4) was considered in [20]. As a reminder, the formula is given below:

$$\frac{\partial}{\partial t}\varphi(x,t) = \omega - \int G(x-x')\sin\left(\varphi(x,t) - \varphi(x',t) + \alpha\right)dx', \tag{2.1.1}$$

[1]Meaning that all the oscillators are identical, i.e. their natural frequencies are equal $\omega_x = \omega_{x'} \equiv \omega$.

Here the natural frequency ω is a constant indicating a homogenous network. Additionally, we have to point out that $G(y)$ is the distance-dependent coupling kernel[2], effectively reducing the number of possible connections from global, all-to-all coupling to a so-called *non-local* coupling, greater than the smallest oscillator distance and smaller than the network size. Then, under appropriate initial conditions, the network of oscillators Eq. (2.1.1) splits into two domains: a coherent and the other, incoherent. Such state of the network is illustrated Figure 2.1.1.

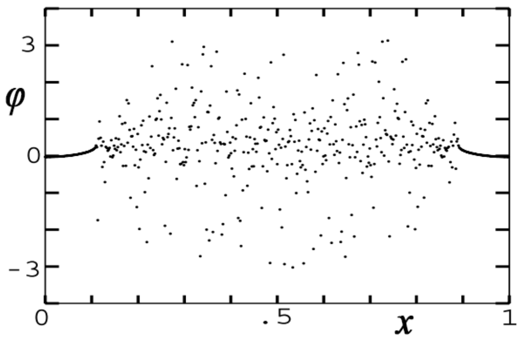

Figure 2.1.1: Instantaneous distribution of phases in Eq. (2.1.1) exhibiting coherence and incoherence: The coherence region is from normalized positions 0 to 0.1, and from 0.9 to 1. The network is made of discrete oscillators forming a closed chain (end of the chain connected to the beginning). There are 512 oscillators distributed over a chain with a length normalized to one. The vertical axis is in unit of radians. Image adapted from [20].

The majority of the oscillators' phases in Figure 2.1.1 are not synchronized, however, there is a small cluster of synchronized phases, which persists through the time evolution of Eq. (2.1.1). In [21] such coherent-incoherent motion was named a *chimera state*. According to the Greek mythology, Chimera was a hybrid creature composed of several animals. By analogy, chimera states simultaneously exhibit two different solutions: chaos and synchrony. Chimera states may provide a useful insight on such phenomena as pattern formation in networks of coupled oscillators of different nature [20, 21], ventricular fibrillation [22], scenarios of transition to turbulence similar to [69, 70, 71, 72]. Since the original theoretical work [20], confirmation of chimera state was found in physical systems such as liquid crystals [73, 74], chemical [75], mechanical [76, 77], and quantum oscillators [78].

In this Chapter, we describe the first experimental observation of chimera state in a delay system. We experimentally implement an optoelectronic laser wavelength nonlinear delayed feedback oscillator, which exhibits chimera states. For that purpose we exploit the analogy between DDEs and spatially-extended systems that was introduced in Chapter 1. Inspired by chimera states in networks of coupled oscillators, chimera

[2]Such as $G(y) = \frac{K}{2}\exp(-K|y|)$ in [20] or $G(y) = (1 + A\cos x)/2\pi$ in [21].

states in nonlinear delay dynamics [79, 80, 81] are sustained self-organized patterns recurring over the delay interval. An essential finding is that chimera patterns can be modeled by *autonomous* DDEs, i.e. without any external input breaking the system's symmetry.

We characterize chimeras in nonlinear time-delayed systems that are a generalization of the Ikeda DDE. We find that striking behavior of this type arises in a wide domain of system parameters. With increase of the amplification gain, chimera solutions are destroyed, giving rise to a spatiotemporal intermittency and high-dimensional chaos. On the other hand, the considered delayed-feedback model exhibits multistability, i.e. different chimera configurations can coexist within the same parameter set. One of potential applications exploiting high system's multistability is fast information storage, e.g. [38]. The benefit of our implementation is that it shares the delayed-feedback architecture, and thus, the dynamical properties with alternative optoelectronic computing systems [33]. This suggests easier integration between both systems.

2.2 Observation of chimera states in delayed-feedback systems

In Chapter 1, we discussed the route to chaos in DDEs depending on the gain β of a nonlinear function f. Smaller values of β corresponded to fixed-point solutions, while bigger values of β gave rise to limit cycles. The limit cycles led to chaos through the cascade of period doubling bifurcations. In the case of fixed point, small values β resulted in a near-flat shape of the nonlinear function f near extremum. In contrast, higher values of β resulted in sharp maxima giving rise to chaotic behavior. Subsequently, use of a nonlinear function f with two different extrema, a broad minimum and a sharp maximum, will result in one of two solutions: a fixed point or chaos. That type of function is then required in order to break the symmetry in a nonlinear DDE.

2.2.1 Ikeda DDE. The Airy function and two coexisting attractors

Behavior of the following DDE is determined by the nonlinear function f:

$$\tau \dot{x}(t) + x(t) = f\left(x(t-\tau_D)\right), \tag{2.2.1}$$

where parameter τ is a linear decay time and τ_D is time delay. As a nonlinear function f, consider the Airy function with asymmetric extrema:

$$f(x) = \frac{\beta}{1 + m \cdot \sin^2(x + \Phi_0)}, \tag{2.2.2}$$

where β is the amplification, or nonlinear function stretching parameter (Figure 2.2.1). We take β as a control parameter and, for definiteness, fix the other parameters $m = 4.7$ and $\Phi_0 = -0.4$.

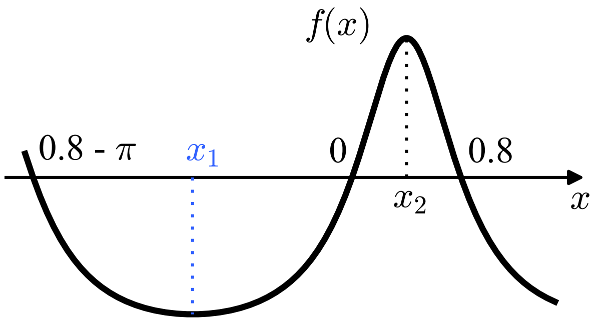

Figure 2.2.1: The Airy function vertically centered around the origin, $m = 4.7$, $\Phi_0 = -0.4$.

To understand the influence of f on the behavior of DDE, a one-dimensional iterated map is used:

$$\begin{aligned} x_n &= f(x_{n-1}) - f(0) \\ &= \beta\left(\left[1 + m \cdot \sin^2(x_{n-1} + \Phi_0)\right]^{-1} - c\right), \end{aligned} \tag{2.2.3}$$

where the constant term $c = \left(1 + m \cdot \sin^2 \Phi_0\right)^{-1}$ has been subtracted to vertically align f around zero.

The function f has asymmetry: it has a broad minimum $f(x_1)$ and a narrow maximum $f(x_2)$ (Figure 2.2.1). That follows, f has two *attractors* A_1 and A_2, toward which the system tends to evolve starting with close initial conditions. Those initial conditions are known as *basins of attraction*. The first attractor A_1 is a negative fixed point (Figure 2.2.2). It has a basin of attraction $x_0 \in (0.8 - \pi; 0)$. The second attractor A_2, depending on β, is either a fixed point, a limit cycle, or chaotic motion. The corresponding basin of attraction is $x_0 \in (0; 0.8)$. Therefore, difference equation Eq. (2.2.3) has two distinct coexisting solutions depending on the initial condition x_0, as it is shown in Fig. 2.2.2.

Two solutions of the iterative map correspond to distinct attractors of the Airy function. The question is how such asymmetric function can influence the DDE dynamics.

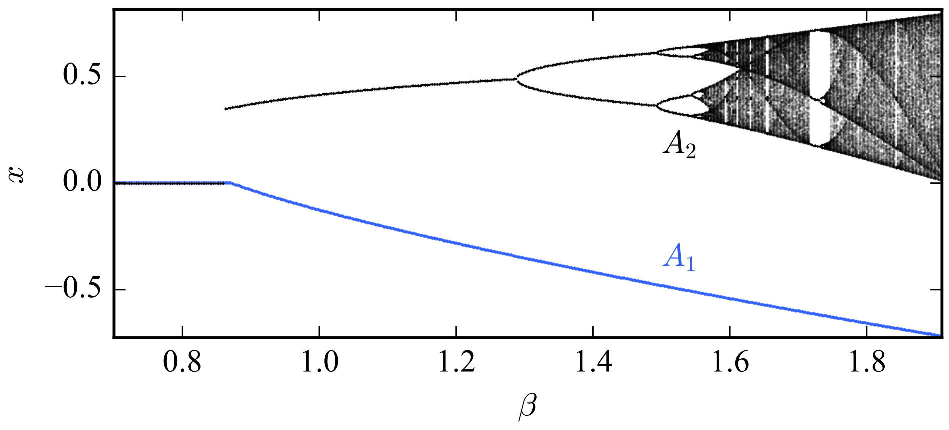

Figure 2.2.2: Bifurcation diagram of one-dimensional map Eq. (2.2.3). Depending on the initial condition x_0, solution x_n tends to one of two attractors A_1 and A_2.

2.2.2 Transient dynamics of Ikeda DDEs

Extending the one-dimensional map Eq. (2.2.3) to a DDE of Ikeda type gives

$$\tau\dot{x}(t) + x(t) = \frac{\beta}{1 + m \cdot \sin^2\left[x(t-\tau_D) + \Phi_0\right]}, \tag{2.2.4}$$

where τ_D is a time delay and $\tau \ll \tau_D$ is a fast-scale system response time. Let us fix parameters $\beta = 2$, $m = 4.7$ and $\Phi_0 = -0.4$, then the function f has two attractors, A_1 is a fixed point and A_2 is a chaotic attractor. Normalized with respect to time delay τ_D, Eq. (2.2.4) becomes

$$\varepsilon\dot{x}(s) + x(s) = \frac{\beta}{1 + m \cdot \sin^2\left[x(s-1) + \Phi_0\right]}, \tag{2.2.5}$$

where $s := t/\tau_D$ is the rescaled time in dimensionless units, $\varepsilon := \tau/\tau_D$, $\varepsilon \ll 1$ is the normalized system response time, and $x(s-1)$ is the delay term. With forced sine-wave initial conditions $x(s) = \sin(2\pi s), s \in [-1, 0]$, switching between the two attractors A_1 and A_2 arises. That results in partially regular and partially chaotic behavior on the delay interval (Fig. 2.2.3, (a)). However, this motion is transient: the coherent low plateau shrinks with time (Fig. 2.2.3 (b-c)) and eventually dies soon after 600 delay intervals (Fig. 2.2.3 (d)). The dynamics becomes purely chaotic, given by the attractor A_2.

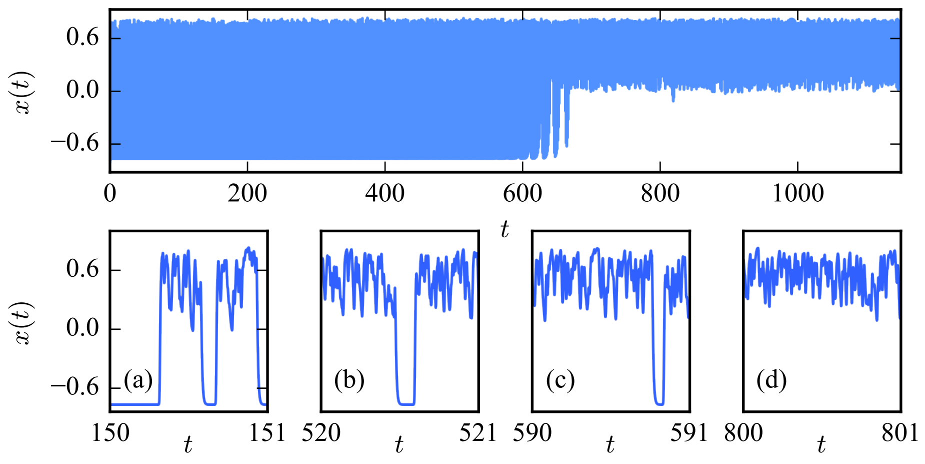

Figure 2.2.3: Transient coherent-incoherent motion in Ikeda dynamics Eq. (2.2.5). **Upper**: Time trace of $x(t)$ dynamics for 1200 delay intervals. **Lower**: Zoom-in of delay-length intervals revealing shrinking of the coherent regions and their eventual disappearance. Parameters $\beta = 2$, $\varepsilon = 5 \cdot 10^{-3}$, $m = 4.7$, $\Phi_0 = -0.4$.

This phenomenon, when the asymptotic dynamics is given eventually by one of two co-existing attractors, is common for different classes of DDEs, i.e. also for Eq. (2.2.1). In the literature, such phenomenon is called *coarsening* [25, 82, 83]. Coarsening means that a bistable system[3] initially exhibiting both states on delay interval, eventually is restricted to motion along only one of the two original attractors. In [82] coarsening was mathematically proved for a class of piecewise smooth functions f. Another DDE, with a linear delayed feedback was demonstrated in [25]: $\dot{x} = -f(x) + \beta x(t - \tau_D)$, where $f(x) = x(x + 1 + a)(x - 1)$ is the nonlinear function, a is the measure of global asymmetry, $\tau_D > 10$ is long delay, and β is the feedback gain. There, the system had two stable solutions: a lower and an upper plateau. The initial conditions were forced such that both, the lower and upper plateaus are assigned inside the delay interval. However, after some transient time, one of the plateaus ceased to exist, giving rise to a single stable regime.

Similarly, the bistable system Eq. (2.2.5) is attracted to one of its two attractors. Employing the space-time plot visualization scheme introduced in Section 1.1.3, we can observe the coarsening process (Fig. 2.2.4). The new question is if Eq. (2.2.5) can be modified in such a way that the bistable system, instead of falling into one of its attractors, can switch between both of them. In other words, can the transient dynamics depicted in Figs. 2.2.3 and 2.2.4 be stabilized?

[3]I.e. with two co-existing attractors A_1 and A_2.

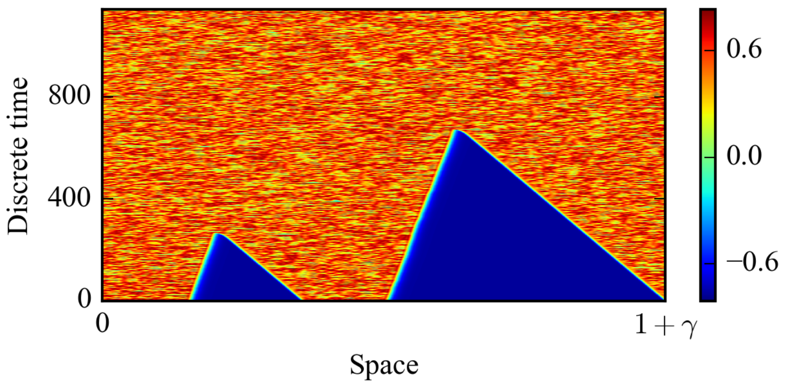

Figure 2.2.4: Coarsening in the Ikeda dynamics described by Eq. (2.2.5)

2.2.3 Chimera states in a bandpass Ikeda model

Chimera states are exhibited by networks of nonlocally, or long-range coupled oscillators [20]. By analogy, a long-range coupling is required to observe stable chimera patters in a DDE. The reason why the coherent-incoherent motion in Eq. (2.2.5) was transient, is likely to be a short-range coupling provided by the differential term $\varepsilon\dot{x}$. In order to introduce a long-range coupling, an additional term is needed. For instance, an integral term that is dependent on all the previous history of a DDE. Indeed, delayed-feedback systems with an integral term were reported to exhibit qualitatively new solutions. One example of such solutions are period-1 oscillations with strongly asymmetric waveforms [84]. The asymmetry was tunable depending on a nonlinear feedback offset phase. Another example are so-called *chaotic breathers* evolving on a slow timescale comparing to time-delay [85].

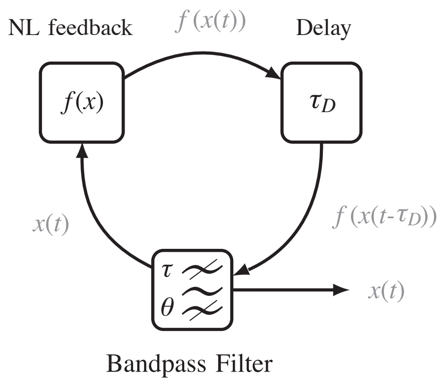

Figure 2.2.5: Ikeda oscillator with a bandpass filter

The transient dynamics in the Ikeda DDE physically corresponds to a dynamics in a feedback loop with a *lowpass filter* (Fig. 1.1.3). Replacing the lowpass with a *bandpass* filter, the dynamics of a bistable system switching between two attractors can be stabilized. Consider a bandpass filtered system (Fig. 2.2.5) described in the Fourier domain:

$$H(\omega) = \frac{(i\omega\theta)G_0}{(1+i\omega\theta)(1+i\omega\tau_0)} = \frac{I_F(\omega)}{I_D(\omega)} \tag{2.2.6}$$

where G_0 is the gain of the filter. Converting from the Fourier into the time domain, an Ikeda DDE with an integral term $\frac{1}{\theta}\int_{t_0}^{t} i_F(\xi)d\xi$ is then obtained:

$$\frac{1}{\theta}\int_{t_0}^{t} i_F(\xi)d\xi + \left(1+\frac{\tau}{\theta}\right)\cdot i_F(t) + \tau\frac{di_F}{dt}(t) = i_D(t), \tag{2.2.7}$$

where $i_D(t) = f\left(x(t-1)\right)$ is the input of the bandpass filter and $i_F(t) = x(t)$ is its output. In the normalized form, integro-differential Eq. (2.2.7) reads as

$$\delta\int_{s_0}^{s} x(\xi)d\xi + (1+\varepsilon\delta)\cdot x(s) + \varepsilon\frac{dx}{ds}(s) = f\left(x(s-1)\right), \tag{2.2.8}$$

where $s = t/\tau_D$ is normalized time, $\varepsilon = \tau/\tau_D \ll 1$ and $\delta = \tau_D/\theta \ll 1$ are normalized parameters representing differential and integral time characteristics of the dynamics. As in Section 2.2.1 we use the Airy nonlinearity as nonlinear function $f(x)$. Taking into account that $\varepsilon\delta \ll 1$, the term $\varepsilon\delta\cdot x(s)$ is small and can be neglected. Then, Eq. (2.2.8) is rewritten as a system of two DDEs:

$$\begin{aligned} \varepsilon\dot{x}(s) &= -x(s) - \delta\cdot y(s) + f\left(x(s-1)\right), \\ \dot{y}(s) &= x(s). \end{aligned} \tag{2.2.9}$$

The difference between (2.2.9) and the original DDE (2.2.5) is the presence of the integral term $\delta y = \delta\int_{s_0}^{s} x(\xi)d\xi$ with $\delta > 0$. Setting $\delta = 0$ therefore reduces Eq. (2.2.9) to (2.2.5).

The role of the control parameter β can be explained using the bifurcation diagram in Fig. 2.2.6. The behavior is determined by both attractors A_1 and A_2 of the iterated map Eq. (2.2.3) for dynamics shown in Fig. 2.2.2. Indeed, the lower branch influenced by the attractor A_1 remains in a state of equilibrium, while the upper branch determined by the attractor A_2 undertakes a cascade of period doubling bifurcations leading to

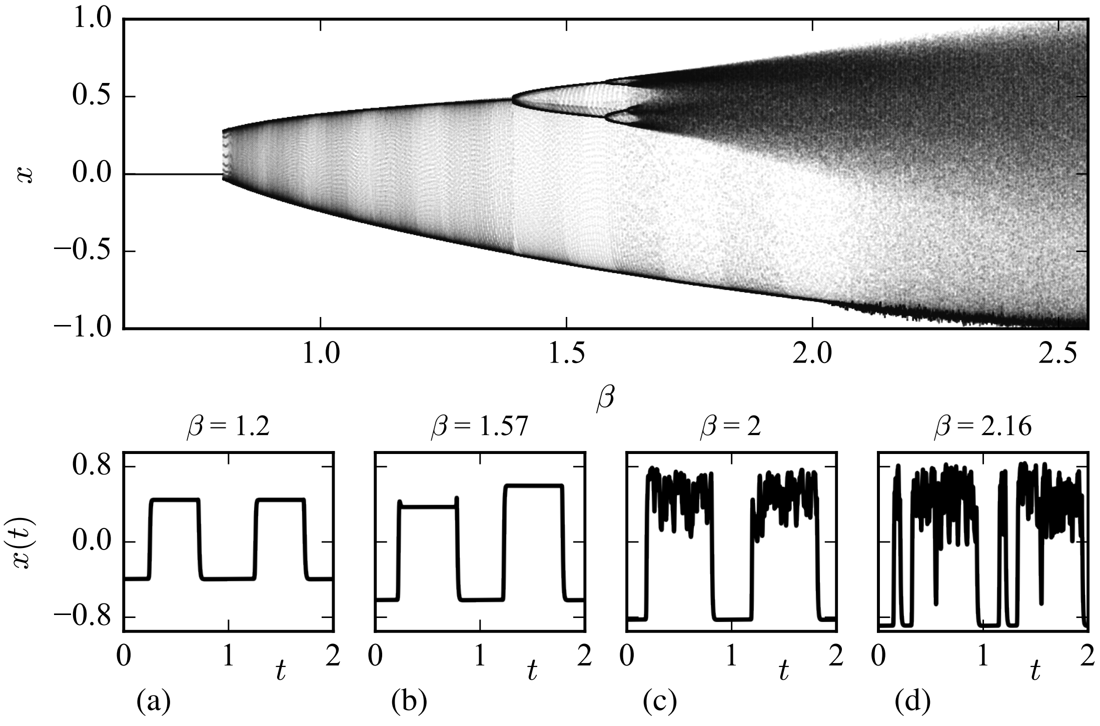

Figure 2.2.6: Upper: Bifurcation diagram for coherent-incoherent motion in Ikeda DDE with a bandpass filter. The Hopf bifurcation gives birth to a stable period-1 limit cycle at $\beta \simeq 0.8$. Then, a cascade of period doubling bifurcations in the upper branch occurs until reaching singularity (chaotic behavior) at $\beta \simeq 1.65$, giving rise to a *chimera state*. For $\beta > 2$, spatiotemporal intermittency arises. **Lower**: Time traces of (a) Periodic motion, (b) First period doubling bifurcation, (c) Chimera state, and (d) Spatiotemporal intermittency are depicted over two successive delay intervals. Note that all period doubling bifurcations occur in the upper branch of the bifurcation diagram, influenced by the attractor A_2.

chaos. Such a behavior results in coherent and incoherent motions that repeat within the delay interval, representing *chimera state*.

In order to illustrate the spatio-temporal character of the dynamics, space-time plots, visualization schemes from Section 1.1.3, are used (Fig. 2.2.7). There, the evolution of dynamics, spanning several thousands of delay intervals is illustrated with the color representation encoding the amplitude of the variable x. Red colors correspond to positive, blue to negative amplitude values of x, respectively. In contrast to Fig. 2.2.4, the coherent-incoherent motion over a delay interval does not shrink with time and is stable for bifurcation parameter β between 1.7 and 2.1 (Fig. 2.2.7 (b)). The shown dynamics therefore corresponds to coherent and non-coherent states of the underlying circular network evolving in discrete time.

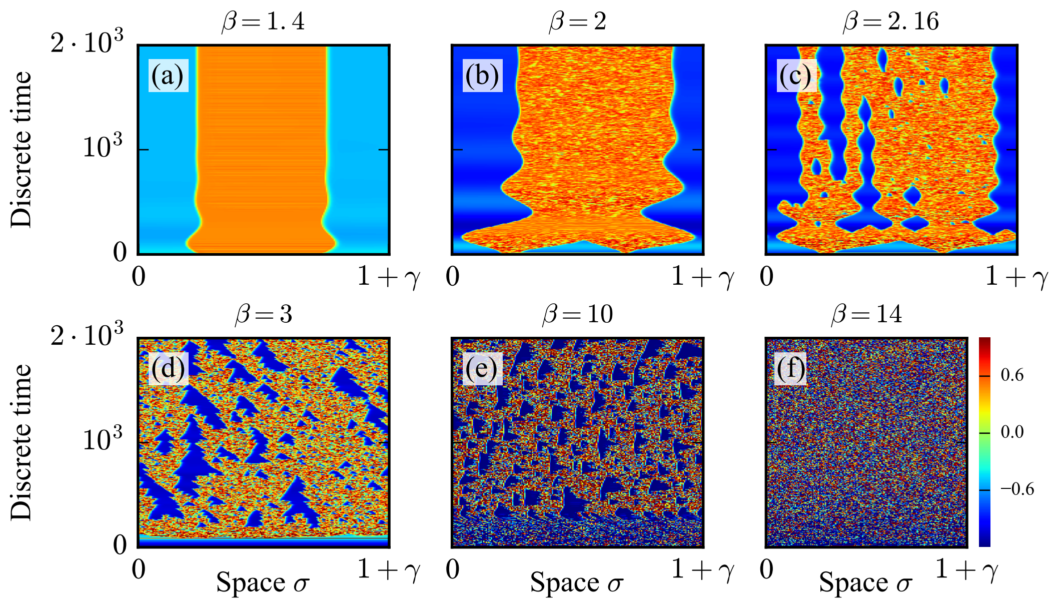

Figure 2.2.7: Spatiotemporal representation of periodic motion (a), chimera state (b), spatiotemporal intermittency (c-e), and chaos (f) in the system Eq. (2.2.9), $\varepsilon = 5 \cdot 10^{-3}, \delta = 8 \cdot 10^{-3}$.

Soon after $\beta > 2$, the coherent and chaotic regions start to be irregularly interspersed ($\beta = 2.16$, Fig. 2.2.7 (c) and corresponding time trace Fig. 2.2.6 (d)). This regime is known as *spatiotemporal* intermittency. Finally, with further increase of $\beta > 13$, the system described by Eq. (2.2.9) becomes turbulent in the form of spatiotemporal chaos (Fig. 2.2.7 (f)).

2.2.4 Optoelectronic experimental setup

Previously we identified several components required to observe coherent-incoherent patterns in DDE (Fig. 2.2.5). The first one is a nonlinear function with broad minimum and sharp maximum. The second required component is the delay line which should store the state of the virtual network of oscillators. The final requirement is a bandpass filter which should provide a long-range coupling within the network. To satisfy those requirements, we employ a Fabry-Pérot interferometer (nonlinear function), an electronic delay line, and an electronic bandpass filter, respectively.

Figure 2.2.8 schematically illustrates our experimental setup for chimera states observation[4]. The first component, a tunable laser diode is a light source, which allows for

[4]The optoelectronic setup was designed and implemented by L. Larger.

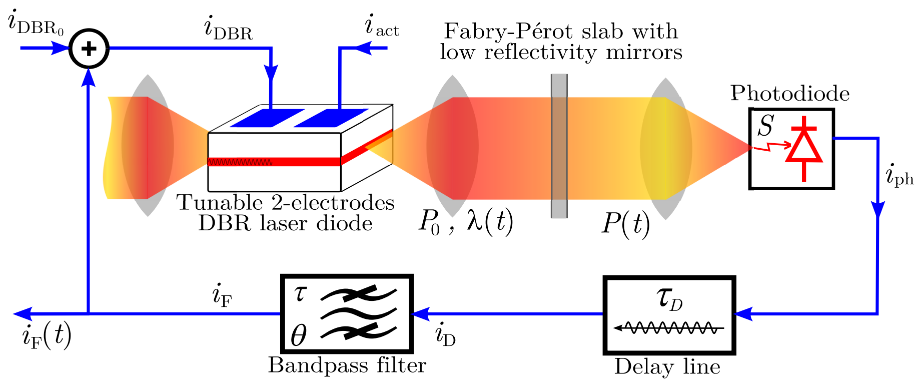

Figure 2.2.8: Schematics of nonlinear oscillator with delayed feedback, the experimental setup to observe chimeras

a monochromatic light beam of power P_0. The beam's wavelength $\lambda(t)$ is within the infrared light range and is proportional to the control current i_{DBR}:

$$\lambda = \lambda_0 + S_\lambda i_{DBR} \tag{2.2.10}$$

where $\lambda_0 \simeq 1549.8$ nm, and $S_\lambda \simeq 0.19$ nm/mA is a constant tuning slope efficiency. Emitted by the laser diode, light is collimated by a lens. Then the light passes through the second component, a Fabry-Pérot interferometer, a 5 mm thick glass plate. The interferometer's parallel input and output faces were coated with pairs of dielectric layers to obtain plane mirrors with intensity reflection coefficient $R \simeq 41\%$ (Fig. 2.2.10, left). The role of the interferometer in our setup is to nonlinearly transform the light wavelength into optical intensity. That nonlinear transformation is described by the Airy function $f(\Phi)$:

$$f(\Phi) = \frac{P}{P_0} = \frac{A}{1 + m \cdot \sin^2 \Phi}, \tag{2.2.11}$$

where the input variable Φ is the signal of the nonlinear system, the instantaneous wavelength of the laser. Other parameters, A and m for a Fabry-Pérot interferometer are defined through the reflection coefficient R: $A = (1-R)^{-1} \simeq 1.69$ and $m = 4R/(1-R)^2 \simeq 4.72$. The optical intensity transmitted through the Fabry-Pérot interferometer is then detected by a photodiode, which converts the optical intensity fluctuations P into voltage fluctuations:

$$u_{ph} = R_{ph}\, S\, P, \tag{2.2.12}$$

where $S = 0.9$ A/W is the photodiode sensitivity and R_{ph} is the resistance of a typical transimpedance amplifier used to convert the photocurrent into a voltage. The resulting product $R_{ph}\,S$ (in V/W) is the conversion efficiency of an integrated amplified photodiode.

The detected electrical signal passes the fourth component of the setup, a digital electronic delay line[5] containing N elements and running at clock frequency f_{CLK}. As a result, the signal is delayed by delay τ_D:

$$u_D(t) = u_{ph}(t - \tau_D) = u_{ph}(t - N_{\text{FIFO}}/f_{\text{CLK}}). \tag{2.2.13}$$

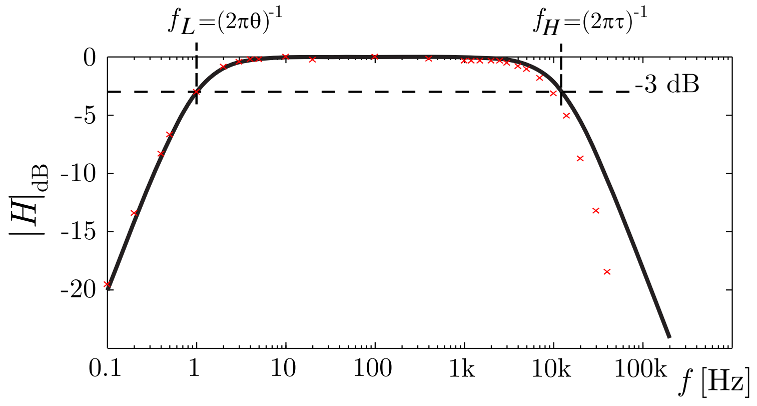

Figure 2.2.9: Bode's magnitude $|H(\omega)|$ plot of a second order bandpass filter. Theory: solid line, experiment: red crosses.

The last component, an electronic bandpass filter is used to establish the differential process ruling the dynamics of the oscillator loop. The bandpass filter is characterized by its high cutoff frequency $f_H = (2\pi\tau)^{-1} \simeq 12.5\,kHz$ and the low cutoff frequency $f_L = (2\pi\theta)^{-1} \simeq 1\,Hz$. Then, assuming a bandpass filtering in its simplest form, the corresponding Fourier filter is:

$$H(\omega) = \frac{I_F(\omega)}{I_D(\omega)} = \frac{(i\omega\theta)G_0}{(1+i\omega\theta)(1+i\omega\tau)} \xrightarrow{FT^{-1}} \tag{2.2.14}$$

$$\frac{1}{\theta}\int_{t_0}^{t} i_F(\xi)d\xi + \left(1+\frac{\tau}{\theta}\right)\cdot i_F(t) + \tau\cdot\frac{di_F}{dt}(t) = i_D(t) \tag{2.2.15}$$

where the delayed signal $i_D(t) = G_0 u_D(t)$ is the filter's input, $i_F(t)$ is the filter's output, $\theta = (2\pi f_L)^{-1} \simeq 0.16\,s$ and $\tau = (2\pi f_H)^{-1} \simeq 12.7\,\mu s$ are parameters representing

[5]The delay line is implemented via FIFO (first-in-first-out) memory chip.

integral and differential characteristic response times of the dynamics. The modulus of the Fourier filter (2.2.14) (Fig. 2.2.9) is then expressed in decibels (dB) as:

$$|H|_{\mathrm{dB}} = 20 \cdot \log \frac{G_0 \cdot 2\pi\theta f}{\sqrt{\left(1 + (2\pi\theta f)^2\right) \cdot \left(1 + (2\pi\tau f)^2\right)}}. \tag{2.2.16}$$

To close the loop, an adder is employed to provide the laser's control current $i_{DBR}(t)$:

$$i_{DBR}(t) = i_{DBR_0} + i_F(t). \tag{2.2.17}$$

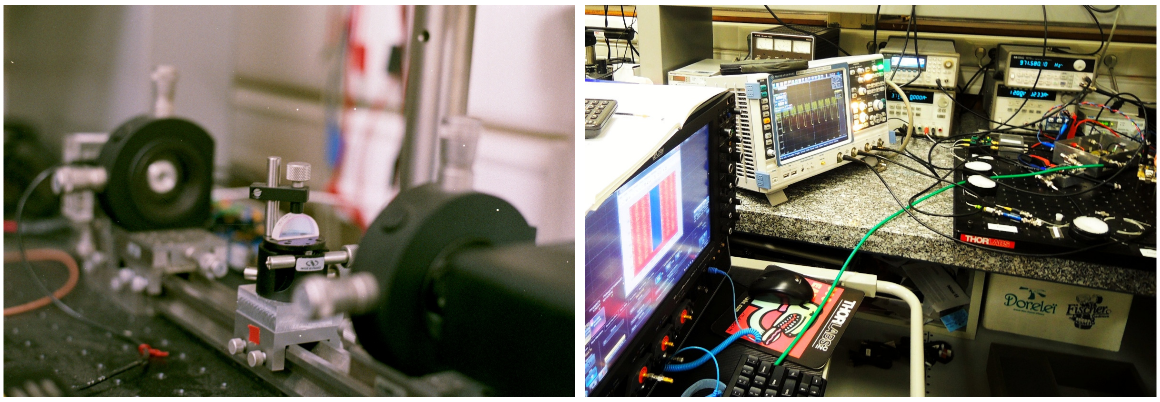

Figure 2.2.10: The experimental setup. **Left**: The Fabry-Pérot interferometer between the laser and photodetector. **Right**: The live setup that demonstrates chimera waveforms on each display of the two oscilloscopes.

The experimental setup in action can be seen on the right of Fig. 2.2.10. To summarize, the amplification parameter β can be changed proportionally to the linear gain G_0 of the bandpass filter. The phase offset Φ_0 is determined by current offset i_{DBR_0}. Long timescale parameter $\delta = \tau_D/\theta$ is proportional to the time delay $\tau_D = N_{\mathrm{FIFO}}/f_{\mathrm{CLK}}$, while the short time-scale parameter $\varepsilon = \tau/\tau_D$ is inversely proportional to τ_D. Therefore, in order to adjust δ and ε, one can either tune the cutoff frequencies f_L and f_H (adjusting the filtering parameters θ and τ) or the time delay τ_D. The time delay τ_D is adjusted by either changing the number of elements in the delay line N_{FIFO}, or by modifying the delay line's clock frequency f_{CLK}. The last parameter f_{CLK} is easily tuned in our setup, thus the parameters δ and ε are in most cases adjusted by changing the time delay τ_D, which is typically set to be around 0.1–10 ms.

2.2.5 Comparison between simulations and experiments

It is essential to verify the dynamics produced by the experimental setup against our theoretical model. In order to simulate the physical system's behavior, we perform numerical integration. The utilized equations are (2.2.9) and (2.2.2) with fixed parameters $m = 4.72$ and $\Phi_0 = -0.4$, corresponding to the expected values in experiment. It is also necessary to introduce what we call a *parametric point* — a point on a parameter plane (ε, δ). If not stated otherwise, the calculations are performed within the parametric point $A : (\varepsilon = 0.005, \delta = 0.008)$. For the numerical integration, we employ a fourth order Runge-Kutta method with a fixed step size $h_s = \varepsilon/10 = 5 \cdot 10^{-4}$. For verification, we compare the results using a third order Bogacki-Shampine method[6] with an adaptive step size. Both methods result in dynamical solutions of the same kind, with the reasonable differences in accuracy corresponding to their orders.

In experiment, we try to match the parametric point A, in which we perform the numerical simulations. In practice, any other pair of values (ε, δ) with sufficiently large $\delta > 0$ could be utilized, as it follows from the next Section. The low-pass filter's parameter $\tau \simeq 12.7\,\mu s$ and the high-pass filter's parameter $\theta \simeq 0.317\,s$ correspond to the dimensionless values of $\varepsilon = 0.005$ and $\delta = 0.008$, respectively. The delay time is $\tau_D \simeq 6.15\,ms$.

A striking similarity between numerical and experimental results is highlighted in Fig. 2.2.11. As β increases, chimera solution is destroyed, giving rise to a turbulent-like behavior. See also Fig. 2.2.6 where the transition to turbulent-like motion is illustrated with the help of the bifurcation diagram of the model Eq. (2.2.9).

2.3 Multistability in the bandpass DDE

2.3.1 Coexistence of chimeras and breathers

The chimera state is not the unique solution of the system described by Eq. (2.2.9). As we found [80], *chaotic breathers* [85] (Fig. 2.3.1 (b)) always coexist with chimeras. Similarly to chimeras, breathers consist of two parts: a regular and a chaotic part. However, chimeras evolve on a timescale of delay[7], while chaotic breathers evolve on a much slower time scale of the order of the integral characteristic time θ. The period of a chaotic breather solution takes hundreds of delay intervals; the exact value depends

[6]Code implementing Bogacki-Shampine method is provided under MIT license by http://pydelay.sourceforge.net.

[7]As already discussed, chimeras span over an interval close to delay $[0, 1 + \gamma]$.

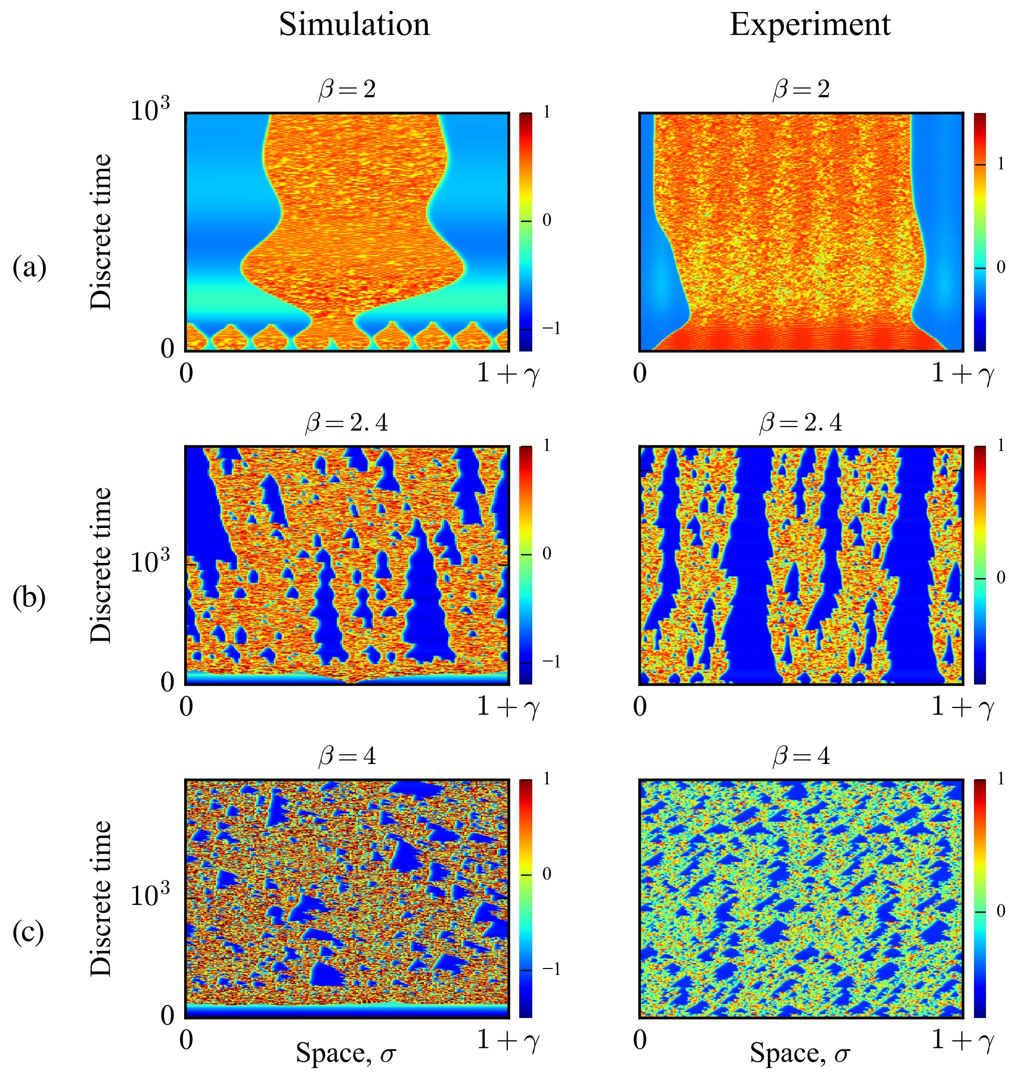

Figure 2.2.11: Comparison between simulation (**left**) and experiment (**right**): (a) Chimera state; (b)-(c) Transition to turbulent-like spatiotemporal intermittence when progressively increasing β.

on the parameter δ^{-1}. Regular and chaotic behaviors are alternated resembling, on the global timescale, neuronal bursting [86].

To highlight the difference between chimeras and breathers, we employ nullcline analysis, technique often applied to study qualitatively the behavior of differential equations. In the $(x, y)-$phase plane, a *nullcline* with respect to the variable x is a set of points such that $\dot{x} = 0$. Therefore, the first nullcline is derived from Eq. (2.2.9):

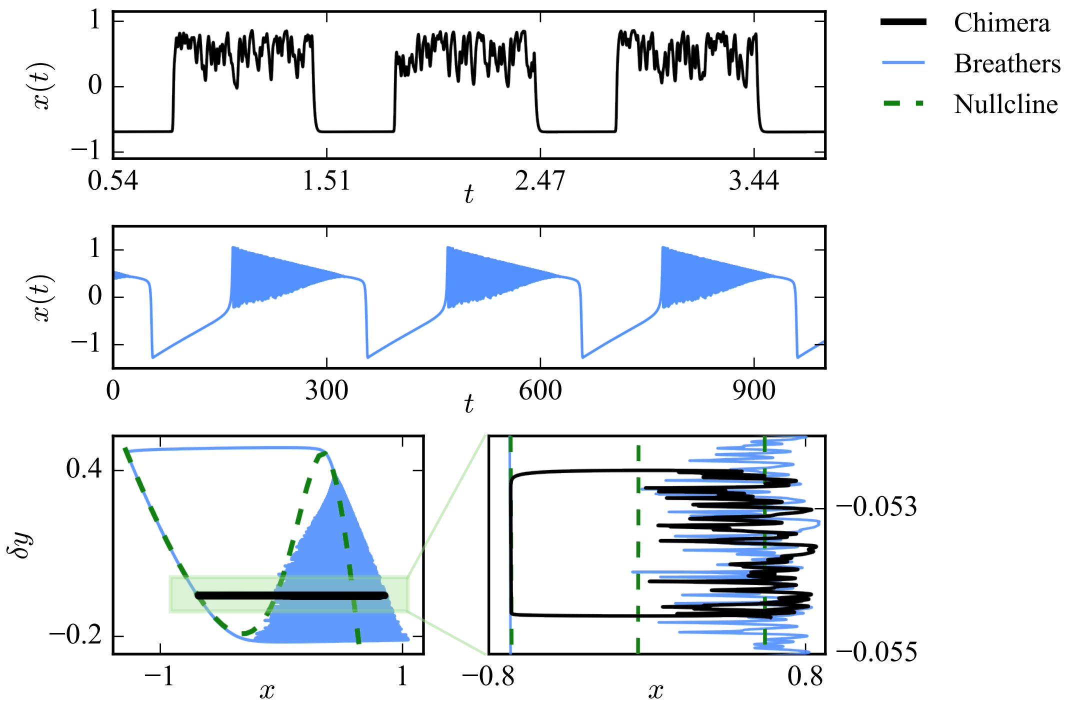

Figure 2.3.1: Coexistence of chimeras and chaotic breathers: (a) Time trace of chimera state (black); (b) Time trace of chaotic breathers; (c) Chaotic breathers (light blue) and chimera (black) in a two-dimensional phase space along the x-nullcline of Eq. (2.2.9). The timescale of chimeras is 310 faster than the one of breathers.

$$\delta y = -x + \frac{\beta}{1 + m\sin^2(x + \Phi_0)}, \tag{2.3.1}$$

where $\beta = 2$, $\Phi_0 = -0.4$, and $\delta = 8 \cdot 10^{-3}$. The second nullcline, with respect to variable y, is trivial $x \equiv 0$ (given by $\dot{y} = 0$). In Fig. 2.3.1 (c), there are depicted breathers and chimeras over single period intervals in the $(x, y)-$plane, for exactly the same parameter conditions, but with different initial conditions.

In the parametric regions of chimera existence, the system described by Eq. (2.2.9) is in principle able to exhibit two different behaviors: chimeras and breathers. It can be seen from Fig. 2.3.1 (c) that both breather (blue line) and chimera (green line) follow the first nullcline. However, in the case of chimera, the switching between the nullcline branches occurs much more rapidly than for the case of breather dynamics. The breather's motion, on the other hand, is a more complicated version of the slow-fast motion, e.g. in FitzHugh-Nagumo neuronal model [87, 88]. The fact of different solutions coexistence opens new points of discussion: (1) regions of chimera existence depending on parameters ε and δ, (2) coexistence of different chimera solutions, and (3) probability of those solutions occurrence in an infinite dimensional phase space.

2.3.2 Coexistence of multiheaded chimera solutions in $(\varepsilon, \delta)-$parameter plane

The behavior of the bandpass Ikeda system is governed by the two-dimensional DDE Eq. (2.2.9) which is highly multistable:

$$\begin{aligned} \varepsilon \dot{x}(s) &= -x(s) - \delta \cdot y(s) + f\left(x(s-1)\right), \\ \dot{y}(s) &= x(s). \end{aligned} \tag{2.2.9}$$

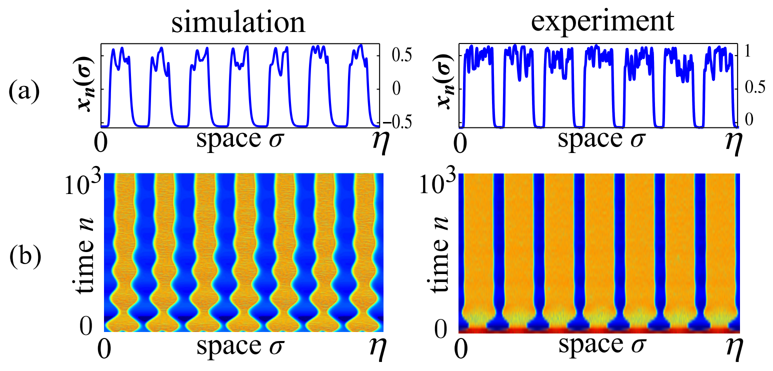

Figure 2.3.2: An example of seven-headed chimera states obtained in a simulation and experimentally. **Upper**: Time traces of the last delay show seven chaotic intervals (heads). **Lower:** Virtual space-time plots reveal stabilization of the seven-headed solution from specially modulated initial conditions $x(t) = \sin(7 \cdot 2\pi \cdot x/N_{\text{samples}}), t = [-1, 0]$. Both numerical and experimental solutions were obtained close to the parametric point A: $(\varepsilon = 0.005, \delta = 0.008)$, with $\beta = 2$.

Depending on initial conditions, there exist chimeras with different numbers N_σ of chaotic regions separated by a steady state within a single delay interval. We call those chaotic regions *heads*, and the corresponding solution *N_σ-headed chimeras* (Fig. 2.3.2). This suggests exploring their existence in an $(\varepsilon, \delta)-$parameter plane. The obtained parameter regions of multiheaded chimeras existence are depicted in Fig. 2.3.3.

Chimeras with different numbers of heads exist in the blue regions shown in Fig. 2.3.3, while chaotic breathers can be found in the whole plane, both in numerics and experiments. The main difference is that in experiment, there is a much higher probability to observe chaotic breathers than chimeras. That discrepancy might be related to presence of noise and also inaccurate values of parameters such as m and Φ_0, which are not controlled precisely with current experimental setup. In fact, m depends on the reflection coefficient of the Fabry-Pérot interferometer, and also on the accuracy with which the laser beam is collimated into the Fabry-Pérot; while the value of Φ_0 cannot be precisely recovered due to the constant system's drift.

In the first, lightest blue region to the right, only one-headed chimeras can be obtained (plus breathers). Moving to the left, in the second region, both one- and two-headed chimeras coexist, depending on the initial conditions. Moving further to the left, high-order multiheaded chimeras coexist with all previous chimera solutions. For instance, when starting with a N_σ-headed chimera and gradually decreasing ε (or increasing δ), the chimera with N_σ heads will continue to exist. However, when starting with a N_σ-headed chimera and moving to the right, i.e. increasing ε (or moving down, decreasing δ), the number of heads decreases and the solution will eventually transform into a one-headed chimera (or breather).

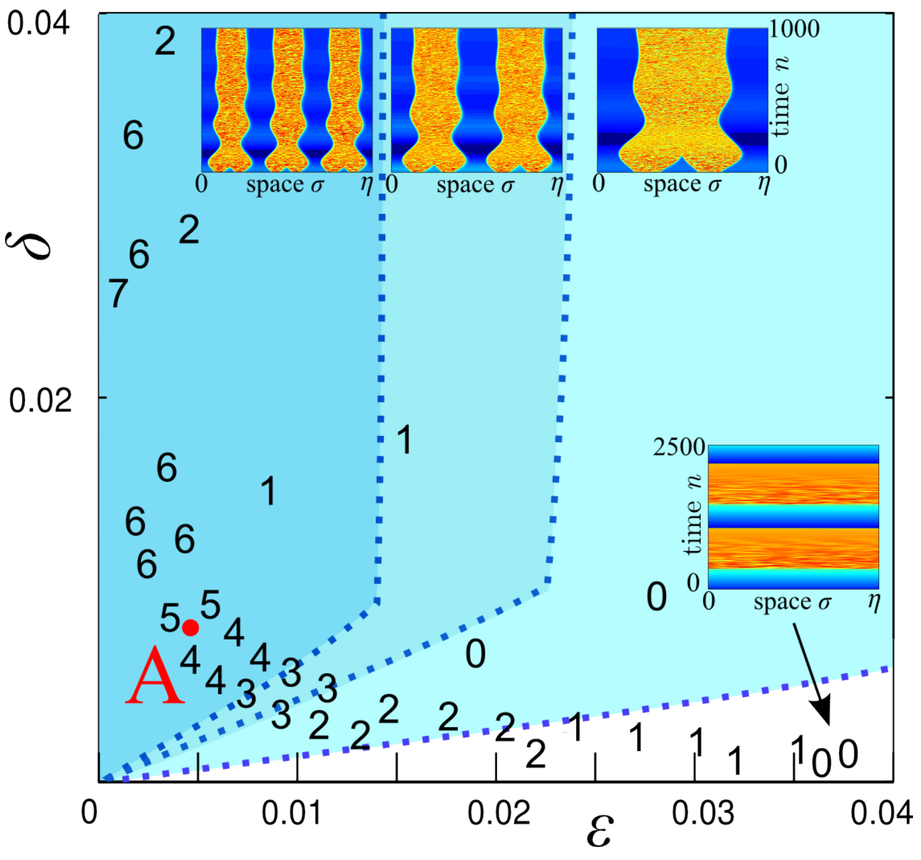

Figure 2.3.3: Existence of multiheaded chimera states in $(\varepsilon, \delta)-$parameter plane with fixed $\beta = 2$. Color regions: numerics, latin numbers: number of heads in experimentally observed chimeras. Decreasing ε, or increasing δ allows for observing chimera states with higher number of heads. Moving in opposite direction, multiheaded chimera states loose stability and are replaced by chimeras with lower number of heads. Finally, in the lower right no chimera states can be found. Examples of breathers and one-, two-, and three-headed chimeras can be seen in space-time plot insets. Parameter $\eta := 1 + \gamma \simeq 1$ is approximately a single delay interval. Red "A" marks the parametric point A: $(\varepsilon = 0.005, \delta = 0.008)$ introduced in Section 2.2.5.

Experimentally, a qualitative agreement for the described above behavior was achieved. By scanning the hyperbolas from top left to bottom right (varying the delay time τ_D, thus moving in the $(\varepsilon, \delta)-$plane along a hyperbolic curve described by the equation $\varepsilon\delta = 1$), we observed transitions from $N_\sigma + 1$ to N_σ-headed chimeras that are depicted by latin numbers in Fig. 2.3.3. The quantitative differences in the size of multiheaded chimera regions of existence between numerics and experiment might be explained as previously by noise inherent to the physical system and uncertainty of the drifting parameter Φ_0.

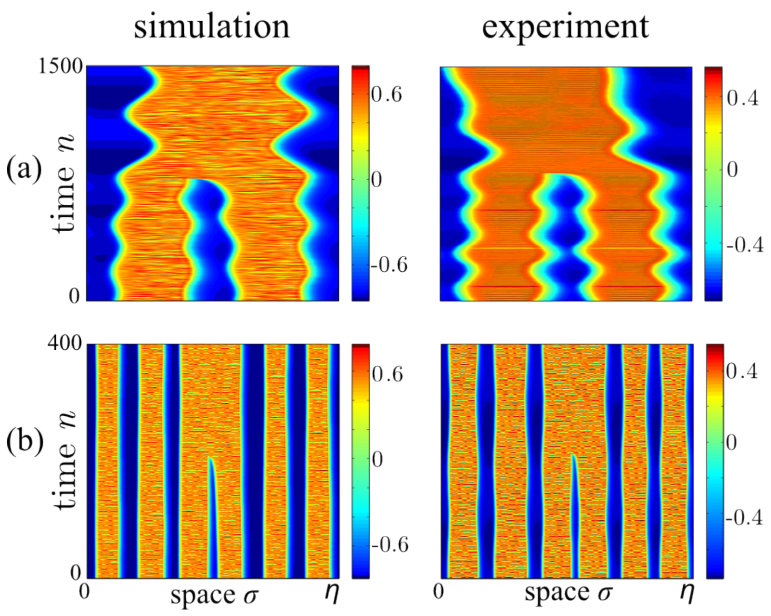

Figure 2.3.4: Transition from $N_\sigma + 1$ to N_σ-headed chimeras in simulation (**left**) and laser-based experiment (**right**) when crossing the existence boundaries: (a) Transition from 2- to 1-headed; (b) Transition from 6- to 5-headed chimeras.

Transitions from $N_\sigma + 1$ to N_σ-headed chimeras in simulations and in experiment are compared in Fig. 2.3.4. Approaching a boundary between Chimera states consisting of different numbers of heads, the chimera solution starts to oscillate (it is better seen in Fig. 2.3.4 (a)), and, after a certain transient time, a pair of heads merges, producing a $N_\sigma - 1$-headed chimera solution.

2.3.3 Chimera basins of attraction

Equation (2.2.9) describes a highly multistable DDE for small values of ε, as it can be concluded from the Fig. 2.3.3. A question which arises is the likelihood to obtain a certain N_σ-headed configuration for starting from random initial conditions. Such an investigation requires a statistical (probabilistic) characterization of the chimera's basin of attraction. By running a number of random numerical experiment realizations, with fixed system parameters, we calculate the probability of occurrence of N_σ-headed chimeras (Fig. 2.3.5).

The obtained results for 300 realizations with random initial conditions, presented in Fig. 2.3.5, reveal that when decreasing ε the probability of occurrence of high-order multiheaded chimeras increases. For instance, for $\delta = 0.02$ and the values of $0.007 < \varepsilon < 0.015$, the most probable solution is the one-headed chimera (red bars),

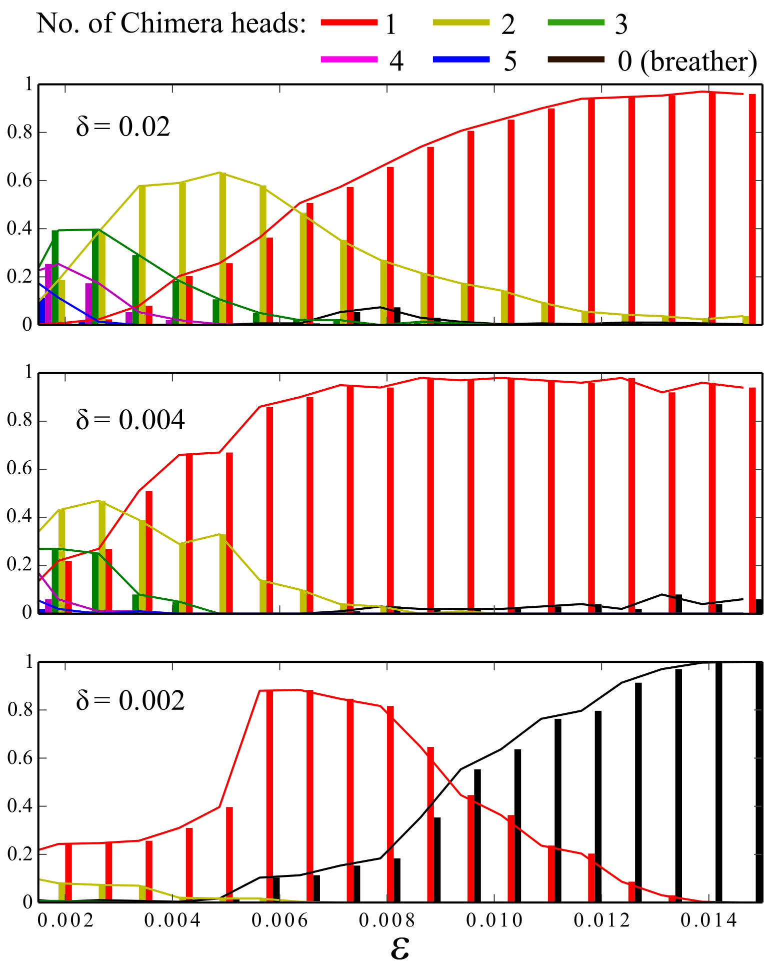

Figure 2.3.5: Probabilistic distributions reflecting basins of attraction of different chimera configurations calculated with 300 different random initial conditions for each value of ε. Three diagrams correspond to different values of δ. The values of initial interval are picked from a uniform amplitude distribution within the range of $x(t) \in [-1, 1], t \in [-1, 0]$; $\beta = 2$.

but the two-headed solution also occurs with a high probability. When decreasing ε, i.e. $0.003 < \varepsilon < 0.007$, the most probable scenario becomes a two-headed solution (yellow bars), which coexists with one-, three-, four-headed chimeras. Further decrease of ε makes three-, four-, and five-headed chimeras more probable, however one- and two-headed chimeras continue to exist, yet with decreasing probability.

When decreasing parameter δ, the probability to obtain high-order multiheaded chimeras decreases. With $\delta = 0.004$, the one-headed solution is the most probable for a wider range of values; the two-headed solutions practically never occur for $\varepsilon < 0.009$. The regions of existence of higher-order multiheaded chimeras shrink and shift to the left. With smaller $\delta = 0.002$, one doesn't observe chimeras with more than two heads. For

$\varepsilon < 0.009$ the most probable in this case is one-headed chimera, while for $\varepsilon > 0.009$, chaotic breathers occur more frequently than chimeras. That picture is comparable with the multiheaded coexistence regions depicted in Fig. 2.3.3.

2.4 Conclusion

Chimera states are described as spontaneous partial desynchronization in homogeneous networks of coupled oscillators. Through the spatiotemporal analogy, delay systems can be regarded as spatially-extended systems, i.e. networks of oscillators. Moreover, in DDEs all the virtual nodes of the network are completely identical because of time multiplexing applied to a single nonlinear node. In spite of the symmetry of DDE, we have shown that with a specially designed nonlinearity f, coherent-incoherent motions can occur. However, that motion is transient and has to be stabilized. We were able to obtain stable *chimera states* after modifying the DDE by including an integral term. The role of this term to provide a long-range coupling within the network. Experimentally, inclusion of the term corresponds to a bandpass-filtered delayed-feedback dynamics, instead of a lowpass-filtered one. Qualitatively new phenomena in bandpass-filtered DDEs have been previously reported in [84, 85].

Subsequently, our current experimental setup is exploited to study complex self-organized motions in the homogeneous ensembles of coupled oscillators. Both numerics and experiments reveal coexistence of chimeras and so-called chaotic breathers [85], depending on the initial conditions. On the other hand, depending on parameters ε and δ in the model Eq. (2.2.9), the complexity of the observed patterns is changed. This change affects the probability of a higher number of chaotic *heads*. In the same time, by increasing the control parameter β, the system becomes more chaotic. The chaotic regions become interspersed with the coherent ones after $\beta > 2.1$. Finally, the structures disappear giving place to fully developed chaos when $\beta > 13$. The observed patterns can be compared with transition to turbulence, e.g. [70].

A potential use of the bandpass delay dynamics taking advantage of high multistability are alternative memories. A so-called regenerative, or self-healing memory[8] may benefit from the chimera pattern robustness. Work in this direction was already reported in [37, 38], also utilizing dynamical systems with delay. Given the multitude of possible trajectories in the system's phase space and efficiency of experimental realization, DDEs are a candidate for the hardware implementation of *reservoir computing* concept, investigated in the following Chapters.

[8]A dynamical system-based memory that is recoverable after perturbations.

Chapter 3

Nonlinear delay systems for neuromorphic computing

The simplest theory is the best.
– Occam's razor

The aim of this Chapter is theoretical and numerical study of the recurrent neural network approach known as *reservoir computing*, for digital hardware implementation. In particular, we concentrate on the method exploiting nonlinear dynamics with delayed feedback playing the role of the *reservoir*, the recurrent layer. The principal difference between reservoir computing and other recurrent neural networks is that the nonlinear part of the network (recurrent layer) is generated separately from the linear readout. The advantages of the method are (1) rapid training procedure guaranteed to converge, that also reduces computational costs; and (2) ability to exploit nonlinear dynamical systems as a computational substrate.

3.1 Introduction to reservoir computing

Every known object in the Universe is bounded to time, that is, it cannot exist without time. Yet every such object with its time-dependent state can be regarded as a unit of information, and change of the state is information processing[1]. That naturally leads to the question, can future computers be designed in a fundamentally different way than they currently are? In order to better understand this question, the concept

[1]Computational universe theory is an assumption that quickly computable universes are more likely than others [89]

of computation needs to be formalized. We define computation as a nonlinear transformation of the input information. Following the discussion, we can state the main components needed for computation in general: (1) ability to inject the information to be processed, (2) a system performing the nonlinear transformation, and (3) ability to read the result. The process of computation is the change of the internal state of the computing system, which can be read directly or indirectly.

At this point it might be illustrating to recall that also a conventional computer is a physical system. Conventional computation is an exploitation of electrostatic laws that results in nonlinear transformations implementing Boolean logic[2]. However, in our computers there is no physical substrate performing the operations in binary domain. All signals are analog and only an arbitrary defined threshold converts these signals into discrete, binary values. In fact, conventional computers implement *a virtual machine* executing abstract logic. As such, computation is a general concept, and a computer, therefore, does not necessary have to be an electronic device, nor a machine implementing Boolean logic.

The concept of the information processing goes far beyond our current implementation. For instance, for the realization of a computing device one could alternatively exploit hydrodynamic or photonic laws. The same claim can be made regarding interactions between organisms, such as bacteria in a controlled environment. In fact, it has been shown that real life phenomena already provide computation by themselves [90, 91, 92]. The principal difference between conventional computers and many emerging alternatives is the way to exploit the underlying physics of the hardware substrate for computation [61, 91]. Finally, logic itself does not need to be constrained to binary, as it is prominently illustrated e.g. by the increasing popularity of quantum computing [93].

All those questions, of physical substrate selection, virtual machine layer versus direct access, and choice of elementary operations, lead to alternative computation paradigms. Among those paradigms, *reservoir computing* (RC) recently emerged [11, 12] as a novel computing strategy. Behind the idea of RC stands emulation of one physical phenomenon using the means of another. For instance, surface of water was shown to be capable of speech recognition, i.e. emulation of a biological system by the means of a hydrodynamical one [94].

The emulation can be advantageous in several cases. First, when the emulator is as efficient or even more efficient than the original computing mechanism. For example, a photonic systems could perform speech recognition with millions of recognized words

[2]Recall the XOR operator we defined in Section 1.2.2. Fig. 1.2.2 illustrated that this operator cannot be resolved linearly. XOR is one of the typical transformations performed by a conventional computer over pairs of bits.

per second [95]. Second, when *several* information processing devices are emulated at once[3]. Third, when the emulator is cheaper or consumes fewer resources than the original system.

3.1.1 Why "reservoir"?

The term *reservoir computing* stems from one of the earliest examples of such devices that was a proof-of-concept reservoir of liquid performing simple logical arithmetic [96] and speech recognition [94]. The idea was that the external information inputs were perturbing the surface of that liquid, while the nonlinear *transient* response in the form of ripples was the computation. For the purpose of information processing, we also utilize dynamical systems generating a nonlinear response to the external stimuli before settling back into a steady state. That nonlinear effect is the mean of computation.

General framework of RC was first developed in early 2000-s independently: bearing a name of LSMs (liquid state machines) by the group of Maass [11], the name of ESNs (echo state networks) by the team of Jaeger [12], and of backpropagation-decorrelation (BPDC) by Steil [13]. The name reservoir computing serves as an umbrella term connecting those approaches by illustrating the idea where the "reservoir" part of the processing system is preexisting (even randomly generated) and remains unchanged, and only the readout component is trained.

RC is frequently referred to as a brain-inspired approach. The first analogy is that human being does not consciously "design" the mechanics of their brain. On the contrary, they inherit it, the brain comes as a *preexisting* organ. That organ serves as a general-purpose "device" allowing to quickly adapt to new and unexpected situations. One of the possible explanations how such adaptability is possible could be the brain is a complex network. If it is very complex, some of the solution trajectories must *already* exist [3] in the high dimensional dynamics of such a system. The remaining procedure, usually called learning, is rewiring the brain connections so that useful trajectories are amplified and not useful ones are suppressed[4].

Another notable analogy between RC and the brain is the memory hierarchy. Unlike conventional computers, the human brain has memory inseparable from computing units (neurons) [2]. RC tries to alleviate this discrepancy by including memory in the computation procedure.

[3]That is achievable for the class of *task-independent* reservoir computers we consider in this work. See also [11, 34] for the examples of parallel information processing.

[4]That hypothesis can also explain the forgotten human memories as the "less useful" trajectories in the brain dynamics.

3.1.2 Architecture

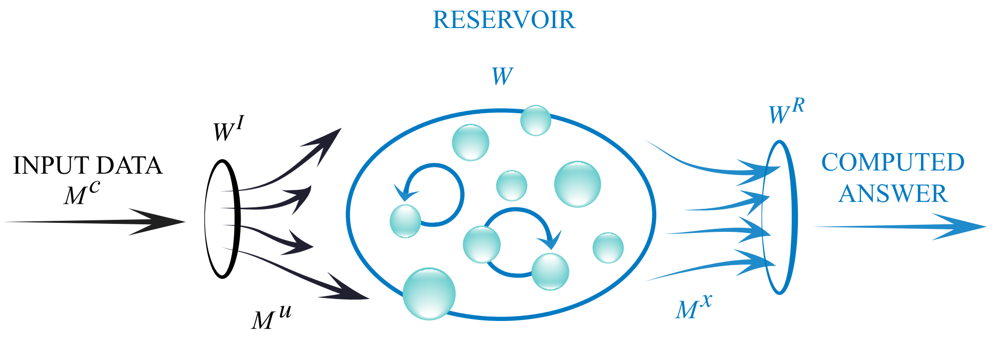

Figure 3.1.1: Typical architecture of a reservoir computer: data are masked with a randomly generated mask W^I, then nonlinearly transformed in a recurrent layer called *reservoir*, and finally, linearly recombined by multiplying a readout matrix W^R. Only some of the reservoir's internal connections are shown for simplicity.

A reservoir computer consists of the three principal functional blocks (Fig. 3.1.1):

1. **The input data masking layer**, which randomly maps the inputs to the internal nodes of the reservoir. In general, a linear data masking is performed.
2. **The recurrent layer** (*reservoir*), which performs a nonlinear transformation of the masked data.
3. **The readout layer**, or readout map, which, after training, linearly recombines the internal states of the reservoir.

The first two blocks (the masking block and the reservoir) comprise the **preexisting part**. They are typically generated with the use of coefficients drawn from a random distribution with zero mean. The preexisting part can be described as a discrete evolution equation

$$\mathbf{x}(n) = f\left(W^I\mathbf{u}(n) + W\mathbf{x}(n-1)\right), \;\; n = 1, \ldots, T. \tag{3.1.1}$$

Here W^I matrix is the input mask and vector $\mathbf{u}(n)$ is the input data vector. Vector $\mathbf{x}(n)$ is a network of "neurons". The network's recurrent connections are determined by the matrix W. Coefficients in W^I and W are generated randomly. Finally, f is a nonlinear transformation, e.g. sigmoid, tanh, sinusoid, etc. The internal state of the reservoir $\boldsymbol{x}(n)$ described by Eq. (3.1.1) is evolving in discrete time n.

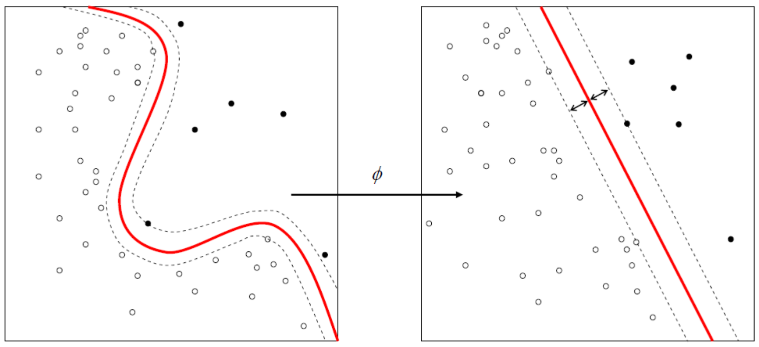

Figure 3.1.2: Kernel trick: two linearly non-separable data sets (left) become linearly separable in a high-dimensional space (right) after mapping with a similarity function (kernel) ϕ. Image source [97].

Nonlinear transformation of the masked inputs via the reservoir's dynamics in Eq. (3.1.1) allows projecting the information input into a much higher dimensional space[5]. The idea of transition to a higher dimensional space is similar to the machine learning technique called *kernel trick* [98]. Fig.3.1.2 illustrates that with appropriate mapping ϕ, nonlinearly separable problems become linearly separable in a higher dimensional *feature space*[6]. Similarly to kernel tricks, the preexisting RC part (masking operation and reservoir transformation) provides a random (implicit) mapping ϕ. The mapping allows operating in a high-dimensional feature space[7].

The computation is finalized in the last block, called a **readout layer**. This layer is typically a linear recombination of the outputs coming from the reservoir. The computation result, vector $\mathbf{y}(n)$ is expressed as:

$$\mathbf{y}(n) = W^R \cdot \mathbf{x}(n), \tag{3.1.2}$$

where W^R is the readout matrix.

Equation (3.1.1) is equivalent to Eq. (1.2.10), which makes clear that reservoir computer is an RNN. The difference between RC and RNN is only in training. Unlike

[5]To better understand why operating in higher dimensional space is useful in reservoir computing and other machine learning techniques, recall, for instance, the XOR operator from Section 1.2.2. A single-layer perceptron was not able to approximate the XOR function. However, after the initial 2D inputs were mapped into a 3D space by providing an additional layer of perceptrons, the problem was solved.

[6]In machine learning, feature space is a vector space of all possible input information vectors. The latter vectors are also called *feature vectors*.

[7]Even though the random mapping *a priory* will not be optimal, in practice it is usually sufficient to make problems linearly solvable [99].

RNNs, in RC only the linear readout map W^R is trained, while the other matrices W^I and W are generated randomly. That not only solves the RNN training convergence problem, highlighted in Chapter 1, but also makes training procedure much more efficient. The RC approach drastically reduces time and computational power in comparison to the full RNN training methods, such as BPTT.

3.1.3 Learning procedure

RC is a supervised learning approach (see Section 1.2.4.1). The learning procedure we perform in this work is so-called offline learning[8] where all the training data are available ahead.

To train the readout layer, the internal states of the reservoir $\mathbf{x}(n)$ have to be recorded. A two-dimensional matrix M^x consists of vectors $\mathbf{x}(n)$ corresponding to reservoir's response to each of inputs $\mathbf{u}(n)$. The goal of learning procedure is to obtain a matrix W^R, such that $W^R = \operatorname{argmin} \left\| W^R \cdot A - \mathrm{B} \right\|$ where A is the reservoir's response and B is the *teaching matrix* containing the desired outputs. That can be achieved by linear regression. Often, a less computationally expensive method called a *ridge regression* is employed instead:

$$W^R = (A \cdot A^\intercal + \mu \cdot I)^{-1}(A \cdot B^\intercal) \tag{3.1.3}$$

where $\mu = 10^{-4}$ is regularization parameter. Matrix A is the reservoir's nonlinear response constructed from the internal $\mathbf{x}(n)$, B is a *desired outcome*, or *teaching* matrix, and $(\cdot)^\intercal$ is the matrix transposition operator.

After the readout matrix W^R is obtained, an RC system can be actually exploited. First, a new, unseen input M^u has to be masked and processed by a reservoir, resulting in a reservoir's response matrix M^x. The result of computation M^y is given by a linear weighting of M^x:

$$M^y = W^R \cdot M^x. \tag{3.1.4}$$

Here, M^y may or may not be post-processed depending on the task. In an example of a RC-based classification provided by Fig. 3.1.3, the most "active" row in M^y is the one corresponding to the first class. Post-processing by row-wise accumulation over time allows for discrimination of the answer candidates in classification problems. This post-processing technique is a so-called *winner-take-all* NN evaluation method.

[8]The alternative method is online training, performed while new training samples are coming. The alternative approach is beneficial when the system needs to adapt to the changes in the environment. For example, a wireless channel equalization is a typical task that requires online training [100].

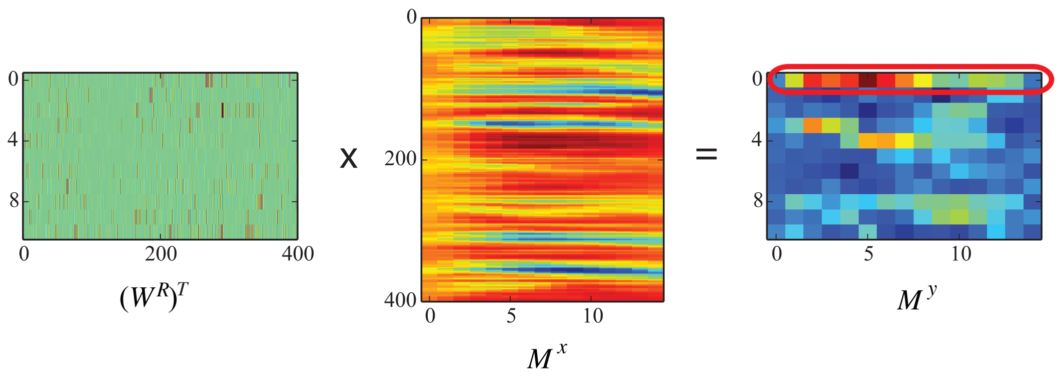

Figure 3.1.3: Exploitation of the readout matrix W^R. Injected into RC system, previously unseen input M^u produces a response M^x, which is subsequently multiplying a readout matrix $(W^R)^T$. The matrix M^y is the RC output; the answer, resulting class is encircled in red.

3.1.4 Computation with dynamical systems

The RC network defined by Eq. (3.1.1) is a recurrent neural network (RNN). However, the RC approach can be generalized to any complex dynamical system conforming to two properties:

1. *Approximation property*
2. *Separation property*

Approximation property requires that similar inputs $\mathbf{u}(n)$ produce similar outputs $\mathbf{y}(n)$. This property is required for RC systems to be robust against minor signal changes such as noise. For instance, without this property, a voice-recognition RC could not be robust against different speaker accents.

Separation property requires different inputs to be distinguishable. A system has the separation property if the different inputs $\mathbf{u}(n)$ lead to the different internal states $\mathbf{x}(n)$. Reading those internal states should help to differentiate the inputs. Note that the differences in the internal state should be much higher then the system's input-output noise.

Any nonlinear system that obeys the mentioned axioms can be considered a reservoir[9]. Generalization to the dynamical systems satisfying the above properties allows to exploit physically existent dynamical systems, such as a bucket of water [94] or a memristive synapses network [101]. Exploiting a dynamical system as a reservoir gives a background to perform computing *in-materio* [61, 90], i.e. utilizing a physical computation system directly instead of building a virtual machine on top of it. Now

[9]For a more rigorous definition of those properties, see Appendix A in [11].

that RC criteria for arbitrary dynamical systems are formalized, we shall examine if delayed-feedback systems are feasible candidates.

3.2 Single-node RC approach

One of our primary goals is building systems which support RC on the fundamental hardware level. That would allow not only achieving higher processing speeds and task parallelism but also reducing the power consumption. For such complex tasks as speech recognition RC networks with a high number of neurons are required. From a practical viewpoint, it is not always convenient to build a real network because of several reasons. The first one is the fabrication cost in terms of resources. For example, fabrication of a network of thousand Mach-Zehnder modulators[10] might substantially raise the total cost. Another reason is practical infeasibility of completely identical units, i.e. a homogeneous network, fabrication. The third reason might be the difficulty to interconnect and control a large network.

To ameliorate the difficulties in RC hardware implementation, a new approach called a *single-node*, or a delay-based RC was introduced [14]. The name reflects the fact that there exists only one physical node, instead of many. The remaining nodes of the reservoir network are emulated via the *time multiplexing* technique, introduced in Section 1.1.4.

3.2.1 DDE as a reservoir

The single-node RC adaptation is schematically illustrated in Fig. 3.2.1. There, the reservoir is a DDE having its internal state stored within a delay line. Such DDE is described by the already familiar from Section 1.1.5 driven system equation:

$$\tau \dot{x}(t) + x(t) = f\left(x(t - \tau_D) + \rho u(t)\right), \tag{3.2.1}$$

where $u(t)$ is the processed information signal and ρ is the amplification factor. As before, τ is the system's response time, τ_D is delay time, and f is a nonlinear transformation. All the emulated nodes of the network span over a delay interval (a virtual space). The network evolves in discrete time n. The mapping between a DDE and a virtual network is: $x(t) = x_\sigma(n)$, $t = n\tau_D + \sigma$, $\sigma \in [0, \tau_D]$. Here, virtual space $x_\sigma(n)$ is comparable to the internal state vector $\boldsymbol{x}(n)$ from Eq. (3.1.1). However, in the case of

[10]Mach-Zehnder modulators may be used to provide a sinusoidal nonlinear transformation, by modulating the amplitude of light beam.

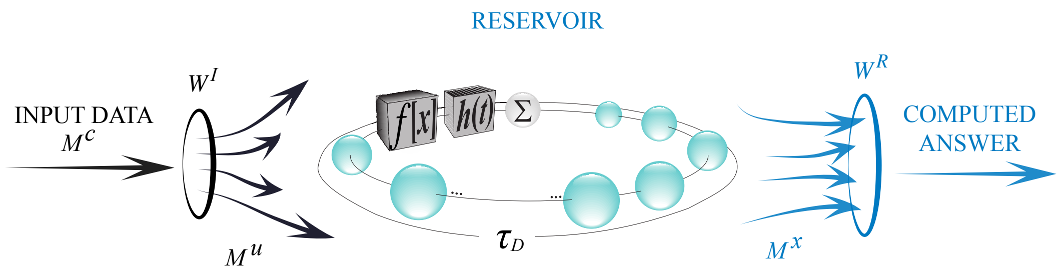

Figure 3.2.1: Single-node RC is based on a network analogy to nonlinear delayed-feedback systems. The input data is randomly projected in the dynamical system's state, which is stored in the delay. The nonlinearly transformed state is subsequently evaluated during the readout operation.

a continuous time system, there is theoretically an infinite number of values on a delay interval. In practice, though, the maximal number of distinguishable values $x_\sigma(n)$ can store is inversely proportional to the response time τ of a dynamical system. Therefore, the interval $x_\sigma(n)$ contains a finite number of nodes, and thus it is equivalent to a finite vector of values $\mathbf{x}(n)$ in Eq. (3.1.1). A prominent example of single-node approach is the utilization of a single Mach-Zehnder modulator in the context of RC, instead of coupling hundreds of them [33, 68].

3.2.2 State of the art in delay-based RC

The first time, a single-node (delay-based) RC demonstrator was attempted at an optoelectronic analog system [68]. This approach was much improved in [14], where an analog[11] electronic RC was implemented using a Mackey-Glass oscillator. Later, this work was logically continued in [102], where a mixed analog-digital system was employed. The first successful optoelectronic implementation was published in [33]. The possibility of *photonic,* Mach-Zehnder modulator-based RC implementation with offline data injection was demonstrated. This work was further developed in [103] where the effects of multiple time-delay feedbacks were studied. Another optoelectronic setup was reported in [31] and [32] where reservoir dynamics was implemented in optics.

Illustrating that RC can be universal, matrix operations were explored within the RC hardware framework in [104]. Parallel information processing within an all-optical system with a delay was demonstrated in [34]. The role of noise in RC optoelectronic systems and its tackling with input masks modification was discussed in [105]. A

[11]However, with time-discrete data input

recently reported RC setup [106] based on a coherently driven passive cavity combined with a simple interconnection input mask[12] allowed for low noise levels in experiment while maintaining as good or even improving performance in previous reports.

3.2.3 Training methods

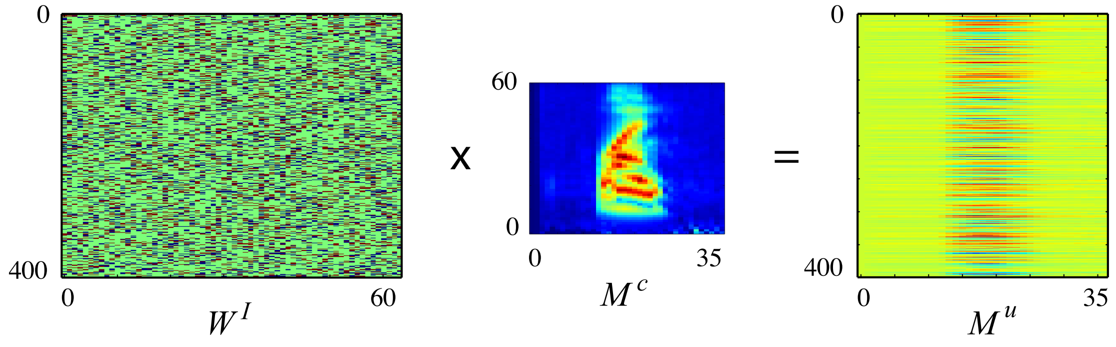

Figure 3.2.2: The masking step: a randomly generated mask W^I is applied to the input signal M^c to generate the input data M^u.

Below, there are described methods we apply to train single-node RC systems in the rest of this work.

During the first step of computation, the input data matrix $M^c \in \mathbb{R}^{M \times T}$ is masked. That is achieved by multiplication of a connectivity mask $W^I \in \mathbb{R}^{N \times M}$, which was randomly generated:

$$W^I \cdot M^c = M^u. \tag{3.2.2}$$

The role of the masking operation is linear recombination or *weighting* of the input features. The mask also expands the input information (low dimension N) into the high dimensional phase space of the reservoir (dimension $M > N$). In case of large number of input dimensions, an input mask W^I is often sparse. Then, the role of masking operation is also to keep the reservoir dynamics close to the operating fixed point. It is worth mentioning that, once generated, the input mask W^I has to be kept unchanged for each particular task. It is essential that the weighting operation (masking) is equally applied to the features of all the processed samples. However, different tasks, even with the same input dimensionality, may require different input masks. In fact, the choice of W^I may substantially affect the performance of the

[12]This type of mask was introduced in a theoretical work on RC [107].

RC system. Unfortunately, there is no known way how to randomly generate W^I with optimal RC characteristics for a particular problem. Multiple matrix generation parameters, such as value range, value offset, sparsity, etc. are mostly selected as a result of trial and error process. In Perspectives, on page 115, we propose a method how to circumvent this limitation.

During the second step, the resulting matrix M^u is converted into time series $u(t)$. The time series $u(t)$ are then injected into the reservoir described by Eq. (3.2.1). Note that the elements of matrix M^u are in discrete domain, while the dynamical system (Eq. 3.2.1) operates in continuous time. First, the two-dimensional matrix M^c has to be encoded as a one-dimensional vector $\mathbf{u}[n]$. That is achieved by an operation known as column-wise matrix *flattening*[13]. Next, in order to achieve a transition to the continuous domain, a so-called *sample and hold* method is applied: $u(t) \equiv u(nT_{\text{samp}}) = \boldsymbol{u}[n]$ where T_{samp} is a sampling period and $\boldsymbol{u}$ is a vector column in the input matrix $M^u \in \mathbb{R}^{N\times T}$. In such a way, the information from M^u can be injected into the reservoir's delay dynamics.

The third step is the dynamics simulation, where the DDE's trajectory is perturbed with external forcing $u(t)$. For numerical simulation, a second-order fixed-step integration scheme known as improved Euler's method is applied.

In the fourth step, the reservoir's response $x(t)$ is processed. A reverse operation to flattening, conveniently named *unflattening*, is applied. By unflattening, a two-dimensional matrix $M^x \in \mathbb{R}^{N\times T}$ is constructed column-wise from the sequence $x(t)$.

During the last step, learning process, the readout matrix W^R is obtained using Eq. (3.1.3). Subsequently, the obtained matrix W^R is exploited for RC as illustrated in Fig. 3.1.3, Eq. (3.1.4).

3.2.4 Model effectiveness measurement

A trained system needs to be validated in order to understand how good does a reservoir computer generalize from the training information. The validation is performed using data unseen during the training phase. There are several popular evaluation methods employed, which also depend on the type of the task being solved. Here, we use widely-known measures for prediction and for classification tasks.

In a case of **prediction tasks**, the time series signal is divided into two parts. The first part of the signal is used for training. Then the second part, a target signal

[13]The internal state of RNN (Eq. (1.2.10)) $\mathbf{x}(n)$ is the state of the network. Therefore in a single-node reservoir (DDE) it is the state of the delay line. Flattening column-wise allows to project each masked point (column in M^u) into a separate state of the delay line.

y_{target} is used for model validation by comparing with the actual RC prediction result y. As an error measure (or an objective function) a normalized root mean square error (NRMSE) is employed.

$$\text{NRMSE} = \sqrt{\frac{1}{N_s \sigma^2_{\text{target}}} \sum_{n=0}^{N_s-1} \left(y_{\text{target}}(n) - y(n)\right)^2}, \tag{3.2.3}$$

where N_s is the number of signal samples. The error is normalized by the variance of the target signal σ^2_{target}.

In the case of a **classification task**, an error measure called error rate (ER) is engaged to estimate how good is the model. It is calculated as a fraction of failed cases over the total number of test cases.

$$\text{ER} = \frac{\text{failed tests}}{\text{total tests}} \cdot 100\% \tag{3.2.4}$$

In addition, for classification problems with a small data set a method called *k-fold cross-validation* is often applied. The idea behind the method is to randomly separate the training and previously unseen testing subsets in several rounds. The initial dataset[14] $\mathbf{D}$ containing all the input data samples $\mathbf{D} := \bigcup\{M^c\}$ is divided by k non-overlapping subsets D_i^t with an equal[15] number of elements in each: $\bigcup_{i=1..k} D_i^t = \mathbf{D}, D_i^t \cap D_j^t = \emptyset \, \forall i \neq j$. Then, for each of those subsets D_i^t, also known as *test* datasets, a complementary *training* dataset $D^r_{i=1..k} : D_i^r = \mathbf{D} - D_i^t$ is obtained. For each input (training) dataset D_i^r an intermediary readout matrix W_i^R is calculated. Then, ER is computed using, an unseen during the training phase, test dataset D_i^t. The result of cross-validation is the average over ERs after all k folds (rounds).

3.3 Towards digital RC

This Section is based on methods described in Section 3.2.3. Here we propose improvements aimed at RC implementation in digital hardware. We also uncover which techniques can be used in order to improve the accuracy and speed of a delay-based RC along with the specific hardware-oriented comments.

[14]Here $\mathbf{D}$, D_i^t, etc. are sets. $\mathbf{D} - D_i^t$ is a set difference operator, $\cap$ and $\cup$ are set intersection and set union correspondingly.

[15]When the total number of elements $|\mathbf{D}|$ is not divisible by k there might be some subsets that contain one element less. It does not effect the procedure, however.

3.3.1 DDE solver

In the perspective of numerical simulation of the solution of a differential equation, one expects to achieve outputs as close as possible both on the computer and on the actual digital hardware. In both cases, the Heun's (improved Euler's) discretization method is employed to simulate the dynamics. Heun's method is a second-order ODE solver known as an explicit trapezoidal rule. The method solving a system of equations $F\left[x_n, x_{n-\text{delay}}, u_n\right]$ with a single delay $x_{n-\text{delay}}$ is given by equations:

$$\begin{aligned} \widetilde{x}_{n+1} &= x_n + \Delta t \cdot F\left[x_n, x_{n-\text{delay}}, u_n\right], \\ x_{n+1} &= x_n + \tfrac{\Delta t}{2} \cdot \left(F\left[x_n, x_{n-\text{delay}}, u_n\right] + F\left[\widetilde{x}_{n+1}, x_{n-\text{delay}+1}, u_{n+1}\right]\right), \end{aligned} \tag{3.3.1}$$

where Δt is the integration time step, $F\left[x_n, x_{n-\text{delay}}, u_n\right] = \varepsilon^{-1}\left(f[x_{n-\text{delay}} + \rho u_n] - x_n\right)$ is derived from Eq. (3.2.1), u_n is the information signal, and $f(x)$ is a nonlinear function (e.g. $f(x) = \sin^2(x+\Phi_0)$). Index n indicates that the operated variables are discretized in time, i.e. $x_n = x(t_0 + n\Delta t)$. The method given by Eq. (3.3.1) can be extended to any number of variables and/or delays in a straightforward way.

The advantage over higher order methods is hardware implementation efficiency. Unlike higher-order Runge-Kutta methods, there is no need of interpolation, which arises in DDE solving. On the other hand, Heun's method evaluates the derivative in two points, thus, improving accuracy comparing to the first-order Euler method. Therefore, the method we use is a compromise between higher integration accuracy and efficient hardware implementation.

3.3.2 Discrete time dynamics

Section 3.3.1 illustrated the fact that implementation of the delay-based RC on a digital electronic device is a discrete approximation of a continuous-time delay dynamics. In that sense, simulation of the Ikeda dynamics is already discrete in time, through the introduction of non-zero integration time step Δt, such that $dx/dt \simeq (x_n - x_{n-1})/\Delta t$. In the latter case, the goal of simulation is to get an approximation of a physical system through the use of an integration time step Δt small enough compared to the fastest characteristic time ruling the physical continuous time motion, i.e. $\Delta t \ll \tau$, where τ is the fast response time in Eq. (3.2.1):

$$\tau \dot{x}(t) + x(t) = f\left(x(t-\tau_D) + \rho u(t)\right), \tag{3.2.1}$$

This, for instance, can be achieved when the DDE is calculated each clock cycle corresponding to a time step Δt, while the dynamics of the fastest system's timescale changes not instantaneously, but during P time steps. Then, in the context of RC, each processed data point $u(t)$ has to be held for P samples (see Fig. 3.3.1, left) so that the system can respond to the perturbation. We call this information injection into continuous-time systems with non-instantaneous response a *P-oversampling*. In other case, when $u(t)$ is held for much less than P time steps, P-oversampling automatically plays a role of noise removal, i.e. the DDE is less sensitive to short perturbations $u(t)$.

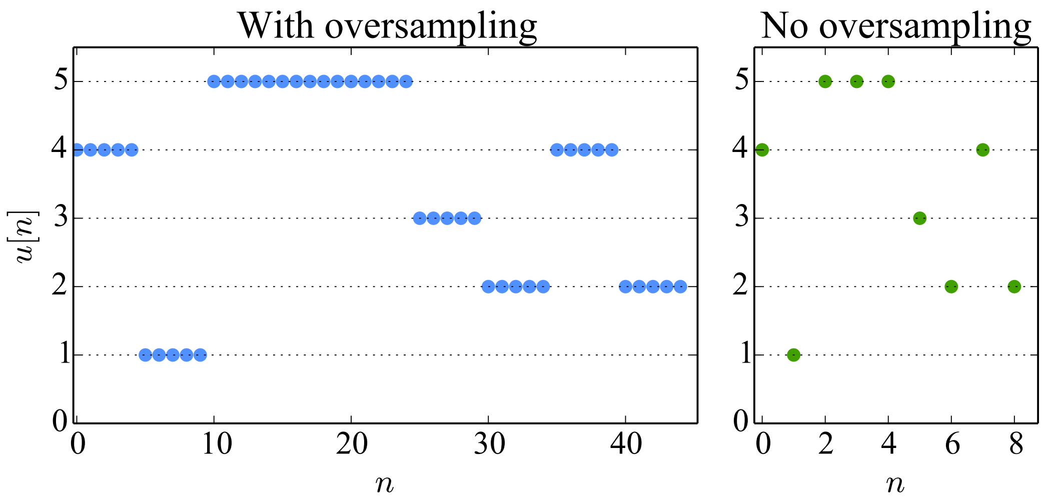

Figure 3.3.1: Discrete inputs encoded as constant levels with oversampling $\times 5$ times (left) and equivalent digital inputs with no oversampling, or one input per clock cycle (right).

However, for practical goals of information processing (RC), the accuracy of the physical system approximation may not be critical. On the contrary, each data point of the input data signal $u(t)$ has to be processed as fast as possible. That is maximized when each data point is processed during the minimal processing time, i.e. a clock cycle of a digital system. In that sense, stepping away from the physical continuous time approximation requirement $\Delta t \ll \tau$ is a possible compromise. Thus, Δt can be increased as long as the RC produces satisfactory results. That means, Δt can be comparable to τ and oversampling can be avoided (Fig. 3.3.1, right).

To investigate this issue and to understand how far the dynamics is approximated, we will make use of convolution product representation of DDEs introduced in Chapter 1. In discrete time case, the filter can be defined by a impulse response function $h[n]$. The input-output relation reads as a discrete time convolution product described by Eq. (1.1.26). The system has an instantaneous response if it responds to an input impulse held only during one time step (clock cycle).

In Fig. 3.3.2, there is a comparison of oversampled dynamics and the one with no oversampling. It can be seen that even though the dynamical trajectories are not identical, the version without oversampling still produces a comparable result. In the case of no oversampling (black dots), each point directly corresponds to a virtual node of the reservoir; whereas in the case of oversampling (blue crosshairs), an additional step (e.g. interpolation) is required in order to map the virtual nodes. All RC implementations in this work, both PC and FPGA, exploit the discretized version of the Ikeda DDE where Δt is smaller or comparable to the fastest characteristic time τ.

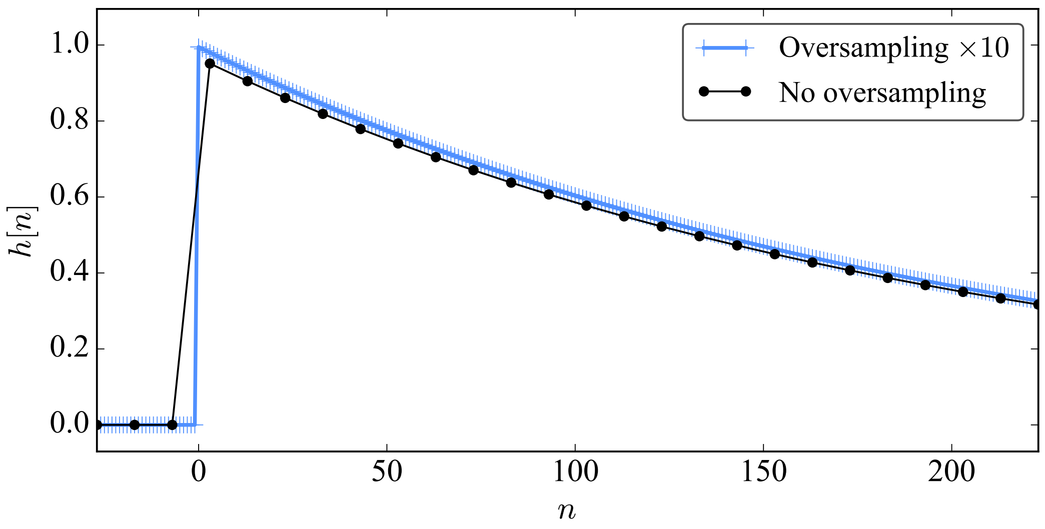

Figure 3.3.2: The system's response function $h[n]$ of case with $\times 10$ oversampling (light blue) and corresponding approximation with no oversampling (black). The number of samples on the horizontal axes corresponds to the oversampled case.

3.3.3 Efficient nonlinear transformation f

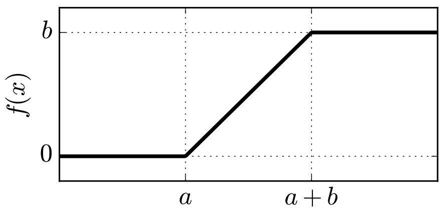

Figure 3.3.3: Hard sigmoid nonlinear function $f(x) = \max(0, \min(b, x - a))$

From the point of view of digital electronics, implementation of an arbitrary nonlinear function f is typically in the form of a lookup map, i.e. memory containing the approximate values of f. The lookup map along with auxiliary modules may consume significant device resources, especially in terms of memory. That is why we prefer, where possible, to replace f with such a nonlinear function that would be inherently supported by the digital hardware, and thus being resource efficient. A good candidate for f therefore is a *hard sigmoid* (Fig. 3.3.3), a piecewise approximation of a sigmoid function:

$$f(x) = \max(0, \min(b, x - a)), \tag{3.3.2}$$

or alternatively described via:

$$f(x) = \begin{cases} 0 & x < a \\ x - a & a \leq x < b + a \\ b & b + a \leq x \end{cases} \tag{3.3.3}$$

where a and b are constants. From the digital electronics point of view, the hard sigmoid would require as much as two comparator circuits and no additional memory.

Another, yet even simpler nonlinear function that can be utilized in the DDE is a so-called *rectifier* function:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{3.3.4}$$

which can be also expressed as $f(x) = \max(0, x)$. Rectifier function was derived *automatically* from Eq. (3.3.3) by evolutionary algorithm (described in Section 3.3.5). An electronic circuit implementing the function f comprises only a comparator. That makes f described by equations (3.3.3) and (3.3.4) extremely efficient.

So-called rectified linear units (ReLU), functions of type $f(x) = \max(0, x)$ are trending in machine learning community [67]. The advantage of ReLUs over tanh or sigmoid is faster learning of multi-layer FNNs, avoiding need of unsupervised pre-training [67]. We anticipate positive impact of ReLUs might have on RNNs, which are equivalent to FNNs with a large number of hidden layers (explained in Section 1.2.4.3).

3.3.4 Fixed-point arithmetic

The most crucial remaining difference between common digital hardware and a CPU is the usage of floating point operations. Their support may be not practical to implement in hardware. The main reasons for that are speed and area.

Speed. Floating point operations require up to four clock cycles on modern processors such as Intel.

Area. Digital electronic devices contain a limited number of logic gates that can work in parallel. Floating point standards require not only 32 or 64 bits, but also "housekeeping" infrastructure.

Digital devices usually provide *fixed-point* arithmetic. The name reflects the fact that the number of fractional bits remains constant during operations. Fixed point arithmetic is done within a single clock cycle. On the other hand, the floating point arithmetic is superior: it gives much better precision and usually avoids the unexpected overflows. However, many devices have built-in support only for integer arithmetic. The integer arithmetic can be equivalently interpreted as fixed-point. Therefore, fixed-point arithmetic consuming less area is inherently supported. Subsequently, the limitations of this arithmetic have to be considered during theoretical RC system investigations.

3.3.5 Multiparametric optimization

Different tasks may require different parameter sets in order to achieve the best performance. Parameters from Ikeda-type models (Eq. (3.2.1)), such as $\beta, \varepsilon, \delta, \Phi_0, \rho$ and others result in a high dimensional parameter space. Those parameters need to be optimized simultaneously every time to adopt the *reservoir* dynamics to a new problem. Methods that are usually applied include Monte-Carlo simulations and grid search. However, both of them have a drawback: a rapidly growing number of RC simulations to be done when the number of parameters increases. Each simulation corresponds to a complete reservoir training with a new set of parameters, therefore, the parameter optimization procedure can take a significant amount of time.

Inspired by [108], we propose a different strategy. The reservoir is still treated like a black box, but its parameter configuration is optimized via evolutionary algorithms. The strategy is the following: a conventional computer configures the RC device by adjusting the controllable parameters. The advantages of such approach: (1) the evolutionary algorithm permits to effectively cut the number of searches and (2) this method can be applied to any physical RC candidate, including, but not limited to electronic, photonic, mechanic reservoirs, etc. This idea also works when both the reservoir and the evolutionary algorithm are performed on the computer.

The type of evolutionary algorithm we chose is *genetic algorithms* (GAs), which is a biologically-inspired optimization approach. GAs imitate a natural selection process by implementing two probabilistic operators: *crossover* and *mutation.* The parameters being optimized by GA, i.e. the reservoir configuration, are encoded as *chromosomes* Q^i. The *crossover* operator is responsible for information exchange between the chromosomes. *Mutation* operator does occasional random change in a small fraction of chromosomes. Initially, a population of chromosomes $\{Q^i\}, i = 1 \ldots N^{pop}$ of size $N^{pop} = 50$ is randomly pregenerated. The generation is evaluated according to a certain rule, a *fitness* function. In the case of RC, many possible functions can be chosen for that purpose[16]. Similarly to natural selection, the fittest chromosomes have the highest probability to survive and produce offsprings. Then, the next generation of chromosomes is built according to the genetic operators. Therefore, each next generation tends to be fitter in average than the previous one. In total we run up to $N^{gen} = 30$ generations, exploring not more than $N^{pop} \cdot N^{gen} = 50 \cdot 30 = 1500$ configurations of RC. The algorithm usually converges to an optimal solution before the 20th generation, effectively cutting the time of an optimal configuration search.

Implementation of genetic operators is context-dependent, in particular it depends on the chosen chromosome encoding. Among the popular encoding schemes are strings, such as a string of numbers, and trees, such as a tree of transformation functions. To explore the multiparametric space, we use a binary representation of our parameters, i.e. chromosomes Q^i are encoded as binary strings. A chromosome $Q^i = \text{concat}(Q^i_1 Q^i_2, \ldots Q^i_K), i = 1 \ldots N^{pop}$ is a concatenation of binary encoded parameters $Q^i_k = \left\{q^i_j : q^i_j \in \{0,1\}\right\}, k = 1 \ldots K$, where K is the total number of parameters being optimized. Each parameter Q^i_k uses fixed-point notation and has a bit resolution $6 \leq M^k_{bits} \leq 10$ bits. The bit resolution is higher if the sensitivity to the parameter is considered to be higher. Binary encoding of chromosomes allows for a straightforward definition of the mutation and crossover operators. The unary mutation operator MUT, which produces 12% of the new generation, is defined as:

$$\text{MUT}\, Q^i = \left\{\hat{q}^i_j\right\}, \text{ where } \hat{q}^i_j = \begin{cases} \text{not } q^i_j & if\ p_j < p_{bit}, \\ q^i_j & otherwise, \end{cases} \quad j = 1 \ldots M_{bits}, \tag{3.3.5}$$

here $\hat{q}^i_j \in \{0,1\}$ is a bit in a mutated chromosome, p_j are randomly generated numbers within the range of $(0,1)$, the constant $p_{bit} = 0.2$ is the probability of bit mutation, and $M_{bits} = \sum_{k=1}^{K} M^k_{bits}$ is the chromosome length. The binary crossover operator $\otimes$ produces the remaining 88% of the new generation:

$$Q^a \otimes Q^b = \{\hat{q}_j\}, \text{ where } \hat{q}_j = \begin{cases} q^a_j & if\ p_j < 0.5, \\ q^b_j & otherwise, \end{cases} \quad j = 1 \ldots M_{bits} \tag{3.3.6}$$

[16]The ones we have utilized in this work have been introduced in Section 3.2.4

where $\hat{q}^i_j \in \{0,1\}$ is a bit in a child chromosome, the bit q^a_j comes from the parent chromosome Q^a, and q^b_j, from the parent chromosome Q^b.

The ratio between the entities produced by crossover and mutation operators might be different and effects mostly the speed of GA search in terms of explored configurations. In addition, we keep track of the best configuration chromosomes of all generations in case if the best configuration was found in one of the previous generations. We must admit GAs guarantee only that the found solution is *locally* optimal. Due to their probabilistic nature, there is no guarantee that GAs will find a globally optimal solution or that the same solution would be found when the search is repeated.

3.4 Performance benchmark tests

In the machine learning community, there exists a multitude of popular benchmarking tasks used for evaluation and comparison between different techniques. Among them, we have chosen tasks widely used in the RC community: Mackey-Glass chaotic time series prediction and spoken digit recognition based on TI-46 data set. Additionally, we have also implemented a benchmark test less popular for RC evaluation, though generally known in the machine learning: Aurora-2, a more complex spoken digits recognition benchmark.

3.4.1 Prediction task

Prediction, or forecasting, is a common computational problem. A classical approach solving it would be to build a mathematical model using explicit equations. For instance, given a canon shot, the task is to predict how far the canon ball will go. If all the relevant variables—initial velocity of the ball and the direction—are known, one can build a model to calculate the trajectory. However, this classical approach does not work as perfectly to predict such phenomena as weather conditions or a stock market decline. Those processes may contain hidden variables or their initial conditions may be unknown, and therefore other than classical methods are needed [109]. The machine learning approach to prediction is the “future” signal reconstruction by learning from the past time series, whereas the dynamics model is considered as a “black box”. This means, there is no information provided about factors influencing the dynamics, such as climate conditions or behavior of the market players. In this Section, we approach the forecasting problem from RC perspective.

Prediction of chaotic time series is a typical problem where only a part of the information is available. Chaotic signal properties, i.e. sensitivity to small perturbations, make the task problematic. As a chaotic system, the Mackey-Glass equation (see Section 1.1) is often used:

$$\frac{d\tilde{x}(t)}{dt} = \frac{\beta\tilde{x}(t-\tilde{\tau}_D)}{1+\tilde{x}^{10}(t-\tilde{\tau}_D)} - \gamma\tilde{x}(t), \tag{3.4.1}$$

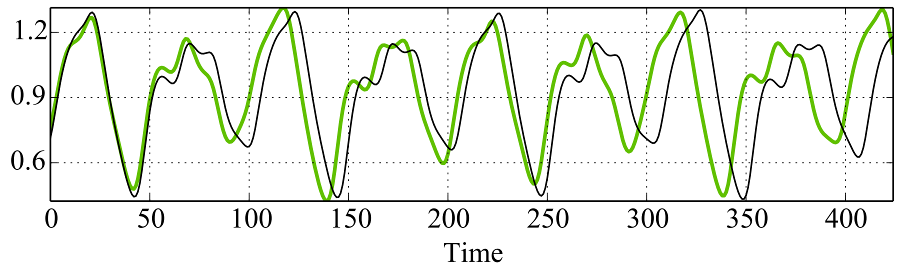

Figure 3.4.1: Example of Mackey-Glass time series where two chaotic signals diverge with time. The depicted time traces lengths are of 25 delays $\tilde{\tau}_D$. The solutions were obtained by simulating Eq. (3.4.1) with the integration step $\Delta t = 0.1$, much smaller compared to the response time $\tau = \gamma^{-1} = 10$. Then, the solutions were downsampled to 425 samples. The distance between the displayed signals in terms of NRMSE Eq. (3.2.3) is 0.87.

where $\tilde{x}(t)$ is a dynamical variable and β and γ are constants. Chaotic dynamics in DDE given by Eq. (3.4.1) can be achieved with $\beta = 0.2$, $\gamma = 0.1$, and delay $\tilde{\tau}_D = 17$ (Fig. 3.4.1). Here, the characteristic response time τ is implicitly defined through $\gamma^{-1} = 10$ time units. The longer is the delay $\tilde{\tau}_D$, the more chaotic becomes the system, i.e. Lyapunov exponents grow with increase of $\tilde{\tau}_D$. We make use of the chaotic time series generated by Eq. (3.4.1) to study the forecasting abilities of RC with a simplistic architecture (introduced in Section 3.2). Note that Eq. (3.4.1) and its parameters are hidden from RC predictor. We explore the role of different shapes of nonlinear function f, as well as multiparametric reservoir dynamics optimization in the context of prediction task.

The following methods are used to prepare the prediction data set. First, Eq. (3.4.1) is solved for $10^5 \cdot \Delta t$ integration time steps (corresponding to 588 time delays $\tilde{\tau}_D$). Before the data are processed by RC, they are downsampled[17] with a downsampling factor of 10. As a result, there are 10^4 data points obtained. The first $T^r = 5000$ data points are then used to train the reservoir, whereas the last $T^t = 5000$ points comprise the *validation* data set.

The first stage of RC is the masking procedure that randomly maps the input values into internal state space (Section 3.1.3). The step is performed as a matrix multiplication

[17]That step is supposed to remove possible redundancy in the input data.

$W^I \cdot M^c = M^u$, where $W^I \in \mathbb{R}^{N \times M}$ is the input mask and $M^c \in \mathbb{R}^{M \times T}$ is the input data. The training dataset consists of $T^r = 5000$ scalar data points, therefore $M^c \in \mathbb{R}^{1 \times 5000}$. The reservoir's size should be a good compromise between low complexity and possible *overfitting*[18], which we empirically estimate to be $N = 1000$, therefore $W^I \in \mathbb{R}^{1000 \times 1}$. The values W^I are randomly drawn from a uniform distribution in the range of $[0.1; 0.3]$.

As a consequence of the masking step, the data matrix $M^u \in \mathbb{R}^{1000 \times 5000}$ is obtained. Then, M^u is transformed by the reservoir's dynamics (Eq. (3.2.1)). To provide the input signal u, matrix M^u is flattened column-wise (Section 3.2.3). That transformation results in a data vector $\mathbf{u} \in \mathbb{R}^{5,000,000}$. Since discrete dynamics without oversampling is utilized (Section 3.3.2), the input signal is already recorded as vector $\mathbf{u}$. Technically, this means that during each discrete time step Δt, a new value $u_n \in \mathbf{u}$ is fed to DDE solver (Eq. (3.3.1)). Equation (3.2.1) is then solved for all $5 \cdot 10^6$ integration time steps, and a vector $\mathbf{x} \in \mathbb{R}^{5,000,000}$ is obtained directly as a concatenation of $x(t) \equiv x_n$. Response matrix $M^x \in \mathbb{R}^{5,000,000}$ is constructed by column-wise unflattening of vector $\mathbf{x}$.

Offline learning is then performed using ridge regression (Eq. (3.1.3)). For time series prediction, the reservoir's response matrix $A \in \mathbb{R}^{N \times T}$ consists of the first $T = T^r - H = 5000 - H$ columns of M^x, where H is the prediction horizon. Therefore, matrix A corresponds to the nonlinearly transformed (or processed by reservoir) information about the past inputs. The teaching matrix B is the Mackey-Glass time-series to be predicted. That is, the original signal M^c shifted by *prediction horizon* H time steps to the left[19]. Therefore, matrix $B \in \mathbb{R}^{1 \times 5000 - H}$ consists of the last $T^r - H = 5000 - H$ columns of M^c. As a result of ridge regression, a readout matrix W^R is obtained.

Finally, the prediction is run over unseen validation data set in order to calculate prediction NRMSE (Eq. (3.2.3)). Below, we compare prediction errors with respect to four different configurations of the reservoir dynamics. The input signal masking, training, and evaluation procedures are identical in all four cases.

First, we look at a reservoir based on an **Ikeda DDE** Eq. (3.2.1), a so-called **low-pass model**. Two cases, depending on f are considered. In the first case, the Ikeda model with a *sinusoidal nonlinearity* is used:

[18]If N is too small, then the system would not contain enough dimensions to generalize; whereas if N is too big, the system may *overfit*, i.e. learn from random noise in training data. In both cases, the so-called *validation* error (error on data unseen during training) is bigger than for optimal N.

[19]To indicate a "future" signal, since the aim is to teach RC prediction. Shifting to the left can be done only offline, i.e. when all the data is collected; it is a reverse operator to delay (looking in the past) operator, which is shifting signal to the right.

$$f(x) = \beta \cos^2 (x + \Phi_0) . \tag{3.4.2}$$

This nonlinearity can be obtained in an experiment using a Mach-Zehnder modulator, such as in [33, 103]. One therefore obtains the following DDE:

$$\varepsilon \dot{x}(t) = -x(t) + \beta \cos^2 (x(t-1) + \rho \cdot u(t) + \Phi_0) , \tag{3.4.3}$$

where, $x(t-1)$ is the delayed term, and $u(t)$ is the input signal. Parameters obtained as a result of a genetic algorithm (GA) optimization against a prediction horizon of $H = 20$ steps (corresponding to $1.2\, \tilde{\tau}_D$) are $\varepsilon = 3.13 \cdot 10^{-3}$, $\beta = 1.69$, $\Phi_0 = 3.13 \cdot 10^{-2}$, $\rho = 7.88$.

In a second experiment, we evaluate possible performance degradation replacing sinusoidal $f(x)$ with a *hard sigmoid* (Eq. (3.3.2)). We use a version of Ikeda DDE with negative nonlinear contribution of delayed feedback (suggested by GA, Section 3.3.5):

$$\varepsilon \dot{x}(t) = -x(t) - f\left(x(t-1) + \rho u(t)\right) , \tag{3.4.4}$$

where $f(x) = \beta \max(0, \min(b, x-a))$, where parameters $a = 0.44, b = 0.81$ are derived by GA. Other parameters $\varepsilon = 0.01, \beta = 1.69, \rho = 7.2$ are also derived by GA. The number of nodes is fixed $N = 1000$. Using the hard sigmoid nonlinearity, comparable results in terms of NRMSE are obtained. The two top curves in Fig. 3.4.2 summarize the results for the low-pass model.

From Fig. 3.4.2, it can be seen that the accuracy of prediction degrades as the prediction horizon expands. The increasing prediction error is a natural property of all chaotic systems, due to the intrinsic property of sensitivity to initial conditions. Therefore, any long-term forecasting of chaotic systems is impractical. Surprisingly, the prediction accuracy of the system with a hard sigmoid (piecewise sigmoid approximation) is not worse than the one of the original system with a smooth $\cos^2$ nonlinearity. That might be explained by GA that was able to find an optimal dynamical configuration (β, ε, ρ) exactly for this kind of nonlinearity. This result allows for flexible selection of a nonlinear function f depending on the target system, e.g. digital one.

Because a **bandpass model** was the one used for chimera states observation, we are curious to know if this modified Ikeda model is also able to provide any benefits for RC. The bandpass model Eq. (3.4.5) contains an integral term $\delta \cdot y(t)$, and therefore, the corresponding virtual network possesses a long-range coupling topology, which in turn may provide a more complex dynamics (see also Section 2.2.3):

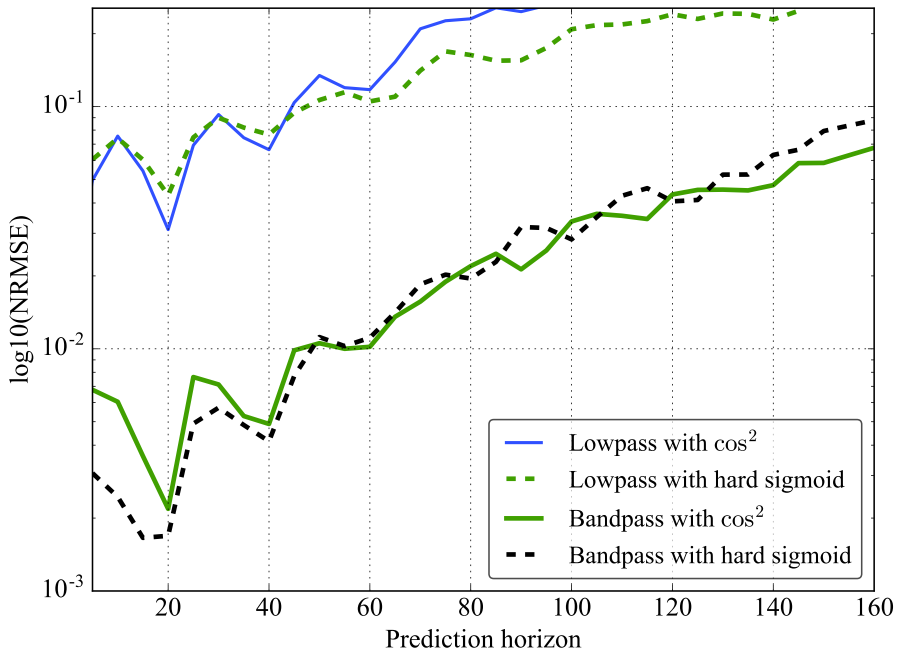

Figure 3.4.2: Mackey-Glass chaotic time series prediction. The bandpass model significantly outperforms the low-pass one for both sinusoidal and hard sigmoid nonlinearities. In the same time, the two nonlinearities f give similar results. The reservoir's dynamics parameters were automatically selected by GA in all four cases.

$$\begin{aligned} \varepsilon\dot{x}(t) &= -x(t) - \delta \cdot y(t) + f\left(x(t-1) + \rho u(t)\right), \\ \dot{y} &= x(t). \end{aligned} \tag{3.4.5}$$

where $u(t)$ is the input information signal, ρ is its amplification, $\delta > 0$, reminding that $\delta = 0$ reduces Eq. (3.4.5) to the classical Ikeda model Eq. (3.2.1). Again, we start with a sinusoidal f (Eq. (3.4.2)). Using GA, the obtained reservoir dynamics parameters are $\varepsilon = 1.56 \cdot 10^{-3}$, $\delta = 0.94$, $\beta = 2$, $\Phi_0 = -1.97$, $\rho = 7.25$. In the fourth and last experiment, with the hard sigmoid f, the respective parameters are $\varepsilon = 1.56 \cdot 10^{-3}$, $\delta = 1.266$, $\beta = 1.125$, $a = 0.25$, $b = 1.75$, $\rho = 4.75$. In comparison to the classical Ikeda dynamics, i.e. with a low-pass filter, we notice that the model Eq. (3.4.5) significantly improves prediction accuracy. The prediction error is lower both in the case of $\cos^2$ nonlinearity (Fig. 3.4.2, solid green curve) and the hard sigmoid (Fig. 3.4.2, black dashed line). The parameter sets obtained by GA produce comparable results for the both nonlinearities. Additionally, the advantage of GA over a single-parameter scan for a bandpass model can be seen in Fig. 3.4.3.

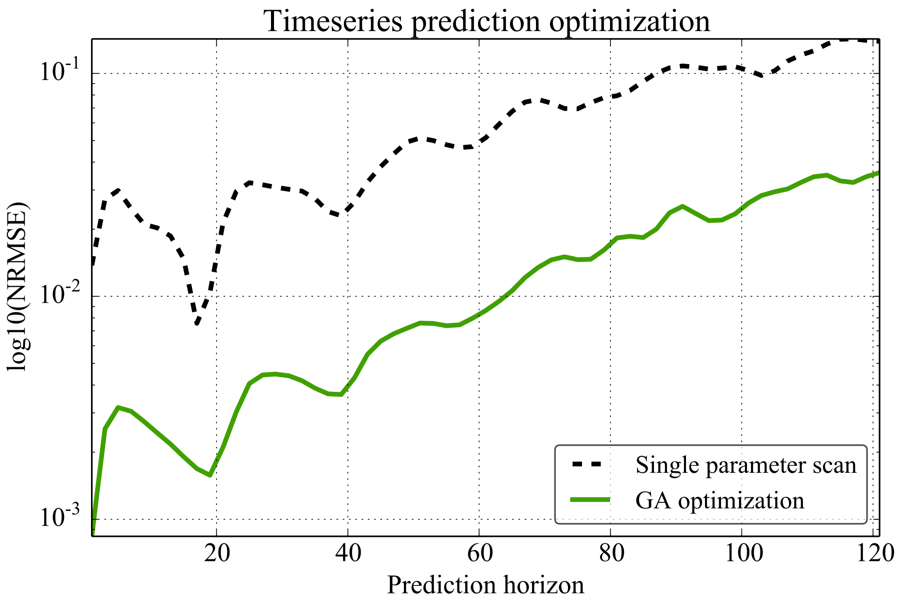

Figure 3.4.3: RC time series prediction task accuracy achieved by single-parameter optimization (dashed) and by genetic algorithm (solid line). GA optimization yields a result better by an order of magnitude. In both cases the bandpass model (Eq. (3.4.5)) is considered.

3.4.2 Classification task

Classification is a problem to identify—given a predefined set of samples—in which category belongs to a new, previously unseen example. Typical examples of classification tasks are: to determine if a product is defective or not, if a patient suffers a certain illness and which one, what kind of object is in the scene: a human, an animal or a building. The classification problems solved below utilize the same RC architecture and methods for time series prediction task with the differences only in W^I and W^R dimensions.

Two instances of classification, namely speech recognition, are covered in the Section. In both cases we investigate the isolated spoken digit recognition task with different recognition difficulty levels. The first one contains clean records of five female speakers, while the second one contains clean and noisy records of mixed genders and age, increasing the task complexity. In that case, the entire digit's sound information is required to make a decision.

As in the case of forecasting, training is performed by solving ridge regression (3.1.3). However, teaching matrix B for classification problems is different: it is a horizontal concatenation of matrices $\tilde{M}_i^y$ corresponding to each training sample. A matrix $\tilde{M}_i^y \in \mathbb{R}^{K \times T_i}$ is a *one-hot* encoded desired output (Fig. 3.4.4), i.e. the correct class at each moment of time $1..T_i$, where T_i corresponds to a training sample duration:

$$\tilde{M_i^y}(k, 1 \dots T_i) = \begin{cases} [1, 1, \dots 1] & \text{if } k \text{ is correct class,} \\ [0, 0, \dots 0] & \text{otherwise.} \end{cases}, k = 1 \dots K. \tag{3.4.6}$$

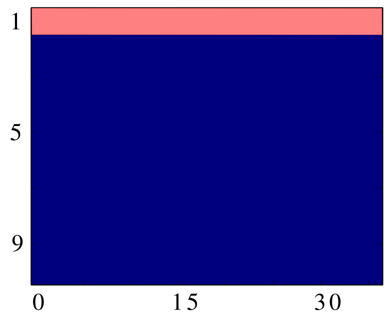

Figure 3.4.4: Matrix $\tilde{M_i^y}$ corresponding to a desired training sample, the first out of ten classes. This training sample has duration of 35 time units.

Speech recognition Numerous modern applications, starting from everyday online search and virtual assistants, ending with space applications[20], benefit from voice recognition techniques. The basic goal of voice recognition is to find in a predefined dictionary of symbols (such as words or phonemes) those corresponding to the sounds articulated by a human. Isolated spoken digit recognition task is an example voice recognition problem. In this case, the dictionary consists of ten symbols $0 \dots 9$.

A common practice in spoken digit recognition is initial preprocessing of the sound waveform. One of standard techniques is *Lyon's Passive Ear* model that mimics dynamics of the inner ear's auditory portion, the human *cochlea* [110]. Spirally shaped cochlea contains hairs sensitive only to specific frequencies. Hairs close to the organ's entrance are responsive to the higher frequencies, while the ones deeper in the cochlea, to the lower frequencies. Imitating the cochlear transformations, sound waves are converted from the time to the frequency domain within a sliding window. The transformation occurs within approximately 10 ms. A *cochleagram* is constructed by applying a set of *gammatone filters* $g(t)$ to the signal:

$$g(t) = at^{n-1}e^{-2\pi bt}cos(2\pi f_c t + \theta) \tag{3.4.7}$$

[20]See e.g. http://blog.neospeech.com/neospeech-first-texttospeech-in-space/

where f_c is central frequency, a is the gain, b is the bandwidth, $n = 4$ is the order of the filter, θ is the phase and is usually set to zero.

The resulting cochleagram M^c is a matrix with a fixed number of rows (typically 60–90 rows) corresponding to the frequency channels on an equivalent rectangular bandwidth (ERB) scale. The number of columns in M^c is variable for each individual record (between 30 and 90 columns) and reflects the duration of a spoken word. However, even after this bio-inspired preprocessing, human speech cannot be identified with linear regression. Therefore, a more sophisticated machine learning approach, such as RC, is required.

TI-46 benchmark TI-46 database is a conventional classification test benchmark used in machine learning community. It is particularly known in RC research [31, 34, 107, 111, 112, 113, etc.]. The records contain utterances by five female speakers. Each digit is pronounced ten times by every speaker, resulting in a total of 500 data records. We use TI-46 benchmark as an entry-level classification problem.

The data records are preprocessed using Lyon's model, which results in cochleagram matrices $M^c \in \mathbb{R}^{86\times T_i}$. The number of frequency channels is usually proportional to the recordings sampling rate, e.g. the records in TI-46 are sampled at 12.5 kHz. The number of cochleagram columns T_i is proportional to the record length. Being relatively simple, the benchmark does not require a very high dimensionality. Therefore, the reservoir can be restricted to a moderate number of virtual nodes $N = 400$.

The first step of RC, masking, is done via multiplying a pregenerated random sparse matrix W^I by the cochleagram matrices M_i^c. The density of W^I is 25% with non-zero elements drawn from the set of $\{-1, 1\}$. The dimensions of $W^I \in \mathbb{R}^{N\times 86}$ are related to the number of cochleagram channels and the number of reservoir nodes. The resulting masked input M_i^u is flattened into the one-dimensional input signal $u(t)$ that is then nonlinearly transformed by Ikeda delay dynamics. Here we employ the driven Ikeda DDE (Eq. (3.2.1)) with a sinusoidal[21] nonlinearity $f(x) = \beta \sin^2(x + \Phi_0)$:

$$\varepsilon\dot{x}(t) = -x(t) + \beta\sin^2(x(t-1) + \rho\cdot u(t) + \Phi_0) \tag{3.4.8}$$

Parameters $\varepsilon = 2.5\cdot 10^{-3}, \Phi_0 = 0.01$ are kept fixed, whereas β and ρ are optimized via grid search. The optimal result was obtained for $\beta = 1.3$ and $\rho = 0.4$. Alternatively, all the parameters could be optimized simultaneously using genetic algorithms or another

[21]We also tested the hard sigmoid nonlinearity yielding worse recognition accuracy. This problem can be an interesting starting point for the future investigation in the context of speech recognition task.

multiparametric optimization technique, as it was done e.g. in chaotic time series prediction task. To measure the model effectiveness, we perform a 5-fold cross-validation (see Section 3.2.4), meaning that each training dataset D_i^r consists of 400 voice recordings, and each corresponding test dataset D_i^t has 100 recordings. As a result, we get a word error rate (WER) equal to 0.4%, i.e. two wrong recognitions out of 500. We run the experiment ten times shuffling the dataset, and the content of randomly chosen training versus testing datasets insignificantly affects the performance ($\pm 0.4\%$).

In the RC community there exists a hypothesis (formulated in [114], see also [115, 116]) that the dynamical system utilized for information processing should operate on the "edge of chaos". The asymptotic fixed point state—recoverable when no information is coming—should allow for the approximation property of the system. The proximity to the bifurcation threshold, on the other hand, should allow for a multitude of possible trajectories of the system, driven through the high dimensional state space while processing the input. That should preserve the separation property.

To check the consistency of the edge of chaos hypothesis, consider Ikeda dynamics (Eq. (3.4.8)) with no input, i.e. $\rho = 0$. Now, the bifurcations of such autonomous system can be studied depending on the control parameter, amplification β. Scanning the control parameter reveals a transition from a fixed point to a periodic, and then, chaotic motion. The obtained bifurcation diagram is depicted by green dots in Fig. 3.4.5.

Then, the speech recognition is solved progressively scanning β. Blue crosshairs depict the corresponding classification error. The best results are obtained close to the first bifurcation threshold, around $\beta_{op} \simeq 1.3$.

Aurora benchmark Introduced in the early 2000s by the Motorola company, the AURORA-2 benchmark test is a speech recognition task known in the telecom industry. The test introduces not only clean data, but also artificially added noise that was previously recorded in different environments (car, subway, street, etc.). Moreover, the number and the variety of recordings is much richer than in TI-46, e.g. the data in Aurora-2 were collected with the help of speakers of mixed age and gender. We employ Aurora-2 to study how can a real-world problem be tackled with RC.

Similarly to the previous benchmark, we preprocess the data using Lyon's Ear model. Sampled at 8 kHz, sound waveforms produce cochleagrams with 64 frequency channels. To be able to compare the results with the ones of TI-46, we solve the isolated spoken digit recognition problem. There are more than 5100 records of isolated digits, which comprise four major parts: one training dataset (2400 records) and three different test datasets (2700 records in total). The different test subsets, in additional to clean data, provide several noisy environments. In test A there are subway, babble, car, and exhibition noisy environments; test B, restaurant, street, airport, and train station;

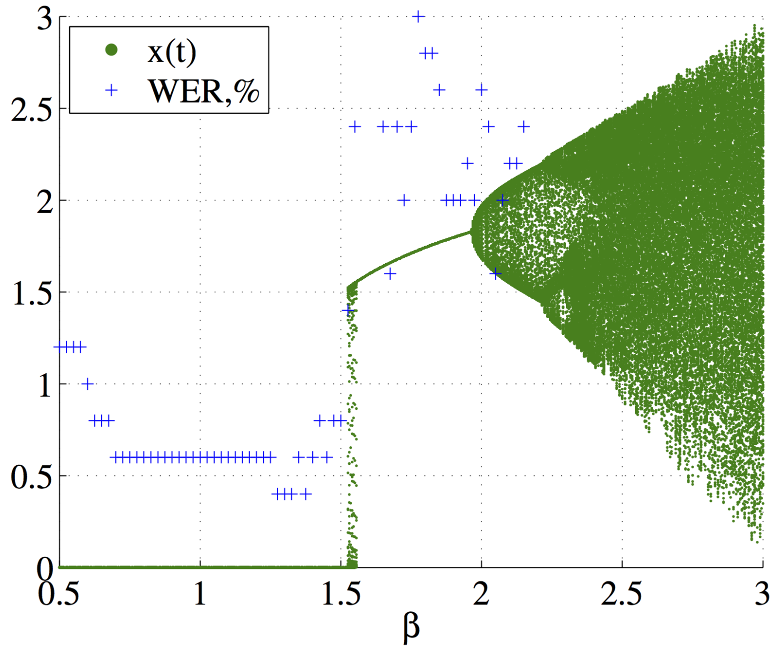

Figure 3.4.5: Accuracy of speech recognition in terms of WER (blue crosshairs) depending on the control parameter β and corresponding bifurcation diagram (green dots) of Ikeda delay dynamics underlying RC.

and test C, subway and street, respectively. Additionally, all the data in Test C have artificially altered spectrum. That corresponds to acceptable distortions during data transmission.

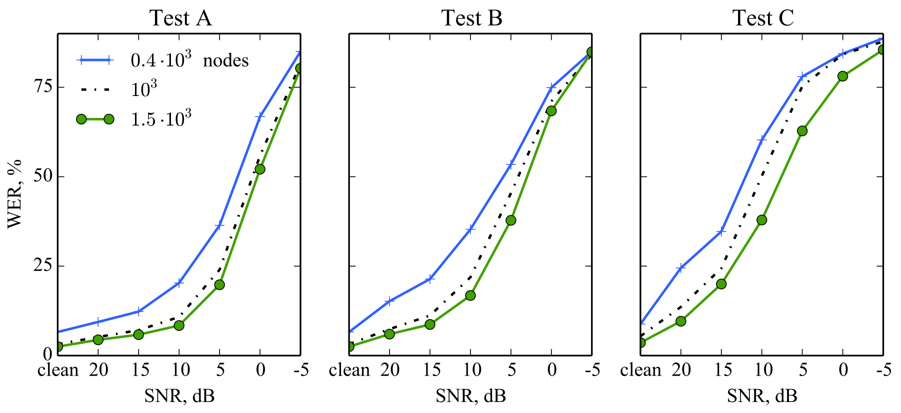

Figure 3.4.6: AURORA-2 benchmark results: average WERs for reservoir sizes of $0.4 \cdot 10^3$, 10^3 and $1.5 \cdot 10^3$ virtual nodes trained on mixed (clean and noisy) data.

Since the digit “0” is pronounced either as “zero” or just “o”, we define eleven possible outcomes instead of ten. The input mask $W^I \in \mathbb{R}^{N\times 64}$ is a sparse matrix generated similarly to the previous task, with non-zero elements drawn from set of $\{-1, 1\}$. Empirically, we found that input masks W^I with 30% density are performing slightly better than those with 25% density. We test reservoirs with $N = 400$, 1000 and 1500 nodes. The reservoir's dynamics model is the bandpass model with a sinusoidal nonlinearity:

$$\begin{aligned} \tau\dot{x}(t) &= -x(t) - \delta \cdot y(t) + \beta \sin^2\left[x(t-\tau_D) + \rho u(t) + \Phi_0\right], \\ \dot{y} &= x(t), \end{aligned} \tag{3.4.9}$$

where $\tau = 3.125 \cdot 10^{-2}$, $\delta = 0.828$, $\beta = -1.297$, $\Phi_0 = -3.5$, and $\rho = 0.61$ are parameters optimized by the GA exclusively on the training data. The results are summarized in Fig. 3.4.6. With sufficiently large number of nodes, the recognition is robust (under 10% WER) for clean data and noisy data with signal to noise (SNR) ratios of 20 and 15 dB in tests A and B. However, our system is robust only for clean data and SNR=20 dB in test C. See also Section 4.3.3 for comparisons with the state of the art in RC and with our hardware implementation; in Appendix A there is also a detailed report corresponding to each test.

3.5 Conclusion

Future computing systems are not restricted to electronics. Biologically-inspired computing systems are promising to drastically increase energy efficiency comparing to modern devices. One of those alternatives is reservoir computing (RC), a simplified approach to training RNNs that is fast and converging [12]. RC enables usage of dynamics as computational substrate and takes advantage of the underlying system's complexity. In the Chapter, we introduced a single-node approach to RC [14, 68] and employed DDE with a nonlinear delayed feedback (Eq. (3.2.1)). The delay was regarded as a memory containing state of a virtual network of time-multiplexed nodes in continuous time.

It was shown that the single-node approach to RC can be efficient in discrete time. As a confirmation, all the simulations in the Chapter have exploited the time-discretization of nodes. This optimization was targeting digital devices implementing RC. Another crucial change was nonlinearity f simplification in DDEs. Additionally, generic algorithms, a subset of evolutionary algorithms, were employed to obtain an optimal DDE parameter settings.

We ran a series of benchmark tests to validate the discrete-time RC realization. Performing chaotic time series prediction with RC technique, we demonstrated that prediction is possible even when (1) the exact model is unknown (hidden from RC) and (2) the initial conditions are unknown or randomly generated. Two dynamical models were compared: a classical Ikeda and an Ikeda model with a bandpass filter. A surprising fact was that the addition of the integral term significantly improved the time series prediction. A possible reason is a long-term memory requirement, which was enhanced by the integral term inclusion. Finally, in voice recognition tasks, we have demonstrated that reservoir's virtual network increase was needed as the problem's complexity was growing. Next, we will utilize our optimizations in a digital electronics RC implementation.

Chapter 4

FPGA implementation of delay-based RC

In this part of work, RC implemented in a digital hardware is discussed. FPGA, a flexible platform for digital systems design, is chosen for RC research and implementation. Low energy consumption and real-time processing support make FPGA highly valuable in practical applications. Recent progress in FPGA technology causes rapid changes in the electronics market[1]. We also consider FPGA as a promising RC platform.

4.1 FPGA basics

4.1.1 FPGA introduction

A field-programmable gate array (FPGA) is a reconfigurable integrated circuit. It can be viewed as an array of logic blocks, routing channels and I/O (input-output) pads (Fig. 4.1.1). Each logic block consists of multiple *logic cells* (Fig. 4.1.2) that can be configured using so-called *lookup tables* (LUT) memory. Before FPGA device can operate, the LUTs need to be populated with a circuit configuration.

All the FPGA circuits are running in parallel, i.e. every component of the system is working independently from others. That is the main difference from information processing in CPU (central processing unit) where only one command is executed per

[1]Recently, Intel bought Altera, one of the two biggest FPGA companies. That was a consequence of Microsoft's advances in information processing using FPGAs, see https://www.wired.com/2015/06/microsoft-knows-exactly-intels-future/.

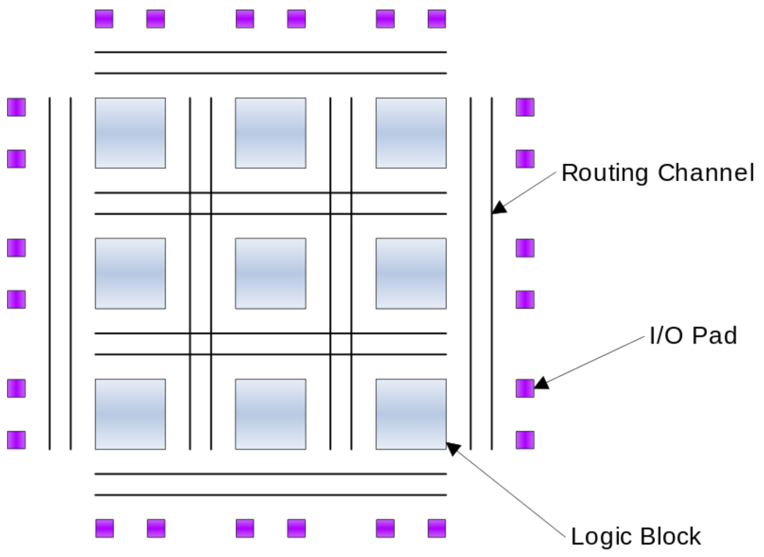

Figure 4.1.1: Simplified FPGA architecture. The logic blocks are interconnected via routing channels. Processed information is entering/coming out the FPGA through the I/O pads. Image source [117].

clock cycle. FPGA's massive parallelism allows achieving a processing system speedup. Techniques and mechanisms of building a reliable RC system that leverages its built-in parallelism will be discussed in Section 4.2.

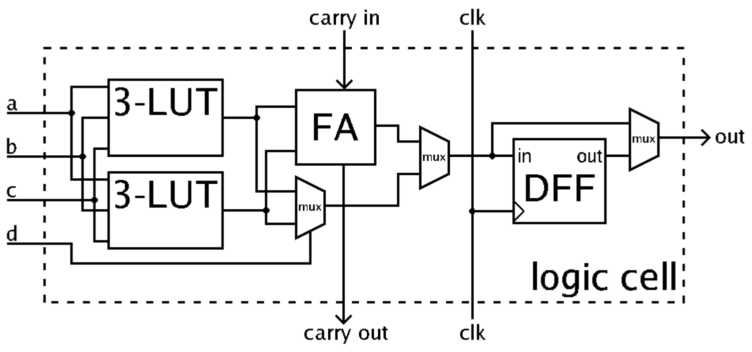

Figure 4.1.2: A typical FPGA logic cell (simplified) comprising two lookup tables, full adder, three multiplexers, and a flip-flop logic elements. The function of this circuit is defined by configuration loaded in lookup tables. Image source [118].

The main advantages of implementing RC on FPGA:

1. FPGAs are standalone, meaning there are no dependencies on external comput-

ing devices, e.g. CPUs, for all information processing steps. The coefficients, parameters and other details of initial configuration can be stored in the FPGA's memory.

2. Their reconfigurability is valuable for a use case where different hardware configurations can be tested with easiness comparable to conventional programming. This property allows designing flexible research platforms to study different RC aspects.

3. Massive parallelism allows increasing the processing speed.

4. Working in so-called *hard real-time*, FPGA is useful for systems performing diagnostics and control systems.

5. Heterogenous computing: in addition to RC units, FPGA can easily host a "soft" CPU core, FFT and other information processing cores, all of them working in parallel.

6. FPGA is a standard electronic component, off-the-shelf device capable to interface virtually any existing electronic equipment.

7. Low energy consumption of a typical FPGA device might be crucial for real-world applications, such as embedded systems working on battery. For instance, depending on configuration and clock speed, the estimated power requirement for Xilinx Artix-7 FPGA in the context of RC is 0.2–0.4 W [119]. It is significantly lower than, for instance, 73–95 W consumed by Intel Core i5 CPU family or 43 W by Radeon HD 7750 GPU[2].

8. Compact size and low costs. The latest low-end FPGA boards can be credit-card size and cost as low as 100 €. That is much cheaper than, for example, an application-specific integrated circuits (ASIC) alternative. Any functionality supported by ASIC is also achievable by FPGA. On the other hand, an FPGA design can be used later to build a dedicated RC chip.

However, there are several common technological limitations of FPGAs:

1. FPGAs, especially the cheapest ones, have slower clocks than CPUs.

2. FPGA design is very specific and demands to satisfy much more constraints, comparing to software design (discussed in Section 4.1.2.1).

3. Tools for FPGA design lag far behind their software development counterparts (IDEs, analysis tools, hardware description languages, etc.).

[2]A modern graphical processing unit with low power consumption.

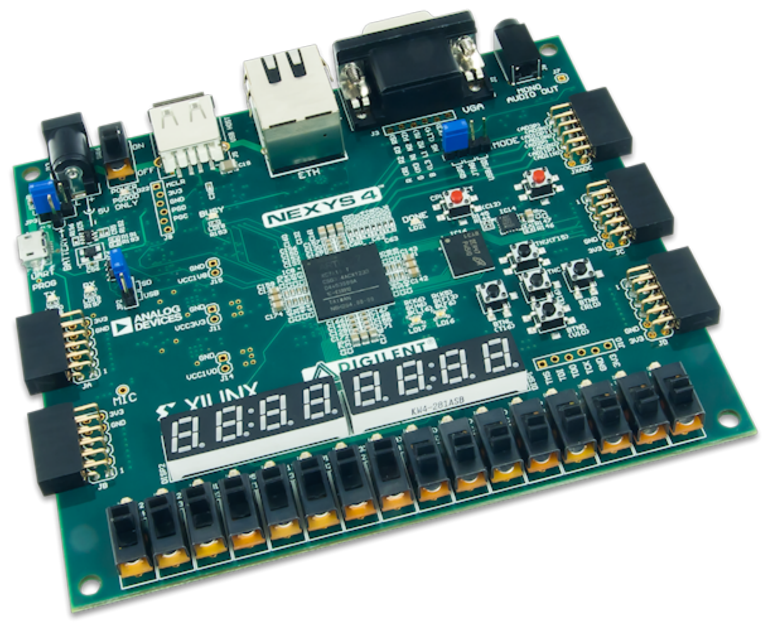

Figure 4.1.3: Digilent Nexys4 hosting Artix-7 FPGA chip, upon which RC is implemented.

4. The number of so-called *open-source* designs is very limited both for component designs (known as Intellectual Property (IP) blocks) and for hardware (FPGAs and supporting boards itself). The commercial licenses are often restrictive and expensive.
5. The previous three limitations inevitably lead to longer development cycles, in contrast to CPU or GPU programming.

Comparing also to other RC trends such as photonic implementations, FPGA-based RC is competitive in many practical aspects. However, there are situations where alternative implementations win. For instance, photonic RC devices are capable of much higher processing speeds. However, those can be exploited only when there is no I/O bottleneck, i.e. the information has to be already encoded as a light signal. Otherwise, the information processing speed would be limited by electronic devices linked with optical components.

FPGA-based RC has the biggest potential in the industries where a combination of low costs and low energy consumption is a requirement. A viable FPGA-based RC application candidate is a *fuel-cell* device. The fuel-cell is an electricity source exploiting energy of exothermic chemical reactions. It is different from batteries since it requires constant supply of chemical reagents such as oxygen (coming from air) and hydrogen. The problem is the performance degradation and limited lifetime of fuel-cells [120, 121], complex nonlinear dynamical systems. To increase the lifetime, "smart" real-time monitoring and control is needed [122, 123, 124]. The tricky requirement is that fuel-cells should also provide energy for their own monitoring/control. That is why minimizing energy impact of monitoring devices is also critical.

Under the framework of the BIPhoProc (Brain-Inspired Photonic Processor) project (ANR-14-OHRI-0002-02), a neuromorphic processor is being constructed for the fuel-cell diagnostics. BIPhoProc is a joint project between the Energy department at FEMTO-ST and the FCLab at CNRS Research Federation. The FPGA device has been chosen as it satisfies both critical requirements: (1) real-time operation and (2) low energy impact. The FPGA-based RC architecture described in the remaining Chapter is directly relevant to the project.

4.1.2 Resources of FPGA chip and supporting board

All the electronic circuits in FPGA are working simultaneously, inherently allowing massive parallelism[3]. However, modern FPGA chips are more than logic gates and programmable interconnections between them. Apart from programmable logic itself, FPGAs host a variety of integrated circuits, called *primitives* or *hard blocks*, such as block RAMs (random-access memories), DSP (digital signal processing) slices[4], etc. The adjective "hard" used in this context indicates that the implemented logic is pre-defined on a factory, i.e. hard circuits are not programmable. On the other hand, they may allow higher processing speeds comparing to programmable logic. Physical proximity of the hard blocks to reconfigurable circuitry permits to easier satisfy physical device constraints.

4.1.2.1 Physical constraints

When developing a FPGA design, one has to bear in mind several fundamental limitations of an FPGA project. First of all, there is a finite number of the logic elements and interconnections between them. Thus, the FPGA design should be small enough to to be mapped onto an actual FPGA device. This characteristic is known a consumed *area* of FPGA. The smaller is the area, the less logic elements are involved.

Another crucial constraint is *timing*. When developing an FPGA project, one needs to respect the signal's arrival time limit. In the simplest case, if the path between logic elements exchanging signals at the same clock cycle is too large, the signals may

[3]E.g. Artix-7 FPGA device XC7A100T provides 101440 logic cells, which in total contain millions of logic gates.

[4]DSP slices implement predefined circuits such as multipliers, accumulators, binary pattern matchers, etc.

not arrive before the next clock cycle. That may cause an unstable and unpredictable behavior of the circuit. Often, logically correct circuits may be physically unimplementable because of this issue. To alleviate this, there exist several design strategies such as logic simplification, saving information in the intermediate registers[5], decreasing the clock frequency and others. To achieve *timing closure*, the project may need to be redesigned with the help of the latter techniques.

Speed and synchronization problems are also noticeable when dealing with FPGA. *Latency* describes the number of clock cycles needed for information unit to finish its task. For example, *multiply-and-accumulate* circuits frequently do multiplication during the first clock cycle and addition during the next one. Thus, latency is two clock cycles. *Throughput* is the number of processed bits per clock cycle, which is inversely proportional to latency. *Synchronization* problems may arise when dealing with circuits running in parallel and sharing common data or instructions.

The need to satisfy not only the functional requirements but also the physical constraints, elevate FPGA design complexity comparing to CPU. Unlike FPGA. in CPU programming there is a single constraint, functional correctness. CPU program is considered correct if it is functionally correct. Decreased complexity of CPU programming process is only possible because problems of area, timing, synchronization and so on are solved by the designers of a CPU itself. The very idea of CPU is a reusable (programmable) electronic circuit without need to be rewired. FPGAs, on the other hand, are circuits that need to be "rewired" in order to be useful. This rewiring, however, gives any FPGA designed more freedom and enables all the processes to be executed at the same time.

4.1.2.2 Major types of memory

Special attention has to be paid to different available memories. Right choice of memory may significantly improve the throughput and ability to achieve a timing closure. There exist various kinds of memories and in general they may be classified as (1) internal with respect to the FPGA chip or external; and (2) volatile or permanent. Volatile memories tend to operate faster than permanent, but they usually need constant power supply. Permanent memory, on the contrary, stores information even after device is powered off.

The most common internal volatile memories are distributed RAM and block RAM. The first one is implemented using FPGAs memory elements (lookup tables) and the second one constitutes *hard* memory units hosted on the same chip as the FPGA logic. Proximity to programmable logic implies shorter wire lengths, and thus, allows for

[5]The technique is known as *pipelining*.

higher clock rates. Often, FIFO (first-in-first-out) memories are based on available block RAMs, however distributed RAM may be employed for FIFO functionality as well. The downside of using distributed RAM is increased FPGA project implementation time as a result of individual lookup tables allocation by accompanying FPGA design tools.

There exist a variety of external volatile memories, such as cellular RAM, DDR3 RAM, etc. External memories tend to have larger volumes, but lower throughput than internal ones. Finally, external non-volatile memories, e.g. flash memory, are used to permanently store information. In practical applications, FPGAs can be self-configured from data stored on a non-volatile memory device.

4.1.2.3 Supporting board

Typically, FPGA devices are sold already integrated into support boards. That boards may contain periphery such as different I/O devices, external memories, sensors, etc. Our project was realized on a Artix-7 FPGA hosted on Digilent Nexys 4 board (Fig. 4.1.3). For RC development along with the FPGA logic we utilize on-chip block RAMs and PLLs (phase-locked loops) to generate a system clock (@50 MHz) and slower peripheral clocks. I/O components of the support board: 7-segment displays, switches, LEDs[6], UART[7] port are also used. For practical applications, we interface 12-bit AD (analog to digital) and DA (digital to analog converters) converters[8] connected to the Nexys 4 board through so-called PMOD[9] ports. The maximal throughput rate of the current analog-digital converters pair is limited by 1 MHz, yet it can be increased by replacing the devices.

4.2 FPGA implementation of RC

The first FPGA demonstrations of a single-node RC approach appeared in [35, 125]. In [35], a mixed digital-analog RC circuit was introduced: the nonlinearity f of the Mackey-Glass type (Eq. (3.4.1)) was analog and external to the FPGA, while the rest of the components were digital and implemented by the FPGA (masking, reservoir's delay, readout, etc.). TI-46 spoken digits recognition and one-step-ahead Santa Fe chaotic time series prediction [126] tasks were reported. In [125], the demonstrator employed a purely digital circuit that was also simulating the reservoir dynamics in a

[6]Light-emitting diodes.

[7]Universal asynchronous receiver/transmitter.

[8]The particular models we utilize are Digilent Pmod AD1 and Pmod DA4.

[9]Peripheral Module interface defined by Digilent company.

form of Mackey-Glass type DDE, solved with a first-order Euler method. A temporal pattern classification task as well as time series prediction were reported.

In our work, a completely digital circuit is employed with the nonlinear delayed-feedback reservoir dynamics simulated by the second-order Heun's method (Section 3.3.1). That is expected to improve numerical integration accuracy using equivalent to [125] number of bits (16 bits). Along with time-series prediction, we evaluate RC on a challenging task of speech recognition.

4.2.1 High-level RC implementation

Reservoir computer in FPGA implements all the principal blocks introduced in Chapter 3. Executed on FPGA, system consists of masking, reservoir and readout layers (Fig. 4.2.1).

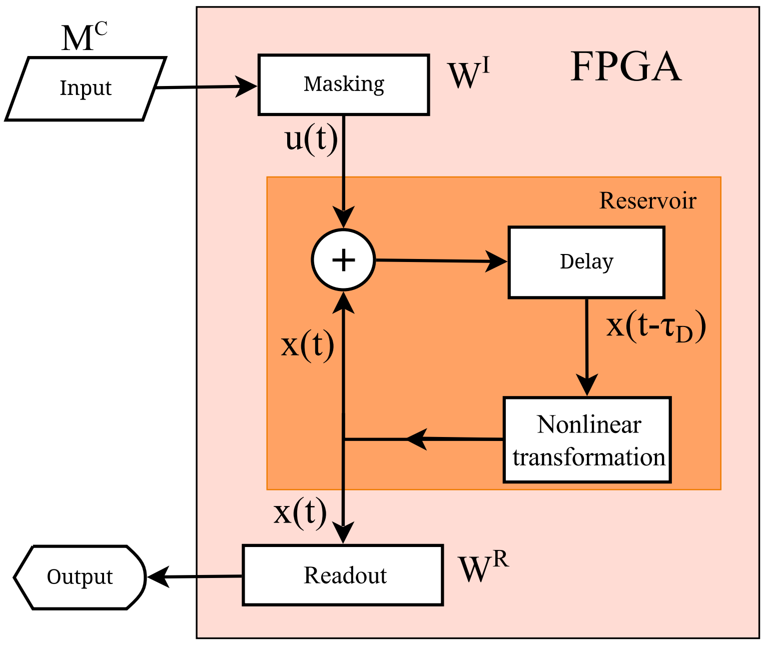

Figure 4.2.1: Complete RC implementation based on FPGA. The readout is trained beforehand (offline learning).

The *masking layer* is a matrix multiplication operation $M^u = W^I \cdot M^c$, where the input mask W^I is a matrix residing in a readonly memory (ROM) of the device and M^c is the input data matrix. As described in Section 3.2.3, the masked data matrix $M^u \in \mathbb{R}^{N\times T}$ is injected to the *reservoir* as a signal $u^k[n], n = 0 \ldots N-1$ where $u^k, k = 0..T-1$ is a k-th column in matrix M^u. The reservoir is a digital implementation of Ikeda

DDE (Eq. (3.2.1)) with a delay stored in a FIFO-memory. The nonlinear function f is implemented either as a stored in ROM lookup map for arbitrary nonlinearities (e.g. sinusoidal $f(x) = \beta \sin^2(x + \Phi_0)$) or a circuit, e.g. realizing the hard sigmoid $f(x) = \max(0, \min(b, x - a))$, to save the FPGA's resources. The reservoir's state $x[n]$ is added to a masked input signal $u[n]$. On the other hand, the internal state $x[n]$ is transferred to the next layer, readout. The readout matrix W^R is calculated on a computer during offline learning and is also stored in ROM. Columns in matrix M^x are constructed from $x[n]$, coming from the reservoir's transformation. Multiplied by the internal state matrix M^x, W^R produces M^y, the result of reservoir computation. Arrows in Fig. 4.2.1 represent 16-bit data buses. To synchronize the data flow we apply a technique producing correct by construction circuits.

4.2.2 Data flow

4.2.2.1 Correct by construction circuits

The principal blocks in Fig. 4.2.1 such as masking, reservoir dynamics, and the readout need a communication channel. In the most simple case, the communication can be done synchronously, i.e. all the data points move on the same clock. However, those components may work on different timescales. Communication with PC is also asynchronous. Therefore, clocked asynchronous communication is preferable. In certain cases, data may be buffered by a dual-clock FIFO memory.

However, this issue can be also resolved with information processing flows that produce *correct by construction* circuits, i.e. when a combination of correct components produces a correct bigger circuit. This method has an additional advantage when verifying the circuits: every component may be tested in isolation.

We employ a synchronization technique known as the *backpressure* protocol[10]. A node in the data flow conforms to the protocol if:

1. It does not consume incoming data when the input data is not valid.
2. It updates its outputs only when the next node is ready.

Correct-by-construction circuits approach also means that new components can be effortlessly added. Moreover, the backpressure protocol is implementing *pipeline parallelism*, a parallelization strategy to speedup computation of sequential data.

[10]For more technical information see https://hackage.haskell.org/package/clash-prelude-0.10.6/docs/CLaSH-Prelude-DataFlow.html

4.2.2.2 Pipeline parallelism

Parallelization allows computation speed-up without increasing the clock speed. There are two main parallelization strategies: data parallelism and pipeline parallelism. The first one is achieved by dividing the data between identical processing units. For instance, that is the strategy widely applied in GPU programming. However, not all data can be equally split, e.g. numerical DDE solving. Still, processing speed can be increased by applying a parallelization strategy known as *pipeline parallelism.*

Pipeline parallelism increases processing speed over a sequence of data that travel between several processing units (e.g. masking, reservoir transformation, readout). The pipelined circuits are composed in analogous way to automobile assembly line. In automobile production, a semi-finished assemblies are simultaneously passed from stage to stage where the parts are added. For instance, on the early stage an engine is installed, on the next stage the hood is installed, then, the wheels and so on. If each part requires 20 minutes, then to build an automobile of three parts without a pipeline, 60 minutes are needed per automobile. Employing a pipeline strategy, the responsibility is split between several stages working simultaneously. Building the first automobile, while the first stage is working, the second and the third are waiting. After the part has passed the first stage, it passes to the second stage and a new part arrives. Now, the first stage is working with the second part and the second state, with the first one. Finally, the first part comes to the last stage, the next part, to the second stage and a new part arrives to the first stage. After the first hour, the first car is produced. However, the second car is ready after only 20 minutes after the first car was built. The reason for that is all the stages were working in parallel. Thus, every new car can be produced in 20 minute intervals. The same idea is in information processing pipeline. Every major component (a so-called IP block) in Figure 4.2.1—such as the *masking* block, the *reservoir*, and the *readout*—is a stage in the pipeline.

In reality the situation may be more complicated. For instance, if the last stage in the automobile assembly line takes 30 instead of 20 minutes, that immediately increases the interval of automobile production to 30 minutes. In that case, other techniques might be applied for such *bottleneck* stages to minimize waiting time. For instance, the analogy to data parallelism in automotive industry is wheels installation. Each pair of wheels are mounted independently and simultaneously. This parallelization divides time from 30 to 15 minutes. In FPGA-based RC for instance, we may exploit data parallelism to speed-up matrix multiplication (Section 4.2.3).

4.2.3 Masking and readout operations

Both masking and readout operations are described in the most general way as matrix multiplications $\mathbf{M} = \mathbf{A} \cdot \mathbf{B}$ (Fig. 4.2.2). In both cases, the multiplied matrix $\mathbf{A} \in \mathbb{R}^{K \times N}$ is constant and the elements of multiplying matrix $\mathbf{B} \in \mathbb{R}^{N \times T}$ come from the signal $b_j \equiv b[j], j = 1, 2, \ldots$. Our system is required to work continuously in real time, that means the multiplying matrix $\mathbf{B}$ tends to have an infinite number of columns $T \to \infty$. Let $\mathbf{b}^{(n)} \ni b_j, n = 1, \ldots T, j = 1, \ldots, N$ be the n-th column-vector in $\mathbf{B}$. Then, to satisfy the constraint $T \to \infty$, it is sufficient to perform matrix-vector multiplication $\mathbf{A} \cdot \mathbf{b}^{(n)}$ in time before the next vector $\mathbf{b}^{(n+1)}$ becomes available.

$$\overset{\mathbf{M}}{\left(\left.\underbrace{\begin{pmatrix}\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\end{pmatrix}}_{T}\right\} K\right)} = \overset{\mathbf{A}}{\left(\left.\underbrace{\begin{pmatrix}\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\end{pmatrix}}_{N}\right\} K\right)} \times \overset{\mathbf{B}}{\left(\left.\underbrace{\begin{pmatrix}\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\end{pmatrix}}_{T}\right\} N\right)}$$

Figure 4.2.2: Matrix multiplication

To achieve higher throughput, the operation of matrix-vector multiplication can be parallelized by dividing the multiplied matrix $\mathbf{A}$ horizontally into several sub-matrices. Then the coefficients of the sub-matrices have to split up among matrix-vector multiplication circuits. All those circuits then receive the input signal $b_j \in \mathbf{b}^{(n)}$. This kind of procedure exploiting data parallelism is known as SIMD (single instruction, multiple data) computing.

There are many possible ways to design a matrix-vector multiplication circuit. In the case of asynchronous data flow such as backpressure protocol (Section 4.2.2), a three-stage matrix-vector multiplication can be implemented: replication, multiplication-accumulation, and filtering. The advantages of the approach are: (1) support for asynchronous data flow, (2) only one memory buffer storing intermediate results is required, and (3) each of the stages can be verified in isolation. During the replication stage, the received data point $b_j \in \mathbf{b}^{(n)}$ is repeated K times and is sent to the second stage. During the second stage, partial multiplication products $m_i^{(j+1)} = m_i^{(j)} + a_{ij} \cdot b_j$ are calculated sequentially. Here $i = 1, \ldots, K,\ j = 1, \ldots, N$, $\mathbf{m} = (m_1, m_2, \ldots, m_K)$ is a memory buffer storing the partial products, and a_{ij} is a value at i-th row and j-th column in the matrix $\mathbf{A}$. During the last stage, filtering, intermediate multiplication products are ignored and only the final matrix-vector multiplication results are returned.

4.2.4 Reservoir dynamics

Both FPGA and computer implementations share Heun's scheme to solve DDE (Section Section 3.3.1). However, FPGA does not provide floating point arithmetic. The available fixed-point fractional number representation has precision limited to 2^{-m}, where m is the number of bits after the radix point. This may cause discrepancies between the implementation on PC and the one on FPGA. Additionally, the system has to be monitored for *overflows*, i.e. incorrect computation results when the existing number of bits is not sufficient to represent the values. Staying invisible to the digital hardware user, overflows are severe computation mistakes, therefore, they should be avoided at all costs.

In our RC implementation, the DDE solver is operating on 16-bit data values. The representation of fractional numbers is the following: 1 bit determines the sign, 2 bits are representing the integral part and 13 bits stand for the fractional part. The values are bounded between $4-2^{-13} \approx 3.99988$ and -4.0. Our preliminary investigations have shown that for certain tasks, such as spoken digit recognition minimum 5 fractional bits are needed, while for time series prediction higher precision of at least 13 fractional bits is required. Thus, minimum 16 bits are needed to cover both types of tasks.

The second order fixed step Heun's method:

$$\begin{aligned} \widetilde{x}_{n+1} &= x_n + \Delta t \cdot F\left[x_n, x_{n-\text{delay}}, u_n\right], \\ x_{n+1} &= x_n + \tfrac{\Delta t}{2} \cdot \left(F\left[x_n, x_{n-\text{delay}}, u_n\right] + F\left[\widetilde{x}_{n+1}, x_{n-\text{delay}+1}, u_{n+1}\right]\right), \end{aligned} \tag{4.2.1}$$

where delay is the depth of a FIFO-based delay memory and $F[\cdot]$ represents the right hand equation of the DDE being solved:

$$\tau \dot{x}(t) = -x(t) + f\left(\beta, x(t-\tau_D) + \rho u(t)\right),$$

Using discrete notation $x_n \equiv x[n]$ and dividing both sides by τ, the last equation can be rewritten as

$$F\left[x_n, x_{n-d}, u_n\right] = \frac{dx}{dt} = \frac{1}{\tau}\left(-x_n + f\left(\beta, x_{n-\text{delay}} + \rho u_n\right)\right).$$

A fixed time step $\Delta t = 10^{-2}$ is used. From digital electronics point of view, multiplication by 10^{-2} can be approximated using a shift by 6 bits to the right, thus, $\Delta t \simeq 2^{-6} = 1.5625 \cdot 10^{-2}$. Automatically derived by GA, τ is e.g. $3.125 \cdot 10^{-2}$ or $1/\tau = 2^5$, thus $\Delta t/\tau = 1/2$. This allows for simplification of Eq. (4.2.1)

$$\begin{aligned} \widetilde{x}_{n+1} &= x_n + \tfrac{1}{2} \cdot \tilde{F}\left[x_n, x_{n-\text{delay}}, u_n\right], \\ x_{n+1} &= x_n + \tfrac{1}{4} \cdot \Big(\tilde{F}\left[x_n, x_{n-\text{delay}}, u_n\right] + \tilde{F}\left[\widetilde{x}_{n+1}, x_{n-\text{delay}+1}, u_{n+1}\right]\Big) \end{aligned} \tag{4.2.2}$$

where $\tilde{F}(x_n, x_{n-\text{delay}}, u_n) = -x_n + f\left(\beta, x_{n-\text{delay}} + \rho u_n\right)$. This scheme is implemented in FPGA using the pipelining technique, where intermediate operations of multiplication and addition are stored in register stages.

Another problem in reservoir dynamics implementation is approximation of a nonlinear function f. There are several alternative ways to resolve this issue, such as CORDIC (COordinate Rotation DIgital Computer) algorithm to calculate trigonometric functions or interfacing FPGA with an external analog nonlinear circuit. We employ yet another solutions.

To emulate sinusoidal nonlinearities, we build a lookup memory table with 14-bit address range effectively covering range $[-2; 2)$ with 2^{-12} *step* (or address) resolution. The table contains 14-bit unsigned integers interpreted as signed fixed-point values with 13 fractional bits, i.e. table has 2^{-13} *value* resolution. As a simplification strategy, the lookup table is replaced with a hard sigmoid (Eq. (3.3.2)).

4.3 Performance

Nominal processing speed of our FPGA-based reservoir computer with 1000 virtual nodes is 12 Mb/s. For instance, estimated number of recognized words at this rate using our methods is up to 350 words/s[11] with FPGA working at the rate of 0.05 GHz. In contrast, a Intel Core i7-3520M CPU running at clock speed of 2.9 GHz is estimated to recognize the same number of words per second[12] using equivalent RC methods. The reasons why FPGA's processing speed does not deteriorate at slower clock rate of FPGA are inherent parallelism, usage of fixed-point instead of floating point arithmetic, and other optimizations (e.g. lookup maps). However, the mentioned CPU requires 35-40 W of power, comparing to 0.2-0.4 W by Artix-7 FPGA. As another example, real-time control tasks, the rate of 1.5 MB/s means a million processed points (12-bit wide) per second. Optimizing the number of parallel operations and interfacing to a physically implemented Ikeda dynamics can further improve the processing speed with 3× to 50× speedup, depending on the task. Implementing our design as

[11]Even faster than real-time.

[12]RC processing on a CPU may vary depending on the operating system, programming language and libraries used, etc. For our checks we used compiler GHC 7.10.2 with -O2 optimization flag. The running system was Mac OS X 10.11. Performance-critical libraries were BLAS/LAPACK, linear algebra packages.

ASIC[13], one could potentially achieve a further computation speed-up compared to FPGA. Data processing speed in our PC–FPGA demonstrations[14] is limited only to RSR232 serial communication protocol used to transfer data between the PC and the RC demonstrator. The digital circuits are designed and formally verified in a high-level functional hardware description language CLaSH[15] with a subsequent low-level VHDL (VHSIC hardware description language) code generation. In the remaining part of the Chapter we discuss FPGA-based RC evaluation using the benchmark tests introduced in Section 3.4.

4.3.1 Chaotic Mackey-Glass time series prediction

For chaotic time series prediction task, the methods described in Section 3.4.1 are used. The data points from Mackey-Glass time series are generated by a PC (*Input* block in Figure 3.4.4). The prediction is done via the Nexys 4 FPGA board connected by a serial cable. The data exchange protocol between PC and FPGA is standard RS232.

During the training step, the FPGA contains only two blocks from Figure 4.2.1: the *masking* block and the *reservoir*. During the masking stage, the input signal $M^c \in \mathbb{R}^{1\times T}$ is multiplying the input mask $W^I \in \mathbb{R}^{N\times 1}$: $W^I \cdot M^c = M^u$. The coefficients of W^I are stored in ROM. During the next stage, reservoir transformation, the signal $u(t)$ representing the elements of the matrix M^u, is perturbing the DDE dynamics:

$$\varepsilon\dot{x}(t) = -x(t) - \beta f\left(x(t-1) + \rho u(t)\right), \tag{4.3.1}$$

where f is the hard sigmoid (Eq. (3.3.2)); parameters optimized against a 20-step prediction horizon are $\varepsilon = 0.01, \beta = 1.69, a = 0.44, b = 0.81, \rho = 7.2$, and $N = 1000$ is the number of reservoir nodes. The result of processing, data matrix M^x is returned back to the PC[16]. There, all the readout weights are subsequently computed in offline fashion, i.e. via ridge regression:

$$W^R = (M^{x_1} \cdot (M^{x_1})^\intercal + \mu \cdot I)^{-1}(M^{x_1} \cdot (M^y)^\intercal), \tag{4.3.2}$$

where M^{x_1} is M^x excluding the last h columns and M^y is the teaching signal, M^x excluding the first H columns, $H = 20$ $(1.2\,\tilde{\tau}_D)$ is the prediction horizon, and $\mu = 10^{-4}$

[13]Application specific integrated circuit.

[14]Roughly 80 Kb/s, a fraction of the nominal speed

[15]http://clash-lang.org

[16]The elements of the matrix M^x are constructed column-wise from the signal $x(t)$.

is the regularization parameter. Then, the obtained coefficients W^R are placed in the *readout* block of the FPGA. This step finishes FPGA-based RC training.

Utilization of data M^x produced by the FPGA is needed to ensure that the DDE solving with fixed-point arithmetic will not deteriorate the readout accuracy. Alternatively, fixed-point arithmetic could be emulated on the PC. However, the former method with FPGA-generated M^x, is preferable as it also provides the additional data for electronic circuit verification.

After the readout coefficients $W^R \in \mathbb{R}^{1\times N}$ have been placed to FPGA, the prediction is performed as $\left(W^R\right)^\intercal \cdot M^x$ multiplication. During our research, it appeared that the accuracy of the prediction is sensitive to the resolution of the readout operation. Because of this reason, the readout arithmetic has resolution of 21 instead of 13 fractional bits. After the computation of $\left(W^R\right)^\intercal \cdot M^x$ product, the 8 extra bits are truncated before the result is sent back to PC (*Output* in Fig. 4.2.1).

The predicted signal is evaluated using 5000 points of data after the transient of 1000 points; NRMSE is the calculated error measure. In Table 4.1, there is a summary after different prediction configurations.

Description	Arithmetic	NRMSE
Simulation on PC	32 bits, Float	0.057
FPGA	16 bits, Fixed	0.059
FPGA + signal smoothing	16 bits, Fixed	0.056

Table 4.1: Comparison between PC- and FPGA-based chaotic time series prediction against the prediction horizon $H = 20$ steps ($1.2\ \tilde{\tau}_D$).

The prediction on PC is done with using 32 bit *floating point* data type. The FPGA prediction with a 16 bits *fixed width* representation (13 fractional bits, i.e. precision is 2^{-13}) results in NRMSE of 0.059. The corresponding Fig. 4.3.1 shows the difference between the original and the predicted signals. From that Figure, one may notice that after the transient period of approximately 190 data points, the dynamics of the reservoir stabilizes and the prediction becomes much more accurate. The caused *washout transient* is the result of RC predictor's internal state initialization in the beginning of the prediction. The transient effects can be less visible if, in addition to the readout weights, RC predictor is provided with appropriate initial conditions. This means the state of the delay line in the end of learning process has to be also preserved. However, in different DDE implementations (e.g. photonic) this might be not practical.

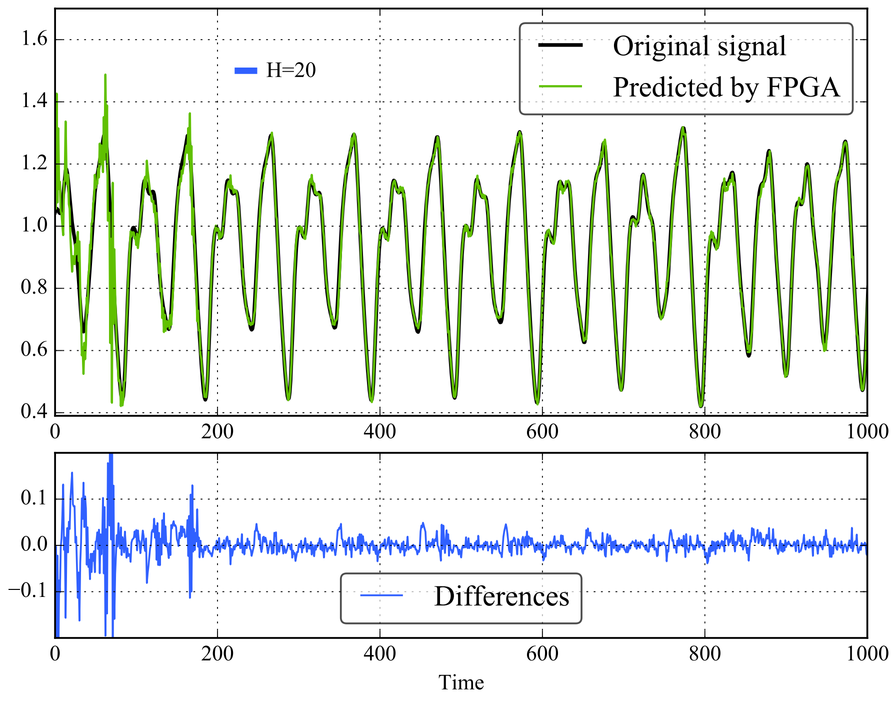

Figure 4.3.1: 20-step ahead (1.2 $\tilde{\tau}_D$ delays ahead) FPGA-based RC prediction, NRMSE = 0.059. **Upper**: The first 1000 data points predicted by FPGA (green) are transposed on top of the chaotic Mackey-Glass signal generated by computer (black). The prediction horizon $H = 20$ is marked by a blue horizontal line. **Lower**: Zoom-in of difference plot between the actual and the predicted signals.

The last row in Table 4.1 corresponds to NRMSE obtained after the predicted signal was post-processed. Post-processing is done by applying a first order *leaky integrator* filter. The filter performs smoothing over the received signal. Setting filter's parameter $\lambda = 0.5$ allows removing some of the undesired noise, and thus decreases the resulting NRMSE. Surprising outcome is that the post-processed FPGA signal results in even lower NRMSE, than that predicted with floating-point arithmetic, but without post-processing. This indicates that even without floating-point precision, FPGAs can be concurrent to CPU-based RC predictors. Leaky integrator filtering can be realized either on PC or FPGA. However, we implemented post-processing on PC only for demonstration reasons.

Figure4.3.2 generalizes prediction accuracy against different horizons, both on PC and FPGA. The dashed green line marking simulations on PC is gradually increasing as prediction horizon expands. This reflects the fact that a long-term prediction is usually less accurate than a short-term one. The horizon of 20 steps is a local minimum. That can be explained by the fact that the GA optimization targeted exactly this prediction

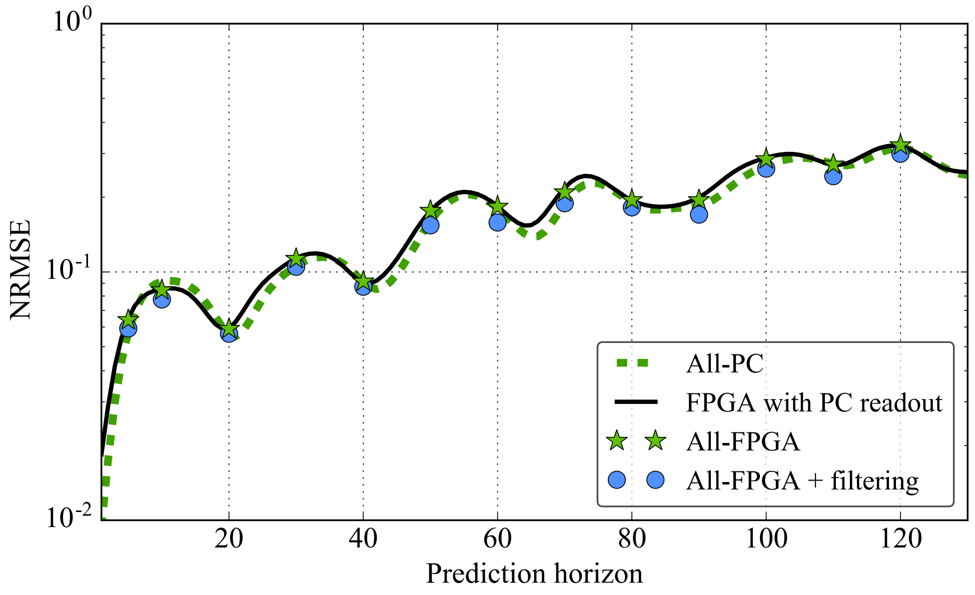

Figure 4.3.2: RC prediction accuracy with DDE (Eq. (4.3.1)). PC-based RC implementation is depicted as a dashed green line. FPGA masking and reservoir dynamics (with computer readout) is marked by black solid line. Complete FPGA-based prediction is indicated by green stars, while the best result achieved with leaky integrator post-processing, is marked by blue circles. The difference between PC and FPGA implementations is in precision: 32-bit floating point on PC versus 16-bit fixed-width on FPGA.

horizon.

Close to the prediction accuracy line comes a black solid line marking experimental results. In this experiment, FPGA is used for masking and nonlinear transformation done by DDE, and the readout operation is performed on PC. The small variation between those two lines is caused by differences in precision between PC and FPGA. Another experimental results are indicated by green stars. They are obtained exclusively with data coming from FPGA. The fact that green stars practically lie on the solid line, indicates that the readout resolution of 21 fractional bits is accurate enough to approximate the floating-point readout. Finally, the best results in terms of NRMSE, are marked by blue circles. Those are obtained with post-processing the FPGA output by the leaky integrator filter.

4.3.2 TI-46 spoken digit recognition

The architecture for the classification tasks is identical to the one used for prediction tasks and is illustrated in Fig. 4.2.1. Therefore, the procedure of training FPGA-based

RC and its evaluation is similar to those previously described. The difference, however, is in the input matrix dimensions $W^I \in \mathbb{R}^{N\times 86}$. The exploited reservoir dynamics model is from Section 3.4.2:

$$\varepsilon \dot{x}(t) = -x(t) + \beta \sin^2(x(t-1) + \rho \cdot u(t) + \Phi_0),$$

with parameters $\varepsilon = 2.5 \cdot 10^{-3}, \beta = 0.56, \Phi_0 = 0.01, \rho = 0.4$, and $N = 400$ nodes. Another feature of this new implementation is the 16-bit lookup map used to implement arbitrary nonlinear functions, e.g. sinusoidal.

As before, the readout mask W^R is trained offline:

$$W^R = (A \cdot A^\intercal + \mu \cdot I)^{-1}(A \cdot B^\intercal), \tag{4.3.3}$$

where matrix A is the horizontal concatenation of all the matrices $M_i^x \in \mathbb{R}^{N\times T}$ obtained after each processed cochleagram $M_i^c \in \mathbb{R}^{86\times T}$, and matrix $B \in \mathbb{R}^{10\times N}$ is one-hot encoded class corresponding to the correct digit value (Fig. 3.4.4). To evaluate the effectiveness of the recognition, the resulting matrices $M_i^y \in \mathbb{R}^{10\times T}$ are sent back to the computer.

On the computer, the resulting answer y is calculated as $y = \operatorname{argmax}(\tilde{y})$ where the vector $\tilde{y} = \sum_n M_i^y(k, n), k = 0..9$; and matrix row number k, is the recognized class index. The maximal column number T in M_i^y is different for each output as it is proportional to the input record length. Alternatively, the described above winner-take-all calculation can be implemented on the FPGA. For the sake of argument, however, we demonstrate that the architecture from Fig. 4.2.1 can be used without modifications.

Performing 5-fold cross-validation (see Section 3.2.4), we run independently 5 FPGA configurations. The difference between the configurations is in the readout matrices W_i^R trained on different subsets D_i^r of the initial dataset $\mathbf{D}$. Then, as before, we validate our model using testing datasets $D_i^t = \mathbf{D} - D_i^r : D_i^t \cap D_j^t = \emptyset \forall i \neq j$ (the full dataset $\mathbf{D} = \cup_i D_i^t, i = 1..5$). As a result we obtain WER = 0.2% that is comparable to our PC-based RC.

4.3.3 Aurora benchmark

The TI-46 benchmark results are indicating that RC can be applied to more complicated tasks. Previously described in Section 3.4.2, Aurora benchmark has almost five times bigger isolated digits dataset comparing to TI-46.

The procedure of FPGA-based RC validation on Aurora is very similar to TI-46. The input mask $W^I \in \mathbb{R}^{N\times 64}$. The number of the input dimensions $M = 64$ is the result of the passive Lyon's ear preprocessing scheme applied to the recordings sampled at 8 kHz. The reservoir dynamics model is:

$$\varepsilon\dot{x}(t) = -x(t) + \beta\sin^2(x(t-1) + \rho\cdot u(t) + \Phi_0),$$

with parameters $\varepsilon = 1.56\cdot 10^{-1}$, $\beta = 1.11$, $\Phi_0 = -3.02$, $\rho = 1.15$. The number of nodes is increased to $N = 1000$. W^R is trained using ridge regression as before.

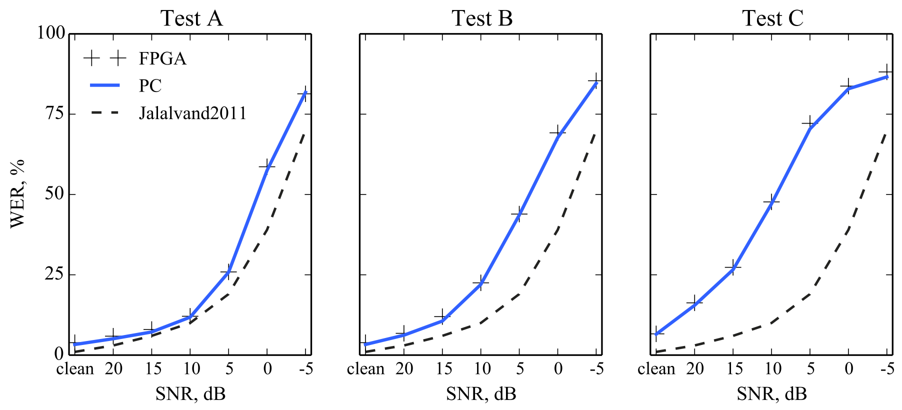

Figure 4.3.3: Comparison between FPGA and computer implementations of a lowpass Ikeda model. Dynamics parameters are: $\varepsilon = 1.56\cdot 10^{-1}$, $\beta = 1.11$, $\Phi_0 = -3.02$, $\rho = 1.15$. The reservoir's size is 1000 nodes. Jalalvand2011 indicates results in [127], averaged over the three tests.

In Fig. 4.3.3, there is a comparison between PC (solid blue line) and FPGA (crosshairs) implementations. The figure reveals a perfect agreement between both PC and FPGA. For the reference, a state of the art result in RC inspired by hidden Markov chains with 5 hidden states per digit [127] is highlighted with a dashed line. Apart from the different model, a different audio preprocessing scheme is applied. In our work cochleagrams are employed, whereas in [127] the Mel Frequency Cepstral Coefficient (MFCC) technique is applied. Another significant difference in our work is a monolithic reservoir of $N = 1000$ nodes, versus 4000 nodes in [127]. Due the limitations of a low-cost FPGA hardware, we were not able to scale to 4000 nodes. While Test A gives results pretty similar to [127], even with only 1000 nodes, Tests B and C reveal that there might be a potential problem with signal preprocessing in our case. More detailed experimental FPGA recognition results are summarized in tables of Appendix B.

4.4 Conclusion

FPGA is a flexible platform for digital electronics design. Containing millions of logical gates working in parallel, FPGA is used for performance-intensive tasks such as real-time image and video processing, prediction, control and others. Another advantage of the device is relatively low costs and low energy consumption. That makes FPGA is a strong candidate for applications such as real-time monitoring and control on mobile platforms. That is why we consider FPGA as a promising RC platform.

For the sake of demonstration, in FPGA we developed a simple RC architecture from Fig. 4.2.1 consisting of the three principal stages: masking, reservoir, and readout. All the principal stages asynchronously communicate with each other, as well as with PC providing the data. The masking and the readout components were implemented as matrix multiplication operations, while reservoir dynamics was embodied in the Ikeda-like DDE, without the integral term. Our RC architecture was reused without modifications both for time series prediction and classification tasks.

We have reproduced the four benchmark tests from Chapter 3 using the FPGA-based RC implementation. Even though the computation resolution of FPGA is worse compared to CPU, i.e. there is no floating number support, the results stand close to the ones obtained in a computer simulation. Surprisingly, FPGA-based time series prediction consistently outperformed the PC-based version. That was the result of applying a noise-removing filter to the FPGA prediction. As a consequence, under appropriate post-processing techniques, FPGAs can be applied to prediction tasks without any major accuracy loss. Another tasks, voice and recognition, produced results close to the PC-based version.

Finally, FPGA's ability to work in hard real-time is beneficial for interfacing with other physical systems. For instance, reservoir dynamics that was simulated inside FPGA as Ikeda DDE instead can be implemented by a real laser dynamics. The only change to FPGA is plugging in a pair of AD-DA converters.

Discussion

Chimera states

A fresh look on phenomena in the networks of coupled oscillators was suggested in present work by discovering purely temporal chimera states [79, 80]. Those self-organized structures appear in the Ikeda DDEs with a bandpass filter and asymmetric nonlinear functions. In this thesis, we explored chimeras in one-dimensional virtual space because of single-delay DDEs. However, the study of multi-dimensional chimera systems, i.e. the ones in DDEs with several delay lines, is promising. The main requirement for such systems is that the delays should lie on different timescales, i.e. $\tau_D^{q-1} \ll \tau_D^q$ where q is an index. This study is likely to lead to appearance of new complex structures in the virtual networks. From RC point of view, this may reveal the self-organizing or unsupervised learning potential of those multistable systems, e.g. associative memory which requires presence of many coexisting attractors.

Reservoir computing

Even though reservoir computing (RC) is a relatively new approach, it proved to be effective for machine learning tasks such as nonlinear system identification, prediction, and classification [128]. In this work we have examined the utility of a single-node RC approach based on discrete-time version of the Ikeda DDE. Then, we have successfully implemented a stand-alone real-time RC system in hardware. Now, there are plenty of possible directions to further investigate. The most crucial, in author's opinion, is to develop RC by solving real-world, practical problems.

Alternative methods

While the RC approach is new and not fully studied, this thesis does not claim to cover all the possible aspects related RC. Below are listed several methods that could be investigated, but were omitted in favor of clarity and consistency of the work.

RC. Feedback from the readout

A more general case of RC architecture with a feedback coming from the readout could be also studied:

$$\mathbf{x}(n) = f\left(W^I \cdot \mathbf{u}(n) + W \cdot \mathbf{x}(n-1) + W^F \cdot \mathbf{y}(n-1)\right). \qquad (4.4.1)$$

Here W^I matrix is the input mask and vector $\mathbf{u}(n)$ is the input data vector. Vector $\mathbf{x}(n)$ is a network of “neurons”. The network’s recurrent connections are determined by the matrix W. The matrix W^F is the feedback weights map. Coefficients in W^I, W, and W^F are generated randomly. Usually after the network’s training, the input $\mathbf{u}(n)$ is disconnected, i.e. $\mathbf{u}(n) = 0, n > n_0$. This architecture with the output feedback is particularly useful for time series prediction, see e.g. [129] for more details.

RC. Prediction in a spiking regime

Optimized using a GA, time series prediction with a bandpass model significantly outperforms the model without an integral term. Curiously, the derived bandpass model does not work in a fixed point regime when there is no external input. In fact, its asymptotic regime is spiking (limit cycle) behavior. It is necessary to take into account that Ikeda-like models are known to describe behavior of a neuron with delay [130, 131]; the model with inertia from [132] directly corresponds to the bandpass model. It is worth to investigate how does spiking regime contribute to time series prediction. It would be curious to understand if that is the reason why the bandpass model outperforms the lowpass one.

RC. A DSP implementation

Alternatively, instead of using a numerical integrator, an architecture with explicit digital filtering could realize the delayed-feedback dynamics. In principle, there was no preference which method to use. A chimera state demo using a digitally-filtered architecture was developed on the FPGA, but was not covered in this thesis. In the end, both approaches, explicit digital filtering and numerical integrations, give very close results with the only major difference in the implementation details.

Other tasks: Image recognition

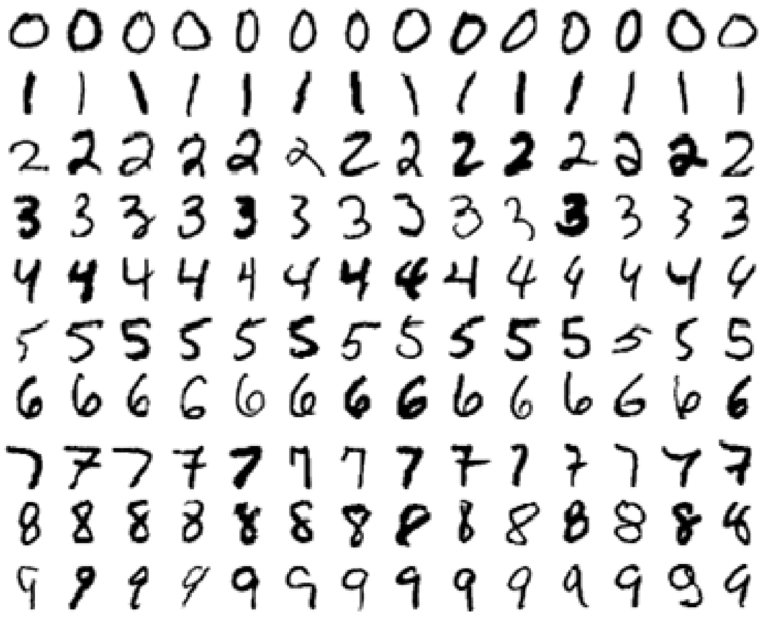

Figure 4.4.1: Handwritten digit examples from the MNIST data set

Another challenging classification problem is image recognition. The MNIST database, containing isolated handwritten digit samples (Fig. (4.4.1)), is often utilized as a benchmark. It is a databank of $m^t = 60,000$ training and $m^v = 10,000$ validation samples. The MNIST database contains 8-bit greyscale images of handwritten digits, so there are ten possible classification outcomes. The size of each image is 28×28 pixels.

In contrast to tasks such as spoken digit recognition, there is no apparent time-dependent information in the input signal. In other words, there are no temporally-encoded information, therefore, we may not benefit from delay dynamics. Thus, the delayed feedback is not used.

The global spatial correlation between the pixels is provided by the input mask, thus, the input images are reshaped as $M = 28 \times 28 = 784$ dimensional vectors. Before the image data are masked, we run a preprocessing scheme to reduce the input dimensions and, in the same time, to compress the data. The motivation of dimensionality reduction is to remove possible linear dependencies in the data. The reasons for compression are technical: (1) to enhance the data transmission rate and (2) to scale down the input mask matrix so that it would occupy less space in an actual RC device.

As dimensionality reduction algorithm, we may employ principal components analysis (PCA) first introduced in [133]. The idea behind the algorithm is to identify the *principal components*, i.e. linear combinations of the input variables that have maximal variance. By analyzing correlations between the input data vectors, PCA is able to determine if multidimensional data fall into a volume with less dimensions. Then, a projection transformation is defined to remove the components with the least variance, and the number of dimensions is effectively reduced. For instance, by identifying 30 components with the maximal variance, the dimensionality of the input image vectors is reduced from $M = 784$ to $M_{pc} = 30$ dimensions.

To perform the preprocessing, a covariance matrix Σ of dimensions $(M \times M)$ is computed from the data set of $m^t = 60,000$ training samples:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} u^{(i)} (u^{(i)})^{\intercal} \tag{4.4.2}$$

where $u^{(i)}(u^{(i)})^{\intercal}$, $i = 1, \ldots m^t$ is the cross-product of the i-th training sample. Then, the eigenvectors matrix U of covariance matrix Σ is computed. The compression matrix $U_{pc} \in \mathbb{R}^{M_{pc} \times M}$ is constructed by taking the first M_{pc} rows of the transposed matrix of eigenvectors U^T. Then, U_{pc} is multiplied by all data vectors $u^{(i)}$, resulting in new compressed samples $u_1^{(i)} = U_{pc} u^{(i)}$ of M_{pc} dimensions. The reverse operation of decompression can be done by multiplication of the transposed compression matrix $(U_{pc})^{\intercal} \in \mathbb{R}^{M \times M_{pc}}$, i.e. $u^{(i)}_{\text{decompressed}} = (U_{pc})^{\intercal} u_1^{(i)}$. During the validation on the test dataset, the matrix U_{pc} that was obtained during training is reused as a part of masking procedure, to perform the preprocessing in exactly same manner as for the training data.

The compressed data do not anymore contain spatial correlation. Thus, to spatially re-correlate the pixels, a new input mask W_1^I is obtained by multiplying the randomly generated sparse mask $W^I \in \mathbb{R}^{N \times M}$ and the decompression matrix $(U_{pc})^{\intercal}$:

$$W_1^I = W^I (U_{pc})^{\intercal}, \tag{4.4.3}$$

where $W_1^I \in \mathbb{R}^{N \times M_{pc}}$ is the new input mask.

In the first numerical experiment, we compare the model with an integral term $\delta \cdot y$ ($\delta > 0$) and the model with $\delta = 0$:

$$\begin{aligned} \tau \dot{x}(t) &= -x(t) - \delta \cdot y(t) + \beta \sin^2 \left(\rho u(t) \right), \\ \dot{y} &= x(t), \end{aligned} \tag{4.4.4}$$

where $\tau = 0.1$, $\beta = 1.1$, $\rho = 10^{-3}$ are optimized with GA. The values of $\delta = 0$ and $\delta = 0.1$ correspond to the classical Ikeda and the Ikeda with a bandpass filter, respectively. In Fig. 4.4.2, the accuracy of RC-based recognition of the two models is illustrated depending on the number of nodes N. Digit error rate (DER) is a fraction of correctly recognized digit images per total number of recognition attempts. First, we see the tendency to saturation, i.e. the accuracy becomes almost constant for $N > 1500$. Second, the low-pass and bandpass models produce practically the same results. This might indicate that this particular task is not memory-sensitive, but only sensitive to the dimensionality. The delayed-feedback and long-term memory (provided by the integral term $\delta \cdot y$) are not required, however increasing N actually improves the result.

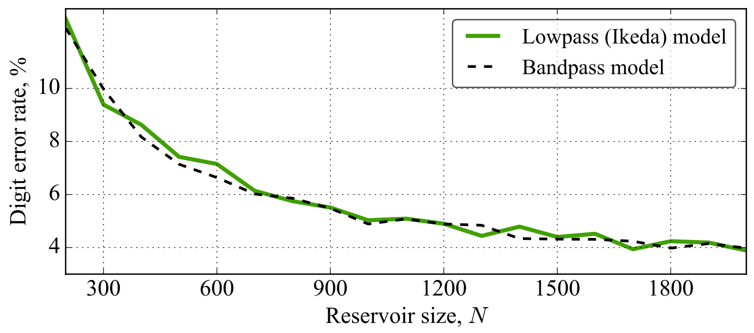

Figure 4.4.2: Digit error rate depending on the reservoir size N. The integral term $\delta \cdot y$ (bandpass model) does not improve the accuracy of handwritten digits recognition.

Such compression algorithms as PCA are called *lossy*, i.e. the information cannot be completely restored after decompression. Consequently, noise is artificially added to data proportionally to the amount of compression. However, it turns out that applying PCA not only decreases the number of dimensions but also enhances the recognition accuracy. That indicates that (1) adding noise to training data may improve recognition accuracy and, more important, (2) the meaningful information falls on a much smaller volume than the original inputs. The images are centered, and therefore the information is encapsulated in $20 \times 20 = 400$ pixels. The large amount of peripheral pixels ($784 - 400 = 384$) remain blank and convey no information at all.

In the second numerical experiment employing PCA, the reservoir's dynamics is given by:

$$\tau \dot{x}(t) = -x(t) + f\left(\rho u(t)\right), \tag{4.4.5}$$

where parameters $\tau = 0.02$, $\rho = 6.4 \cdot 10^{-3}$ are again obtained by the GA. Interestingly, the nonlinearity, rectifier function $f(x) = \max(0, \beta x)$, $\beta = 3.125 \cdot 10^{-2}$, is blindly obtained by the GA! Such type of nonlinearity drastically simplifies practical implementation of digital circuits.

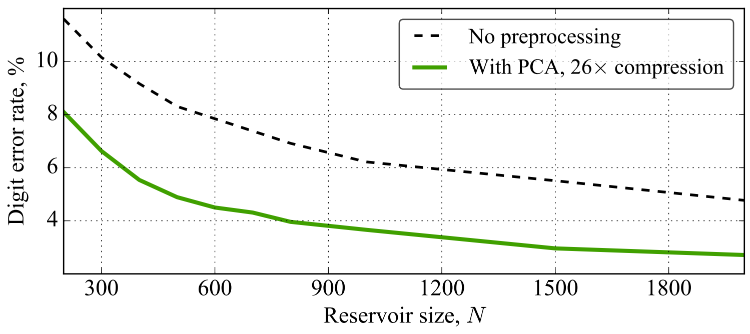

Figure 4.4.3: Handwritten digits recognition improvement achieved by dimensionality reduction (compression) of the MNIST images. In both cases, lowpass-filtered reservoir dynamics is described by Eq. (4.4.5) with $f(x) = \max(0, \beta x)$.

In Fig. 4.4.3, there is a comparison between error rates of applying RC to the original and to the $\times 26$ times compressed MNIST images. As it can be seen, preprocessing by dimensionality reduction significantly improves the classification accuracy (solid green line) comparing to the original, unprocessed inputs (dashed black line). In practice, running the whole MNIST benchmark using RC approach takes mere 17 seconds on a mid-2012 laptop with 2.9GHz Intel Core i7 CPU, including the PCA preprocessing stage, and yields an error rate of 3.96% for $N = 800$ nodes. In contrast, training one epoch of a typical state-of-the art convolutional neural network (CNN) such as Keras[17] results in DER = 2.24%, but it takes 206 seconds (12 times longer) to train and evaluate on the same laptop. A compromise can be achieved with $2,000$ nodes RC that takes around 77 seconds to train and evaluate, still resulting in lower error rates: DER = 1.86%, which highlights the computational efficiency of single-node RC methods.

FPGA implementation

Before FPGA training, MNIST image data are preprocessed (or compressed) on a PC by applying the PCA technique. The immediate advantage of image compression is faster data exchange between the PC and the FPGA. On the FPGA, data is consecutively decompressed and masked. It is achieved in one step by multiplying a modified mask $W_1^I = W^I (U_{pc})^\intercal$, where $W^I \in \mathbb{R}^{N\times 784}$ is randomly generated and $(U_{pc})^\intercal \in \mathbb{R}^{784\times 30}$ is the decompression matrix obtained after the PCA. Therefore, another advantage is

[17] Based on the MNIST example from https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py

$W_1^I \in \mathbb{R}^{N\times 30}$ occupies $784/30 \approx 26$ times less memory inside the FPGA than W^I would.

The reservoir dynamics is realized by an ODE:

$$\tau \dot{x}(t) = -x(t) + f\left(\rho u(t)\right), \tag{4.4.6}$$

with parameters $\tau = 0.02$, $\rho = 6.4 \cdot 10^{-3}$, $\beta = 3.125 \cdot 10^{-2}$. The rectifier nonlinearity f is employed:

$$f(x) = \begin{cases} \beta x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

By fixing the reservoir's size $N = 800$, one is able to evaluate the reservoir's learning capacity. As before, the validation is performed on a separate predefined in MNIST dataset of $m^v = 10,000$ samples. With the maximal training dataset size $m^t = 60,000$, the best FPGA validation error rate of 4.02% is obtained. In the same time, the worst learning error rate is 3.92%. Those rates can be explained. Keeping fixed validation dataset size $m^v = 10,000$, the validation error rate decreases as the number of training samples m^t is increased since the model receives more information about possible configurations. Meanwhile, the learning error rate can only increase as the dataset increases. That is related to the fact that additional data complicate the model, i.e. it becomes more and more difficult to generalize the existing information. These trends are illustrated in Fig. 4.4.4. The training and validation curves are approaching towards each other as the training dataset size m^t increases.

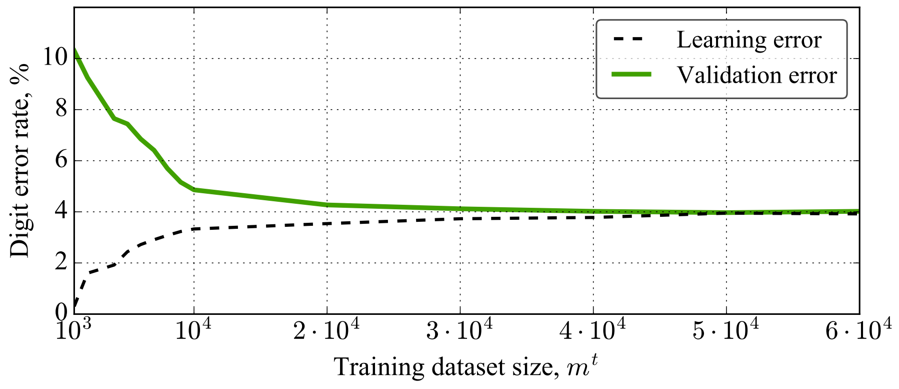

Figure 4.4.4: Learning and cross-validation error curves for FPGA-based RC depending on the training dataset size m^t. The fixed number of nodes is $N = 800$.

At approximately $m^t = 30,000$, the learning and validation error curves come very close, meaning that additional increase dataset size m^t will not improve learning. Therefore, we estimate the learning capacity of the given reservoir to be equal to $30,000$ training samples. The only way to achieve the improvement is increasing the system's complexity. One of the ways to do that is increasing reservoir's size N. This was demonstrated in Figs. 4.4.2 and 4.4.3.

Image recognition conclusions

The image recognition task, in contrast to previous tasks, allowed use of a simplified model: the task did not require neither long-term memory provided by an integral term, nor delayed feedback. Again, as with time-delay based RC, the main advantage was extremely fast learning. Additionally, our work revealed that a preprocessing step of dimensionality reduction actually improves the recognition results by 30-40%. In [134], the digit error rate of 0.92% was achieved using three connected reservoirs, totaling in $48,000$ nodes. Our PC-based result was twice less accurate (1.86%), however only a single-layer reservoir of $2,000$ nodes was employed. Therefore, our implementation is very competitive in terms of computational efficiency with respect to the state of the art in the field of image recognition.

Reservoir computing applications

One yet unsolved practical problem is fuel cells diagnosis, prediction and control. Fuel cell arrays are alternative energy generating devices based on chemical reactions. The main advantage of the devices is ecological sustenance since the only byproducts of energy production are water and heat. Thus, fuel cells have a great potential in every-day life. Their drawback is a limited lifetime. In order to maximize its life span, it is crucial to be able, first to diagnose any possible failure of fuel cell subsystems (such as supply of oxygen, hydrogen, cooling system, etc.) in realtime, and second to be able to predict such failures and the state of the fuel cells in the future. The requirement is a low-power hardware solution. Another viable application is the control of femtosecond lasers, unsolved at the moment task. It involves realtime control of intricate nonlinear dynamics of the system, a job where RC comes in hand.

Our research indicates that those problems can be approached with FPGA-based RC. The main limiting factors are: (1) An ability to create a valid teacher signal required by RC, which is a supervised learning approach. (2) In the case of laser control problem, the input signal is an image, thus it is high-dimensional. Like in the handwritten digits recognition task, the preliminary goal is to reduce the input dimensionality.

From the application view point, further development of the RC system is required. For instance, the FPGA-based RC would greatly benefit from interchangeable external memories such as SD cards to store mask and readout configurations making the hardware more versatile. Possible creation of ASIC[18] RC chip would be a logical step towards industrial applications. This would allow pushing the clock speeds of RC further by an order of magnitude.

Last but not least there is the theoretical view point where it might be appealing to extend RC with other learning techniques, such as reinforcement or unsupervised learning. Alternatively, other learning approaches might be integrated into and/or cooperating with RC.

Functional RC programming

Another opportunity to develop the RC framework is creation of functional reservoir networks. That means, if each of the reservoirs was trained to perform an elementary function, for instance:

$$f(t) = \sin\left[x(t)\right], g(t) = x^2(t), h(t) = x(t-\tau), \tag{4.4.7}$$

then the composition of reservoirs should be able to produce the composition of functions G:

$$g \cdot f \cdot h = \sin^2\left[x(t-\tau)\right]. \tag{4.4.8}$$

Here we used a right to left notation for the functional composition: $f{\cdot}g \equiv f[g(x)]$.

Our preliminary research indicates that such a composition *is* possible. It opens a possibility of programming using functional RC blocks. To benefit from such a situation, instead of elementary mathematical functions, more complex RC blocks such as control, image or sound recognition units can be utilized. That would be one more step towards the development of RC-based AI. The functional approach can be especially effective if implemented in real hardware where all the units can run in parallel.

[18]An application-specific integrated circuit (ASIC), an circuit customized for a particular use, a “hard” chip. In contrast, FPGA is a “soft”, programmable chip.

Digital hardware-oriented improvements

Pseudo-random input mask generation

Digital hardware has limited resources, and it is wise to use them sparingly. One of the biggest memory-hungry components is the input mask W^I. Using a dense matrix encoding, it may require up to 1Mb of memory for the Aurora task. However, the mask is generated randomly, and thus it is one of many possible random configurations. There is no objective reason why this mask should not be generated deterministically. Therefore, all the coefficients of the input mask would be generated on the fly, and thus saving the memory. This idea was first introduced in [107]. However, we would prefer to take best of two worlds — purely random, to get the best results, and deterministic, to save the memory. That means we would like to generate a sequence of *pseudo-random* numbers. For example, employing a so-called *xorshift* routine.

Optimal input mask generation

On the other hand, it is also interesting to maximize the accuracy of the system. GA or any other evolutionary technique can be employed to create the optimal input mask. To preserve the benefits of the previous optimization, deterministic input mask generation, GA should operate on the space of algorithms. The goal would be to find a perfect deterministic algorithm to create an optimal mask that suits one or several problems.

Deep reservoirs

An unexplored horizon of "deep" reservoir learning lies ahead. One may relate this complex network-centric approach to success of *deep learning* techniques in recent years. During the past decade, deep learning proved to be very powerful to solve complex problems [56, 57, 59, 67]. The level of the accuracy allows deep neural networks competing with human experts. Those networks are essentially feed-forward (not recurrent) neural networks consisting of the multiple hidden layers. The effectiveness of such a system is achieved due to (1) abundance of hidden layers that perform detection starting from primitive objects (like edges) and finishing with more conceptual ones (such as living beings) and (2) very large number of training samples.

At the moment little is known about deep reservoir networks, i.e. the ones where multiple reservoirs are the units of a macro-network (e.g. as in [135]). The opportunities of such a construction have not been investigated. A diversity of directions such as network density, size, (non)homogeneity[19], and topology are still left to be explored. However, the main question is: how to design the training signals, which have to back-propagate through hidden reservoir layers?

The idea of deep reservoirs is actively discussed in the RC community[20], however, not so much research was done on that [134, 135]. It means that RC is still in its early origin. Moving further in the RC deep networks direction, a great option is to work in artificial intelligence development. First, is to study the possibility of creation of a high level programming language, where elementary units are reservoirs with some predefined *fuzzy* logic[21] functionality. Thus, programming task would be designing the system based on such elementary blocks. Finally, can reservoir computer play chess?

[19]I.e. if the reservoirs should be identical or differ. If they should differ, then where should be the difference? The input mask characteristics, dynamics, the number of nodes or all of them altogether?

[20]E.g. in talks on Beyond! von Neumann bottleneck workshop, May 18th-21th, 2016

[21]A form of logic in which the truth values may be any number between 0 and 1

Bibliography

[1] Crutchfield, J. P., Ditto, W. L. & Sinha, S. 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**, 037101 (2010).

[2] Indiveri, G. & Liu, S. C. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE* **103**, 1379–1397 (2015). `arXiv:1506.03264v1`.

[3] Enel, P., Procyk, E., Quilodran, R. & Dominey, P. F. Reservoir computing properties of neural dynamics in prefrontal cortex. *PLOS Computational Biology* **12**, e1004967 (2016).

[4] McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* **5**, 115–133 (1943). `arXiv:1011.1669v3`.

[5] Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerves. *J. Physiol.* **117**, 500–544 (1952).

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

[7] Minsky, M. & Papert, S. *Perceptrons: An Introduction to Computational Geometry*, vol. 165 (1969).

[8] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems* **2**, 303–314 (1989).

[9] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics* **43**, 59–69 (1982).

[10] Pineda, F. J. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters* **59**, 2229–2232 (1987).

[11] Maass, W., Natschläger, T. & Markram, H. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput* **14**, 2531–2560 (2002).

[12] Jaeger, H. The “echo state” approach to analysing and training recurrent neural networks. *German National Research Center for Information Technology GMD Technical Report* **148:34** (2001).

[13] Steil, J. Backpropagation-decorrelation: online recurrent learning with o(n) complexity. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, 843–848 (IEEE, 2004).

[14] Appeltant, L. *et al.* Information processing using a single dynamical node as complex system. *Nature Communications* **2**, 466–468 (2011).

[15] Farmer, D. J. Chaotic attractors of an infinite-dimensional dynamical system. *Physica D: Nonlinear Phenomena* **4**, 366–393 (1982).

[16] Erneux, T. *Applied Delay Differential Equations*, vol. 3 (Springer-Verlag, New York, 2009).

[17] Arecchi, F. T., Giacomelli, G., Lapucci, A. & Meucci, R. Two-dimensional representation of a delayed dynamical system. *Physical Review A* **45**, 4225–4228 (1992).

[18] Giacomelli, G., Meucci, R., Politi, A. & Arecchi, F. T. Defects and spacelike properties of delayed dynamical systems. *Physical Review Letters* **73**, 1099–1102 (1994).

[19] Giacomelli, G. & Politi, A. Relationship between delayed and spatially extended dynamical systems. *Physical review letters* **76**, 2686–2689 (1996).

[20] Kuramoto, Y. & Battogtokh, D. Coexistence of coherence and incoherence in nonlocally coupled phase oscillators. *Nonlinear Phenomena in Complex Systems* **5**, 380–385 (2002).

[21] Abrams, D. & Strogatz, S. H. Chimera states for coupled oscillators. *Physical Review Letters* **93**, 174102 (2004).

[22] Panaggio, M. J. & Abrams, D. Chimera states: coexistence of coherence and incoherence in networks of coupled oscillators. *Nonlinearity* **28**, R67–R87 (2015).

[23] Larger, L., Goedgebuer, J.-p. & Udaltsov, V. Ikeda-based nonlinear delayed dynamics for application to secure optical transmission systems using chaos. *Comptes Rendus Physique* **5**, 669–681 (2004).

[24] Larger, L., Lacourt, P.-A., Poinsot, S. & Hanna, M. From flow to map in an experimental high-dimensional electro-optic nonlinear delay oscillator. *Physical Review Letters* **95**, 043903 (2005).

[25] Giacomelli, G., Marino, F., Zaks, M. A. & Yanchuk, S. Coarsening in a bistable system with long-delayed feedback. *EPL (Europhysics Letters)* **99**, 58005 (2012).

[26] Larger, L. Complexity in electro-optic delay dynamics: modelling, design and applications. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **371**, 20120464–20120464 (2013).

[27] Mackey, M. C. & Glass, L. Oscillation and chaos in physiological control systems. *Science (New York, N.Y.)* **197**, 287–9 (1977).

[28] Milton, J. G., Longtin, A., Beuter, A., Mackey, M. C. & Glass, L. Complex dynamics and bifurcations in neurology. *Journal of Theoretical Biology* **138**, 129–147 (1989).

[29] Feng, J., Sevier, S. A., Huang, B., Jia, D. & Levine, H. Modeling delayed processes in biological systems. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* **94**, 1–9 (2016). `1605.07304`.

[30] Chembo, Y. K., Jacquot, M., Dudley, J. & Larger, L. Ikeda-like chaos on a dynamically filtered supercontinuum light source. *Physical Review A* **94**, 023847 (2016).

[31] Paquot, Y. *et al.* Optoelectronic reservoir computing. *Scientific reports* **2** (2012).

[32] Duport, F., Schneider, B., Smerieri, A., Haelterman, M. & Massar, S. All-optical reservoir computing. *Optics Express* **20**, 22783–22795 (2012). `1207.1619`.

[33] Larger, L. *et al.* Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics express* **20**, 3241–9 (2012).

[34] Brunner, D., Soriano, M. C., Mirasso, C. R. & Fischer, I. Parallel photonic information processing. *Nature Communications* **4**, 1364–1367 (2013).

[35] Soriano, M. C. *et al.* Delay-based reservoir computing: Noise effects in a combined analog and digital implementation. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS* **26**, 388–393 (2015).

[36] Hermans, M., Soriano, M. C., Dambre, J., Bienstman, P. & Fischer, I. Photonic delay systems as machine learning implementations. *Journal of Machine Learning Research* 1–22 (2015). `1501.02592`.

[37] Garbin, B., Javaloyes, J., Tissoni, G. & Barland, S. Topological solitons as addressable phase bits in a driven laser. *Nat Commun* **6** (2015).

[38] Romeira, B., Avó, R., Figueiredo, J. M. L., Barland, S. & Javaloyes, J. Regenerative memory in time-delayed neuromorphic photonic resonators. *Scientific Reports* **6**, 19510 (2016). `1503.07781`.

[39] Ikeda, K. Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system. *Optics Communications* **30** (1979).

[40] May, R. M. Simple mathematical models with very complicated dynamics. *Nature* **261**, 459–467 (1976).

[41] Feigenbaum, M. J. Quantitative universality for a class of nonlinear transformations. *Journal of Statistical Physics* **19**, 25–52 (1978).

[42] Sharkovsky, A. N. Coexistence of cycles of a continuous map of the line into itself. *International Journal of Bifurcation and Chaos* **05**, 1263–1273 (English translation) (1995).

[43] Yorke, J. A. & Li, T.-Y. Period three implies chaos. *The American Mathematical Monthly* **82**, 985–992 (1975). `arXiv:1011.1669v3`.

[44] Hoagg, J. B., Lacy, S. L., Babuska, V. & Bernstein, D. S. Sequential multisine excitation signals for system identification of large space structures. *American Control Conference, 2006* 6 pp.– (2006).

[45] Lavrov, R., Jacquot, M. & Larger, L. Nonlocal nonlinear electro-optic phase dynamics demonstrating 10 gb/s chaos communications. *IEEE Journal of Quantum Electronics* **46**, 1430–1435 (2010).

[46] Douglas, R. J. & Martin, K. A. Recurrent neuronal circuits in the neurocortex. *Current Biology* **27**, 496–500 (2004).

[47] Izhikevich, E. M. Simple model of spiking neurons. *IEEE Transactions on Neural Networks* **14**, 1569–1572 (2003). `ArXiv`.

[48] The neuron. URL `https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png`.

[49] Gerstner, W., Kistler, W. M., Naud, R. & Paninski, L. Neuronal dynamics. *Cambridge Univ. Press* 14–17 (2015).

[50] Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks* **4**, 251–257 (1991). `arXiv:1011.1669v3`.

[51] Funahashi, K. & Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks* **6**, 801–806 (1993).

[52] Kilian, J. & Siegelmann, H. T. The dynamic universality of sigmoidal neural networks. *Information and Computation* **128**, 48–56 (1996).

[53] Jaeger, H. A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach. *GMD Report 159, German National Research Center for Information Technology* 1–46 (2002).

[54] Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (MIT Press, 1998).

[55] Nielsen, M. How the backpropagation algorithm works. URL `http://neuralnetworksanddeeplearning.com/chap2.html`.

[56] Russakovsky, O. *et al.* Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* **115**, 211–252 (2015). 1409.0575.

[57] Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

[58] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015). 1312.5602.

[59] Esteva, A. *et al.* Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **542**, 115–118 (2017).

[60] Athale, R. & Psaltis, D. Optical computing, past & future. *Optics and Photonics News* 32–39 (2016).

[61] Dale, M., Miller, J. F. & Stepney, S. *Advances in Unconventional Computing*, vol. 22 of *Emergence, Complexity and Computation* (Springer International Publishing, Cham, 2017).

[62] Pecora, L. M. & Carroll, T. L. Synchronization in chaotic systems. *Physical Review Letters* **64**, 821–824 (1990).

[63] Udaltsov, V. S. *et al.* Bandpass chaotic dynamics of electronic oscillator operating with delayed nonlinear feedback. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **49**, 1006–1009 (2002).

[64] Chembo, Y. K. *et al.* Dynamic instabilities of microwaves generated with optoelectronic oscillators. *Optics Letters* **32**, 2571–2573 (2007).

[65] Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances In Neural Information Processing Systems* 1–9 (2012). 1102.0183.

[66] Taigman, Y., Yang, M., Ranzato, M. & Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1701–1708 (2014). 1501.05703.

[67] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

[68] Paquot, Y., Dambre, J., Schrauwen, B., Haelterman, M. & Massar, S. Reservoir computing: a photonic neural network for information processing. *Proc. SPIE* **7728**, 77280B–77280B–12 (2010).

[69] Barkley, D. & Tuckerman, L. S. Computational study of turbulent laminar patterns in couette flow. *Physical Review Letters* **94**, 1–4 (2005). 0403142v1.

[70] Barkley, D. Modeling the transition to turbulence in shear flows. *Journal of Physics: Conference Series* **318**, 032001 (2011). 1107.3697.

[71] Brethouwer, G., Duguet, Y. & Schlatter, P. Turbulent–laminar coexistence in wall flows with coriolis, buoyancy or lorentz forces. *Journal of Fluid Mechanics* **704**, 137–172 (2012).

[72] Duguet, Y. & Schlatter, P. Oblique laminar-turbulent interfaces in plane shear flows. *Physical Review Letters* **110**, 034502 (2013).

[73] Hagerstrom, A. M. *et al.* Experimental observation of chimeras in coupled-map lattices. *Nature Physics* **8**, 658–661 (2012).

[74] Verschueren, N., Bortolozzo, U., Clerc, M. G. & Residori, S. Spatiotemporal chaotic localized state in liquid crystal light valve experiments with optical feedback. *Physical Review Letters* **110**, 1–5 (2013).

[75] Tinsley, M. R., Nkomo, S. & Showalter, K. Chimera and phase-cluster states in populations of coupled chemical oscillators. *Nature Physics* **8**, 662–665 (2012).

[76] Martens, E. A., Thutupalli, S., Fourrière, A. & Hallatschek, O. Chimera states in mechanical oscillator networks. *Proceedings of the National Academy of Sciences of the United States of America* **110**, 10563–7 (2013).

[77] Kapitaniak, T., Kuzma, P., Wojewoda, J., Czolczynski, K. & Maistrenko, Y. Imperfect chimera states for coupled pendula. *Scientific reports* **4**, 6379 (2014).

[78] Viennot, D. & Aubourg, L. Quantum chimera states. *Physics Letters, Section A: General, Atomic and Solid State Physics* **380**, 678–683 (2016). 1408.4585.

[79] Larger, L., Penkovsky, B. & Maistrenko, Y. Virtual chimera states for delayed-feedback systems. *Physical Review Letters* **111**, 054103 (2013).

[80] Larger, L., Penkovsky, B. & Maistrenko, Y. Laser chimeras as a paradigm for multistable patterns in complex systems. *Nature Communications* **6**, 7752 (2015).

[81] Semenov, V., Zakharova, A., Maistrenko, Y. & Schöll, E. Delayed-feedback chimera states: Forced multiclusters and stochastic resonance. *EPL (Europhysics Letters)* **115**, 10005 (2016). 1511.03634.

[82] Sharkovsky, A. N., Maistrenko, Y. & Romanenko, E. *Difference equations and their applications*, vol. 250 of *Mathematics and Its Applications 250* (Springer Netherlands, Dordrecht, 1993), 1 edn.

[83] Giacomelli, G., Marino, F., Zaks, M. A. & Yanchuk, S. Nucleation in bistable dynamical systems with long delay. *Physical Review E* **88**, 062920 (2013).

[84] Weicker, L. *et al.* Strongly asymmetric square waves in a time-delayed system. *Physical Review E* **86**, 055201 (2012).

[85] Chembo, Y. K., Colet, P., Larger, L. & Gastaud, N. Chaotic breathers in delayed electro-optical systems. *Physical Review Letters* **95**, 203903 (2005).

[86] Chay, T. R. & Rinzel, J. Bursting, beating, and chaos in an excitable membrane model. *Biophysical Journal* **47**, 357–366 (1985).

[87] FitzHugh, R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal* **1**, 445–466 (1961).

[88] Nagumo, J., Arimoto, S. & Yoshizawa, S. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE* **50**, 2061–2070 (1962).

[89] Pancomputationalism or the computational universe theory (2017). URL `https://en.wikipedia.org/wiki/Digital_physics`.

[90] Kari, L. & Rozenberg, G. The many facets of natural computing. *Communications of the ACM* **51**, 72 (2008).

[91] Stepney, S., Abramsky, S., Adamatzky, A., Johnson, C. G. & Timmis, J. Grand challenge 7: Journeys in non-classical computation. *Visions of Computer Science, London, UK* 407–421 (2008).

[92] Gelenbe, E. Natural computation. *Computer Journal* **55**, 848–851 (2012).

[93] Lekitsch, B. *et al.* Blueprint for a microwave trapped ion quantum computer. *Science Advances* **3**, e1601540 (2017). `1508.00420`.

[94] Fernando, C. & Sojakka, S. Pattern recognition in a bucket. In *Proceedings of the 7th European Conference on Artificial Life*, 588–597 (2003).

[95] Larger, L. *et al.* High-speed photonic reservoir computing using a time-delay-based architecture: Million words per second classification. *Physical Review X* **7**, 011015 (2017).

[96] Adamatzky, A. & De Lacy Costello, B. Experimental logical gates in a reaction-diffusion medium: The xor gate and beyond. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* **66**, 1–6 (2002).

[97] Kernel machine. URL `https://upload.wikimedia.org/wikipedia/commons/f/fe/Kernel_Machine.svg`.

[98] Lukoševičius, M. & Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review* **3**, 127–149 (2009).

[99] Scardapane, S. & Wang, D. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **7**, e1200 (2017).

[100] Antonik, P., Smerieri, A., Duport, F., Haelterman, M. & Massar, S. Fpga implementation of reservoir computing with online learning. *Belgian-Dutch Conference on Machine Learning* (2015).

[101] Lin, Y.-p. *et al.* Physical realization of a supervised learning system built with organic memristive synapses. *Scientific Reports* **6**, 31932 (2016).

[102] Keuninckx, L. *Electronic systems as an experimental testbed to study nonlinear delay dynamics*. Phd thesis, Vrije Universiteit Brussel, Brussels, Belgium (2016).

[103] Martinenghi, R. *Démonstration opto-électronique du concept de calculateur neuromorphique par reservoir computing*. Phd thesis, University of Franche-Comte, Besancon, France (2013).

[104] Brunner, D., Soriano, M. C. & Fischer, I. High-speed optical vector and matrix operations using a semiconductor laser. *IEEE Photonics Technology Letters* **25**, 1680–1683 (2013).

[105] Soriano, M. C. *et al.* Optoelectronic reservoir computing: tackling noise-induced performance degradation. *Optics Express* **21**, 12–20 (2013).

[106] Vinckier, Q. *et al.* High-performance photonic reservoir computer based on a coherently driven passive cavity. *Optica* **2**, 438–446 (2015).

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

[108] Thompson, A. An evolved circuit, intrinsic in silicon, entwined with physics. *Evolvable Systems: From Biology to Hardware* **1259**, 390–405 (1996).

[109] Farmer, J. D. & Sidorowich, J. J. Predicting chaotic time series. *Phys. Rev. Lett* **59**, 845–848 (1987).

[110] Slaney, M. Lyon's cochlear model. *Apple Technical Report* 1–79 (1988).

[111] Martinenghi, R., Rybalko, S., Jacquot, M., Chembo, Y. K. & Larger, L. Photonic nonlinear transient computing with multiple-delay wavelength dynamics. *Phys. Rev. Lett* **244101**, 1–4 (2012).

[112] Schrauwen, B., D'Haene, M., Verstraeten, D. & Campenhout, J. V. Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural Networks* **21**, 511–523 (2008).

[113] Vandoorne, K. *et al.* Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature communications* **5**, 3541 (2014).

[114] Langton, C. G. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena* **42**, 12–37 (1990). 9306003.

[115] Schurmann, F., Meier, K. & Schemmel, J. Edge of chaos computation in mixed-mode vlsi - a hard liquid. In Saul, L. K., Weiss, Y. & Bottou, L. (eds.) *Advances in Neural Information Processing Systems 17*, 1201–1208 (MIT Press, 2004).

[116] Bertschinger, N. & Natschläger, T. Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation* **16**, 1413–36 (2004).

[117] Wikipedia. Fpga structure. URL https://upload.wikimedia.org/wikipedia/commons/e/e2/Fpga_structure.svg.

[118] Wikipedia. Fpga cell example. URL http://upload.wikimedia.org/wikipedia/commons/1/1c/FPGA_cell_example.png.

[119] Xilinx power estimator tool. URL https://www.xilinx.com/products/technology/power/xpe.html.

[120] Yousfi-Steiner, N., Moçotéguy, P., Candusso, D. & Hissel, D. A review on polymer electrolyte membrane fuel cell catalyst degradation and starvation issues: Causes, consequences and diagnostic for mitigation. *Journal of Power Sources* **194**, 130–145 (2009).

[121] Gerard, M., Poirot-Crouvezier, J. P., Hissel, D. & Pera, M. C. Oxygen starvation analysis during air feeding faults in pemfc. *International Journal of Hydrogen Energy* **35**, 12295–12307 (2010).

[122] Yousfi Steiner, N., Hissel, D., Moçotéguy, P. & Candusso, D. Diagnosis of polymer electrolyte fuel cells failure modes (flooding & drying out) by neural networks modeling. *International Journal of Hydrogen Energy* **36**, 3067–3075 (2011).

[123] Jouin, M., Gouriveau, R., Hissel, D., Péra, M.-C. & Zerhouni, N. Prognostics and health management of pemfc – state of the art and remaining challenges. *International Journal of Hydrogen Energy* **38**, 15307–15317 (2013).

[124] Jouin, M., Gouriveau, R., Hissel, D., Péra, M. C. & Zerhouni, N. Prognostics of pem fuel cell in a particle filtering framework. *International Journal of Hydrogen Energy* **39**, 481–494 (2014).

[125] Alomar, M. L. *et al.* Digital implementation of a single dynamical node reservoir computer. *IEEE Transactions on Circuits and Systems II: Express Briefs* **62**, 977–981 (2015).

[126] Makridakis, S. Time series prediction: Forecasting the future and understanding the past. *International Journal of Forecasting* **10**, 463–466 (1994).

[127] Jalalvand, A. Connected digit recognition by means of reservoir computing. In *INTERSPEECH 2011* (Florence, Italy, 2011).

[128] Lukoševičius, M. *Reservoir Computing and Self-Organized Neural Hierarchies.* Phd thesis, Jacobs University Bremen, Bremen, Germany (2011).

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

[130] Gyori, I. & Hartung, F. Stability analysis of a single neuron model with delay. *Journal of Computational and Applied Mathematics* **157**, 73–92 (2003).

[131] Maisnam, S. & Singh, R. K. B. Generalization of neuron network model with delay feedback **1**, 1–12 (2015). `1507.04552`.

[132] Li, C., Chen, G., Liao, X. & Yu, J. Hopf bifurcation and chaos in a single inertial neuron model. *Eur. Phys. J. B* **41**, 337–343 (2004). `0411027`.

[133] Pearson, K. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2**, 559–572 (1901).

[134] Jalalvand, A., Wallendael, G. V. & Walle, R. V. D. Real-time reservoir computing network-based systems for detection tasks on visual contents. *7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN)* 146–151 (2015).

[135] Keuninckx, L., Danckaert, J. & Van der Sande, G. Real-time audio processing with a cascade of discrete-time delay line-based reservoir computers. *Cognitive Computation* (2017).

All the listed URLs were accessible in December 2016.

Appendices

Appendix A: detailed results of isolated spoken digit recognition task applying RC to Aurora-2 benchmark using a bandpass model

	Clean 1	Clean 2	Clean 3	Clean4
Test A	2.0	2.5	4.2	4.9
Test C	5.4	7.5	-	-

(a) 400 virtual neurons. Learning WER is 1.9%

	Clean 1	Clean 2	Clean 3	Clean4
Test A	0.3	1	1	2.5
Test C	3.4	3.9	-	-

(b) 1000 virtual neurons. Learning WER is 0.7%

	Clean 1	Clean 2	Clean 3	Clean4
Test A	0	1.8	0.7	1.8
Test C	3.4	3.2	-	-

(c) 1500 virtual neurons. Learning WER is 0.4%

Table 4.2: Word error rates after reservoir validation. Training on clean data.

SNR/dB	Subway	Babble	Car	Exhibition	Average
Clean	6	6	7.4	7	6.6
20dB	6.4	9.3	12.4	9.5	9.4
15dB	8.4	12.5	13.8	14.437	12.3
10dB	14.8	22.6	23	20.8	20.3
5dB	25.2	37.3	41.7	41.9	36.4
0dB	60	66.3	69.6	71.1	66.8
-5dB	85.2	84.9	88	82	85

(a) Test A, training on both clean and noisy data

SNR/dB	Restaurant	Street	Airport	Train station	Average
Clean	6	6.1	7.4	7	6.6
20dB	8	21.9	15.5	15.5	15.2
15dB	11.4	28.7	19	26.4	21.4
10dB	17.1	50.2	29.3	44.7	35.3
5dB	36.2	72.8	45.9	58.8	53.4
0dB	66.4	86	71	76	74.9
-5dB	78.9	90.3	84.5	86.3	85

(b) Test B, training on both clean and noisy data

SNR/dB	Subway (MIRS)	Street (MIRS)	Average
Clean	8.7	9	8.8
20dB	18.1	30.8	24.5
15dB	22.8	46.6	34.7
10dB	46.3	74.2	60.3
5dB	68.1	87.8	78
0dB	79.2	89.6	84.4
-5dB	85.9	91.4	88.7

(c) Test C, training on both clean and noisy data

Table 4.3: WERs (%) for a reservoir of 400 virtual neurons. Learning WER is 12.8%.

SNR/dB	Subway	Babble	Car	Exhibition	Average
Clean	1.7	2.5	2.8	5.6	3.2
20dB	4.4	5.4	4.6	6.3	5.2
15dB	6	7.9	6	8.8	7.1
10dB	10.1	7.9	12.7	12.7	10.8
5dB	16.1	22.9	30.4	26.1	24
0dB	44.6	50.5	65.4	62.7	55.9
-5dB	76.2	83.2	87.3	79.2	81.5

(a) Test A, training on both clean and noisy data

SNR/dB	Restaurant	Street	Airport	Train station	Average
Clean	1.7	2.5	2.8	5.6	3.2
20dB	5.4	10.8	5.7	7.7	7.4
15dB	7.7	16.5	8.8	11.6	11.2
10dB	8.7	34.8	16.6	27.5	21.9
5dB	28.9	64.9	38.5	48.9	45.4
0dB	66.1	81	67.8	69.4	71.1
-5dB	80.2	89.6	82.3	84.9	84.2

(b) Test B, training on both clean and noisy data

SNR/dB	Subway (MIRS)	Street (MIRS)	Average
Clean	4.7	6	5.4
20dB	10.4	16.8	13.6
15dB	14.1	34.8	24.4
10dB	31.9	68.5	50.2
5dB	66.8	83.9	75.3
0dB	79.9	88.5	84.2
-5dB	84.2	91.4	87.8

(c) Test C, training on both clean and noisy data

Table 4.4: WERs (%) for a reservoir of 1000 virtual neurons. Learning WER is 5.9%.

SNR/dB	Subway	Babble	Car	Exhibition	Average
Clean	2	2.5	1.4	3.9	2.5
20dB	4	3.9	4.2	5.6	4.4
15dB	4	5.7	5.7	8	5.85
10dB	8.7	6.8	9.2	8.8	8.4
5dB	12	19.4	26.5	21.1	19.8
0dB	39.9	45.9	65.7	57	52.1
-5dB	75	81.4	87.3	77.8	80.3

(a) Test A, training on both clean and noisy data

SNR/dB	Restaurant	Street	Airport	Train station	Average
Clean	2	2.5	1.4	3.9	2.5
20dB	4.7	9	4.6	6	6
15dB	6	13.6	6.7	8.5	8.7
10dB	4.7	27.6	13	21.8	16.8
5dB	19.5	58.1	32.2	41.5	37.8
0dB	58.4	79.6	65.4	70.4	68.4
-5dB	79.5	89.2	83	87.7	84.9

(b) Test B, training on both clean and noisy data

SNR/dB	Subway (MIRS)	Street (MIRS)	Average
Clean	3	4.3	3.6
20dB	8.4	10.8	9.6
15dB	11.7	28.3	20
10dB	17.5	58.4	37.9
5dB	44	81.7	62.8
0dB	68.5	87.8	78.1
-5dB	79.5	91.4	85.5

(c) Test C, training on both clean and noisy data

Table 4.5: WERs (%) for a reservoir of 1500 virtual nodes. Learning WER is 4%.

Appendix B: comparison between lowpass dynamics implementations on FPGA and PC for Aurora benchmark

SNR/dB	Subway	Babble	Car	Exhibition	Average
Clean	1.3	3.6	2.8	5.3	3.3
20dB	3.4	5.4	4.2	7.4	5.1
15dB	5.7	6.1	6	10.9	7.2
10dB	9.4	10.8	11.7	15.5	11.8
5dB	15.8	25.1	33.9	28.5	25.8
0dB	46.3	50.5	71.7	61.6	57.5
-5dB	77.2	81.7	88.3	79.9	81.8

(a) Test A, training on both clean and noisy data

SNR/dB	Restaurant	Street	Airport	Train station	Average
Clean	1.3	3.6	2.8	5.3	3.3
20dB	4	9	4.6	7.4	6.2
15dB	4.7	16.5	8.8	12.3	10.6
10dB	8	33.7	17	29.2	22
5dB	20.8	64.9	36.4	52.5	43.6
0dB	50.3	81	63.6	76	67.7
-5dB	79.5	90	84.1	84.9	84.6

(b) Test B, training on both clean and noisy data

SNR/dB	Subway (MIRS)	Street (MIRS)	Average
Clean	5	7.9	6.5
20dB	12.1	19	15.5
15dB	17.8	35.5	26.6
10dB	33.6	60.6	47.1
5dB	57.4	83.5	70.4
0dB	75.2	90.7	82.9
-5dB	81.9	91.4	86.6

(c) Test C, training on both clean and noisy data

Table 4.6: Aurora benchmark WERs (%). Computer implementation of a reservoir of 1000 virtual neurons. A lowpass Ikeda model (3.4.8) is utilized as a reservoir. Parameters $\varepsilon = 1.56 \cdot 10^{-1}$, $\beta = 1.11$, $\Phi_0 = -3.02$, $\rho = 1.15$ are obtained by GA. Learning WER is 5.7%.

SNR/dB	Subway	Babble	Car	Exhibition	Average
Clean	2	4.3	3.2	6	3.9
20dB	3.7	6.8	3.9	9.2	5.9
15dB	6.7	7.2	7.1	10.9	8
10dB	9.4	11.1	11.7	16.2	12.1
5dB	17.1	25.8	32.2	28.5	25.9
0dB	45	52.3	73.9	63.4	58.6
-5dB	78.2	82.1	87.6	77.5	81.3

(a) Test A, training on both clean and noisy data

SNR/dB	Restaurant	Street	Airport	Train station	Average
Clean	2	4.3	3.2	6	3.9
20dB	4	10.4	4.9	7.7	6.8
15dB	6.4	17.2	11	13.4	12
10dB	9.4	33.7	19.1	27.8	22.5
5dB	20.8	65.2	36.7	52.8	43.9
0dB	54.7	81.7	64.3	76.1	69.2
-5dB	79.5	90.7	85.2	86.3	85.41

(b) Test B, training on both clean and noisy data

SNR/dB	Subway (MIRS)	Street (MIRS)	Average
Clean	5.7	7.5	6.62
20dB	11.7	20.8	16.3
15dB	18.8	35.8	27.3
10dB	35.2	60.2	47.7
5dB	60.1	84.2	72.2
0dB	77.5	90	83.7
-5dB	84.6	91.8	88.2

(c) Test C, training on both clean and noisy data

Table 4.7: Aurora benchmark WERs (%). FPGA implementation of a reservoir of 1000 virtual neurons. A lowpass Ikeda model (3.4.8) is utilized as a reservoir. Parameters $\varepsilon = 1.56 \cdot 10^{-1}$, $\beta = 1.11$, $\Phi_0 = -3.02$, $\rho = 1.15$ are obtained by GA. Precision of arithmetic operations is $1.22 \cdot 10^{-4}$. Learning WER is 6%.

Abstract

The thesis develops a novel approach to design of a *reservoir computer*, one of the challenges of modern Science and Technology. It consists of two parts, both connected by the correspondence between optoelectronic delayed-feedback systems and spatio-temporal nonlinear dynamics. In the first part (Chapters 1 and 2), this correspondence is used in a fundamental perspective, studying self-organized patterns known as *chimera states*, discovered for the first time in purely temporal systems. Study of chimera states may shed light on mechanisms occurring in many structurally similar high-dimensional systems such as neural systems or power grids. In the second part (Chapters 3 and 4), the same spatio-temporal analogy is exploited from an applied perspective, designing and implementing a brain-inspired information processing device: a real-time digital *reservoir computer* is constructed in FPGA hardware. The implementation utilizes delay dynamics and realizes input as well as output layers for an autonomous cognitive computing system.

Keywords

Reservoir computing, Machine learning, Nonlinear delay dynamics,
Complex systems, Chimera states, FPGA

Résumé

Cette thèse développe une nouvelle approche pour la conception d'un *reservoir computer*, l'un des défis de la science et de la technologie modernes. La thèse se compose de deux parties, toutes deux s'appuyant sur l'analogie entre les systèmes optoelectroniques à retard et les dynamiques spatio-temporelles non linéaires. Dans la première partie (Chapitres 1 et 2) cette analogie est utilisée dans une perspective fondamentale afin d'étudier les formes auto-organisées connues sous le nom *d'états Chimère*, mis en évidence une première fois comme une conséquence de ces travaux. Dans la deuxième partie (Chapitres 3 et 4) la même analogie est exploitée dans une perspective appliquée afin de concevoir et mettre en oeuvre un concept de traitement de l'information inspiré par le cerveau: un *reservoir computer* fonctionnant en temps réel est construit dans une puce FPGA, grâce à la mise en oeuvre d'une dynamique à retard et de ses couches d'entrée et de sortie, pour obtenir un système traitement d'information autonome intelligent.

Mots-clés

Reservoir computing, Apprentissage automatique, Systèmes complexes,
Dynamique non linéaire à retard, États Chimère, FPGA