Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: **«КЛАСИФІКАЦІЯ ПАТЕРНІВ В РЕЗЕРВУАРНИХ ОБЧИСЛЕННЯХ ЗА ДОПОМОГОЮ ПОКАЗНИКІВ ДИНАМІЧНОГО СТАНУ»**

Виконала: студентка 2-го року навчання

освітньо-наукової програми «Системний аналіз»,

спеціальності 124 Системний аналіз

Ровнік Вероніка Костянтинівна

Керівник: Швай Н. О.,

кандидат фіз.-мат. наук, доцент

Рецензент: Шаповал Н.В

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 20____ р.

Київ – 2022

№ з/п	*ПЕРЕЛІК РОБІТ*	*Термін* виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	жовтень 2020р.	10.10.2020		
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	листопад 2020р. - квітень 2021р.	01.11.2021		
3.	Складання плану кваліф. роботи та узгодження з науковим керівником	червень 2021р.	01.06.2021		
4.	Написання розділів роботи, розробка чисельних симуляцій для аналізу динаміки обраної математичної моделі та її спроможності в завданні класифікації	липень - жовтень 2021р.	01.07.2021		
5.	Проміжний контроль виконання роботи	липень 2021р.	23.07.2021		
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	вересень 2021р. - грудень 2022р.	01.10.2021		
	Розділ 1: Вступ (постановка проблеми, теоретичні основи, огляд літературних джерел)	грудень 2021р.	14.12.2021		
	Розділ 2: Методи (аналітично-дослідницька частина: опис та аналіз виконаних кроків на шляху до побудови динамічної системи, що класифікує зображення)	вересень-жовтень 2021р.	05.10.2021		
	Розділ 3: Результати (проектно-рекомендаційна частина)	листопад 2021р.	12.11.2021		
	Розділ 4: Висновки та дискусія (підсумок результатів та огляд запропонованих подальших кроків для дослідження)	листопад 2021р.	12.11.2021		
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	березень 2022р.	20.03.2022		
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	30 червня 2022р.	30.06.2022		
9.	Подання на зовнішню рецензію	середина червня	26.06.2022		
10.	Підготовка до захисту кваліфікаційної роботи на засіданні кафедри: написання доповіді та виготовлення ілюстративного матеріалу	**до 17 червня 2022р.**	14.06.2022		
11.	Попередній захист кваліфікаційної роботи на засіданні кафедри	**17 червня 2022р.**	17.06.2022		

12.	Подання кваліфікаційної роботи на кафедру з усіма супроводжувальними документами	**до 30 червня 2022р.**	30.06.2022		
13.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	8 липня 2022р.			

Графік узгоджено «17» жовтня 2021р.
Науковий керівник Швай Надія Олександрівна

Виконавець кваліфікаційної роботи Ровнік Вероніка Костянтинівна
ПРИМІТКА: Завдання на магістерську роботу та графік підготовки магістерської роботи до захисту друкуються на одному аркуші, на двох сторінках.

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики
Освітній ступінь магістр

Напрям підготовки “Системний аналіз”

Спеціальність Системний аналіз, 124

ЗАТВЕРДЖУЮ
Завідувач кафедри Олійник Богдана Віталіївна
“____” _____2022 року

ЗАВДАННЯ

ДЛЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Ровнік Вероніці Костянтинівні

1. Тема роботи: “Класифікація патернів в резервуарних обчисленнях за допомогою показників динамічного стану” / “Pattern classification in reservoir computing with dynamical state measures”

керівник роботи Швай Надія Олександрівна, старший викладач, кандидат фізико-математичних наук

затверджені наказом вищого навчального закладу від
«__»_____20__ року №___

2. Строк подання студентом роботи

30 червня 2022р.

3. План роботи:

1. Ознайомитись з основними джерелами за напрямком роботи та результатами в даній області дослідження.

2. Спроектувати ескіз майбутніх досліджень.

3. Розробити програмне забезпечення мовою Python для дослідження динаміки системи за двох основних умов: відсутності збурення зовнішнім входом та присутності вхідного сигналу.

4. Проаналізувати спроможність системи до вирішення задачі класифікації зображень та, порівнюючи результати в процесі варіації параметрів динамічної системи, знайти їх оптимальні значення відносно точності класифікації.

5. Розробити текстову частину магістерської роботи: почати з опису запропонованих методів, навести основні результати, завершити написанням вступу, в якому задається контекст даної роботи та актуальність власного досліджень відносно тих, що вже існують в напрямку резервуарних досліджень, рекурентних нейронних мереж та класифікації зображень.

Abstract

Reservoir computing is a brain-inspired paradigm for RNN training that has been successfully applied to different computational problems involving sequential data. One of such problems is *temporal pattern classification*, where time series patterns are to be assigned to a specific category. In neuroscience, the pattern recognition is known as a function of the brain important for a range of intelligent behaviors: hearing, speech, vision, music, and motor control.

Here we present a reservoir network architecture with the phase oscillator models as neurons that solves the given task. The proposed computational scheme consists of data encoding, reservoir dynamics tuning, dynamical state decoding, and readout training steps.

At the data encoding stage we solve the task of presenting the static data to the network as a temporal signal. Next, we find suitable network parameter regimes using dynamical systems theory. At the data decoding stage we extract information about the dynamical state of the network of oscillators, i.e., *frequency synchronization* emerging given the input. Finally, the readout layer is trained to linearly separate the high-dimensional states of the reservoir with accordance to the pattern recognition task.

We evaluate the computational performance of the reservoir architectures by solving an image classification task on the well-known MNIST digits recognition data set.

The designed reservoir computing scheme shows decent performance in the image classification task. We assume that the proposed approach generalizes to other neuron models that exhibit similar properties of intrinsic dynamics. We also presume the possibility of direct mapping between the theoretical model of the reservoirs and their physical implementation in hardware. Due to the considerably reduced number of trainable parameters, the approach promises to be more competitive in power efficiency than recurrent neural networks trained with the conventional back-propagation through time algorithm.

Contents

1 Introduction **1**

1.1 Artificial neural networks 1
1.2 Feed-forward neural networks 2
1.3 Recurrent neural networks 3
1.4 Reservoir computing 5
1.5 Computing by means of oscillator dynamics 8
1.5.1 Active rotator model 9
1.5.2 Kuramoto with inertia model / pendulum equation 10
1.6 Classification task with RC 11
1.7 Aim of this work 12

2 Methods **14**

2.1 Encoding of data 14
2.2 Dynamics of reservoir 17
2.3 Decoding reservoir state 19
2.3.1 Kuramoto order parameter 19

2.3.2 Frequency clustering 21
2.4 Readout training 22
2.4.1 Task specification 22
2.4.2 Model selection and fitting 23
2.4.2.1 Softmax regression 23
2.4.2.2 Random forest classifier 23

3 Results **25**
3.1 Active rotator reservoir 26
3.1.1 Fixed network size 26
3.1.1.1 Input length effect 27
3.1.2 Variable network size 27
3.2 Pendulum reservoir 30
3.2.1 Fixed network size 30
3.2.2 Decoding approaches comparison 30
3.3 Reservoir comparison 31

4 Conclusions and discussion **32**

5 Appendix **34**
5.1 Simulations parameters 34
5.1.1 Neuron models 35
5.1.1.1 Active rotator 35
5.1.1.2 Pendulum 36
5.1.2 (Hyper) parameters of classifier models 36
5.1.2.1 Softmax regression 37
5.1.2.2 Random forest 37
5.2 Algorithms 38

5.2.1 Network connectivity modification 38
5.3 State decoding: visualization 39
5.3.1 Pendulum model 39

References **41**

1

Introduction

1.1 Artificial neural networks

Artifical neural networks (ANNs) are highly abstracted networks that consist of simple "artificial neurons". Connections between neurons of ANNs are trained, which is in a way inspired by synaptic rewiring (plasticity) in the brain (1). In general form, connections in the networks are weighted and directed. The state of each artificial neuron is called *activation*. While the models of neurons may be relatively simple, information processing happens due to interactions within a network: activation of each neuron affects activations of other neurons to which it is synaptically connected.
According to differences in structure, complexity, and computational problems they are expected to solve, most generally, ANNs are divided into two major classes: *feed-forward neural networks* and *recurrent neural networks*.

1.2 Feed-forward neural networks

A *feed-forward neural network* (FFNN), also known as a multilayer perceptron (MLP) is the class of an artificial neural network that is structured in the following way:

1. An input layer, where the information enters the network (**Figure 1.1: A**).
2. A group of hidden layers – a primary part of the network, which performs most of the information processing (**Figure 1.1: B**).
3. An output layer that generates a prediction according to the task (**Figure 1.1: C**).

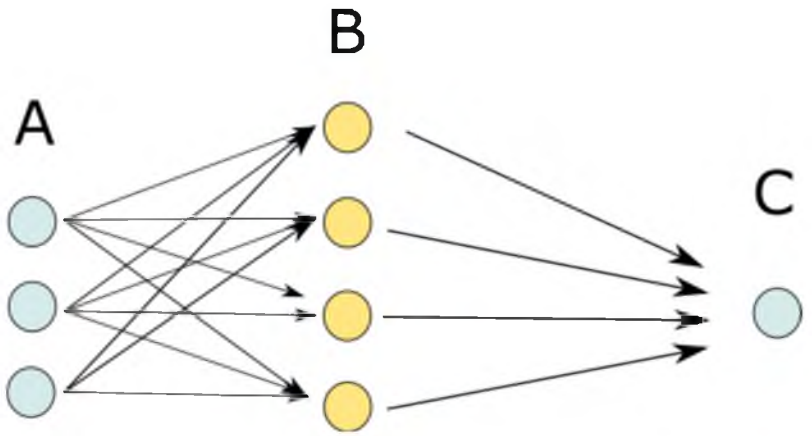

Figure 1.1: Schematic of feedforward neural network's architecture

The information flow is directed from the left to the right in such a network, namely, "forward". The achitecture does not contain any intra-layer connections or recurrences. Hence, there is no self-state dependence in the network: the activation of a node depends

only on the activation of the previous layer. In the absence of input, the system reaches its resting state within a one-time step for every layer.
Examples of application domain are speech recognition (2), natural language understanding (3) computational biology (4), and image recognition (5).
Feed-forward neural networks are trained with *back-propagation* (6). The measure of success of the training can be defined as an ability to correctly classify previously unseen data, which is also known as *generalization property.* For this, training data must be sufficiently diverse and capture essential features for the whole input data space. A feed-forward network becomes more powerful with more training data, and more hidden layers (7). Moreover, multiple FFNNs can be stacked into a deep neural network. The idea of deep learning lies within an attempt to mimic the architecture similar to the one in the visual cortex (8, 9), where, it is believed, different levels learn features or representations at increasing levels of abstraction (10).

1.3 Recurrent neural networks

Recurrent synaptic connections are a characteristic property of neural networks in the brain. According to Buzsaki (11), "the brain is essentially a multitude of superimposed and ever-growing loops between the input from the environment and the brain's outputs." *Recurrent neural networks* (RNNs) follow this biological principle and extend the application of feed-forward networks to sequential/temporal problems by incorporating a temporal dimension in their architecture. RNNs have proved to solve problems involving time series, e.g., video prediction (12), gesture and speech recognition (13), and neural machine translation (14).
The architecture of a recurrent neural network differs fundamentally from that of a feed-forward neural network due to cycles in its computational graphs. Cycles in the connectivity represent the influence of the present value of a node on its own value at a future time step (6). This contrasts to FFNNs, where the information processing flow goes from

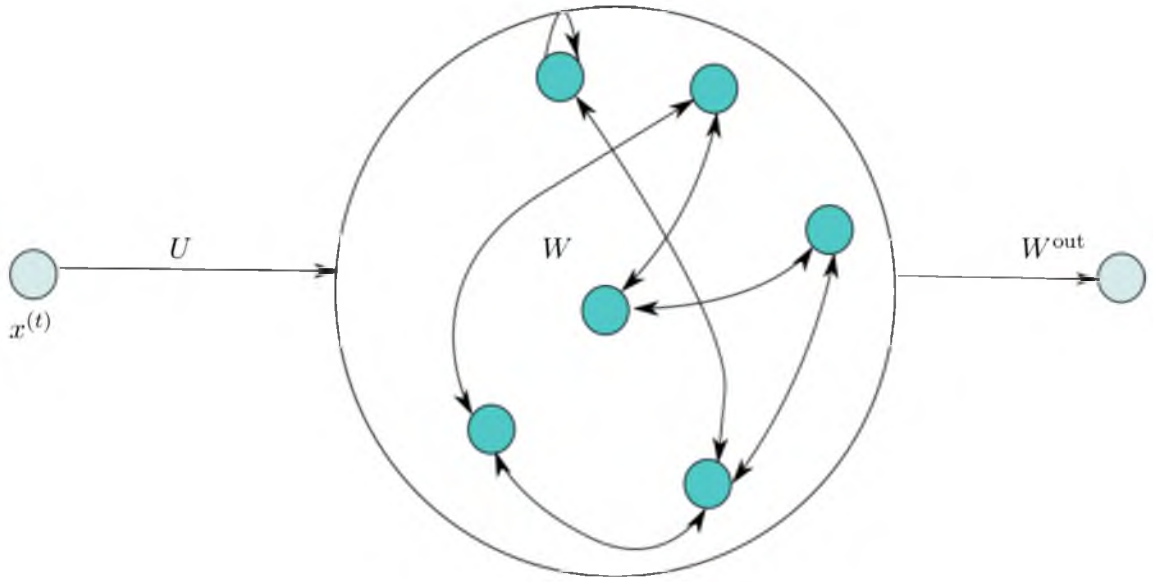

Figure 1.2: Schematic of recurrent neural network's architecture

left to right. The basic RNN architecture is illustrated in Figure 1.2.
The recurrent network processes information given by the input by incorporating it into the network's state h passed forward through time (6). The equation of the network's state at time t can be therefore defined as follows:

$$h^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{1.1}$$

where $h^{(t-1)}$ is the state of the network at the previous time step $(t-1)$, b is a vector of bias weights, W is the matrix of recurrent weights, U is the matrix of input weights, $x^{(t)}$ is the input at the current time step t.

The output of the network is then calculated as

$$y^{(t)} = W^{\text{out}} h^{(t)} \tag{1.2}$$

where W^{out} is the matrix of output weights. It has been shown in the literature that Equation 1.2 is universal: any function computable by a Turing machine can be computed by such a recurrent network of a finite size (15). The recurrent weights need to be trained to solve a given task, which means optimizing matrix W. The standard technique for learning is *back-propagation* (6) – it is applied to the unrolled computational graph of the recurrent network and therefore is called *back-propagation through time* or *BPTT*. It is usually implemented through stochastic gradient descent (16). However, this algorithm is implausible from a biological perspective: error signals are transmitted backwards in time and in space, i.e., from post- to presynaptic neurons (16). Moreover, training is accompanied by high computational costs. The runtime of BPTT is $O(T)$, where T is the number of time steps. Furthermore, the runtime can not be decreased via parallelization due to the sequential nature of the forward propagation: each time step can only be computed after the previous one. The memory cost for storing states is also $O(T)$ (6).
Another challenge that may impede the optimization of an RNN is *learning of long-term dependencies*: gradients propagated over many time steps may vanish or explode. This effect can be explained by the fact that the composition of the same function in the network is evaluated multiple times, once per time step, which consists of the multiplication of many Jacobians and results in a highly nonlinear behavior (6).
The known limitations of recurrent networks training led to the advancement of new research directions aiming to overcome those. One of them is *reservoir computing*.

1.4 Reservoir computing

The concept of the *reservoir computing* (RC) framework was introduced independently by Maass et al. in (17) and Jaeger in (18), where the proposed artificial neural network

architectures were named *echo state networks* and *liquid state machines* respectively. RC suggests a biology-inspired alternative to back-propagation through time for learning in RNNs. It has been studied that the cerebellum has properties similar to a reservoir computer (19). RC has been successfully applied to the problems of robotic motion, image recognition (20), and nonlinear dynamics – reservoirs can emulate the behavior of complex dynamical attractors. Another beneficial aspect of reservoir computing is the plausibility of its physical implementation (21, 22).

The general principle of reservoir computing can be condensed to the following. The recurrent network is a fixed nonlinear dynamical system called a *reservoir*; it aims to capture different aspects of the history of inputs. The recurrent and input connections in the network are generated randomly and remain unchanged during the entire evolution of the dynamical system. Only the output (readout) weights are trained. A *readout* of the network is a (linear) model trained on the reservoir states via least squares or gradient descent methods according to the given task.

To summarize, RC aims to solve linearly non-separable problems via the nonlinear mapping of the input signal into a higher dimensional space. Figure 1.3 illustrates the basic architecture of a reservoir computer.

The fundamental challenge in reservoir computing is two-fold: choosing an optimal dynamical system and devising a way to set the input and recurrent connections so that the rich history of inputs is captured in the spatiotemporal dynamics of the network, which can then be transformed by the readout into the solution to the task. The most common approach to the latter problem is to generate recurrent weights at random. The probability distribution of weights can then be adapted according to the specific task. A more universal approach would be to *optimize the reservoir dynamics for the expected range of tasks.* One of the studied dynamical regimes that has been proved to be suitable for computation is a so-called *Edge of Chaos* (23).

There are also certain quantifiable properties (24) that have been studied in order to evaluate if a given dynamical system can be exploited as a reservoir, e.g., *echo state property*, *separation property*, *approximation property*, *kernel quality*, *memory capacity* and *nonlin-*

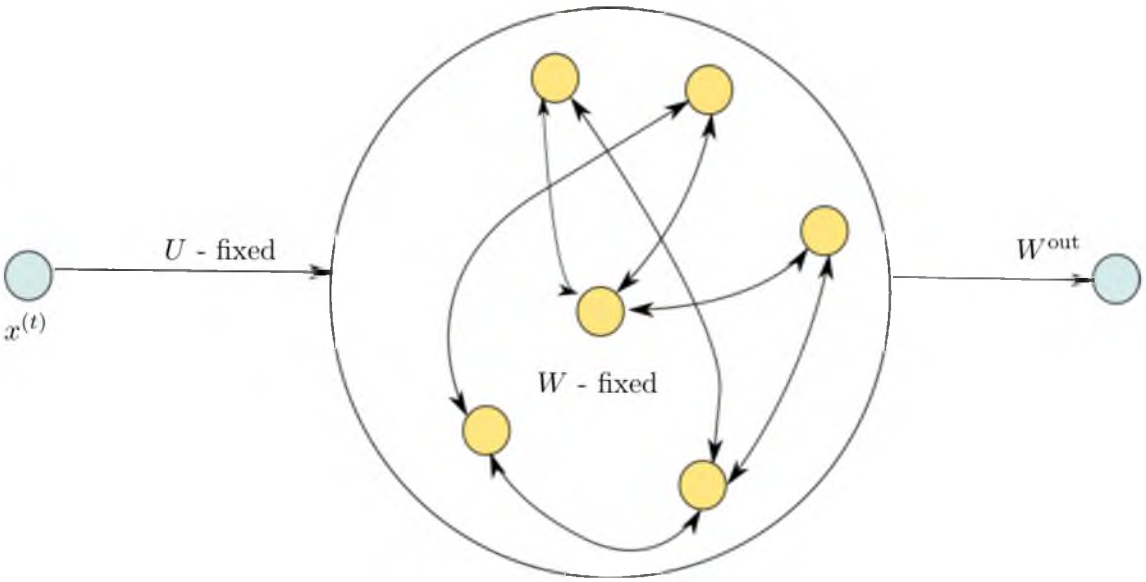

Figure 1.3: Schematic of reservoir computer architecture

earity. For instance, in (25) and (26), the authors established the relationship between the nonlinearity of reservoir dynamics and its memory capacity, which was accordingly formulated as *Memory-Nonlinearity Tradeoff.*

1.5 Computing by means of oscillator dynamics

Coupled oscillators are complex dynamical systems, which have long been studied in physics and mathematics with the goal to model various biological processes (27). Furthermore, networks of coupled oscillators found their place in the theory of computation, e.g., in the collective state computing model (28). According to this paradigm, *computation* is viewed as a result of complex nonlinear interactions in networks of interconnected units. Such computational units can be implemented by oscillators and loosely interpreted as neurons. The phase and frequency of oscillators are then considered a representation of the system's input processed in the network. Encoding information in the phase was first suggested by J. von Neumann in the patent of the device that implied using oscillators as logic gates (29).

The result of a computation in the network of oscillators is commonly expressed in patterns in the phase and/or frequency domain. These often take form of the synchronization of the oscillators into an attractor in the state space, such as a limit cycle or an equilibrium. Since the networks of oscillators are usually high-dimensional nonlinear systems with memory (30), one can exploit the complexity of the emerging dynamical patterns for information processing in reservoir computing. An output layer of the reservoir network can then transform the network dynamics into the desired computational result (28).

One of the examples of the successful usage of coupled oscillators in reservoir computing is the work by (31), where the authors solved function approximation and regression problems with the synchronization-based phase oscillator reservoir. In (32), a reservoir based on nonlinear mechanical oscillators was proposed, which performance was evaluated in a parity of a bit stream and a spoken digit recognition tasks. In (33), the authors designed

a DNA reservoir with coupled chemical oscillators and applied it to a temporal signal-tracking task.
In Methods, chapter 2, we propose to use the phase oscillators as building blocks of the reservoir architecture. The phase dynamics in the reservoirs are governed by the *active rotator* and *pendulum* models with the sinusoidal (Kuramoto) coupling. To justify the choice of each model's parameters for the reservoir, we first summarize the existing analytical and numerical studies of their dynamics and refer the findings to our selection of features suitable for computation.

1.5.1 Active rotator model

The *active rotator* model was first studied by S. Shinomoto and Y. Kuramoto in (34). The active rotator consists of a system moving around the unit circle whose phase θ satisfies

$$\dot{\theta} = \omega - r \sin\theta \quad \text{with} \quad \theta \in S^1 \tag{1.3}$$

where ω is an individual frequency of the oscillator, which can also represent the external stimulation; $\Theta(t, \theta(t_0))$ is the state of the active rotator at time t given the initial condition $\theta(t_0)$.
The active rotator is an elementary model of an *excitable-oscillating* system. By E.M. Izhikevich (35), a neuron is *excitable* if its state is near a bifurcation from resting to spiking activity. This property makes the active rotator model similar to the *theta-neuron* studied in (36) as a canonical model for type I membranes. The results of the study showed that θ rotates around the unit circle when $|\frac{\omega}{r}| > 1$, exhibiting periodic oscillations. When $|\frac{\omega}{r}| < 1$, there is a pair of equilibrium phases on the unit circle, the locally stable and unstable ones. Rotating of θ counterclockwise through $\frac{2\pi}{3}$ can be interpreted as firing of the neuron. When $r = 0$, all points on the circle rotate at the same speed, regardless of their position (37). At $|\frac{\omega}{r}| = 1$, the system undergoes a saddle-node infinite period (SNIPER) bifurcation (38) that separates the excitable and oscillating regimes.

We aim to use the active rotator as the neuron model of the reservoir assuming that its excitable behavior is beneficial for computation in the network. In section 2.2, we elaborate on adjusting the parameters of the given model.

1.5.2 Kuramoto with inertia model / pendulum equation

The generalized *Kuramoto model with inertia* is a second-order continuous-time dynamical system first introduced by Ermentrout (39) as a phenomenological model to explain the slow relaxation of firefly Pteroptyx malaccae's rhythm. The temporal dynamics of θ_i is given by

$$m_i\ddot{\theta}_i(t) = -\epsilon_i\dot{\theta}_i(t) + \omega_i + K\sum_{j=1}^{N} a_{ij}\sin\left(\theta_j(t) - \theta_i(t)\right) \tag{1.4}$$

where m_i is the strength of inertia (mass) of the i-th oscillator, ϵ_i is the damping, K is the coupling strength, a_{ij} is the element from the adjacency matrix $(a_{ij}) \in \mathbb{Z}_2^{N\times N}$, and ω_i is the natural frequency of the i-th oscillator, which is given by some distribution function $g(\omega)$.

The model 1.4 is known to exhibit various dynamical phenomena such as synchronization transitions (40, 41, 42, 43) and chimera states (44, 45). The stability of complete synchronization and nonlinear stability have been studied in (46, 47) and (48) respectively.

We, however, rely on the results from the study (49). In the given work, the authors derived the conditions on the *bistability* of synchronous dynamics. By E.M. Izhikevich (35), a neuron is *bistable* if it exhibits the coexistence of resting and spiking states. It has also been concluded inertia affects the intrinsic oscillator dynamics, i.e., it increases its dimensionality and enables bistability of cluster patterns. We are going to exploit these findings in order to design a dynamical system suitable for reservoir computing.

The Kuramoto model with inertia can be rewritten into the pendulum equation via rotating the coordinate frame and applying other transformations (50). Due to this, we will

henceforth adopt the term *pendulum model* when referring to the Kuramoto model with inertia, but with the following modifications: the gravity term $r \sin \theta_i(t)$ is included, the parameters of mass, damping, and eigenfrequency become homogeneous, i.e., $m_i = m$, $\epsilon_i = \epsilon$, and $\omega_i = \omega$. The latter means the oscillators now have identical intrinsic dynamics in the absence of input. However, in the reservoir computing scheme, we break the symmetry in the model by replacing the constant coupling strength K with random directed weights $w_{ij} \in \mathbb{R}^{N \times N}$, and the coupling strategy to all-to-all by setting $a_{ij} = \frac{1}{N} \forall i, j$.
Hence, Equation 1.4 can be reformulated as follows

$$m\ddot{\theta}_i(t) = -\epsilon\dot{\theta}_i(t) + \omega - r\sin(\theta_i(t)) + \frac{1}{N}\sum_{j=1}^{N} w_{ij} \sin\left(\theta_j(t) - \theta_i(t)\right) \tag{1.5}$$

where $r = \frac{g}{l}$, g is the acceleration, and l is the pendulum length.
We aim to exploit the pendulum model as a neuron model of the reservoir network. We assume that the bistable dynamics may give rise to useful computational properties in the network. Therefore, we select the model parameters values from the bistable region of the bifurcation diagram in (49). In section 2.2, we elaborate on adjusting the parameters of the given model.

1.6 Classification task with RC

According to Devroye's definition (51), *pattern recognition* or *discrimination* is predicting the unknown nature of an observation, where an observation is understood as a collection of numerical measurements. An *observation* can be formalized as an L-dimensional vector x. The unknown nature of the observation is called a *class* or *category*, and denoted by a variable y that takes values in a finite set $\{1, 2, ..., M\}$.
In this work, we consider a special instance of the pattern recognition problem, which is a classification of image data. It can be formulated as the task of learning a mapping $h(x)$

between an image represented as $D \in \mathbb{R}^{L \times L}$ and a target output value $y \in \{1, 2, ..., M\}$, where M is the number of classes:

$$h(x) : \mathbb{R}^{L \times L} \rightarrow \{1, 2, ..., M\} \tag{1.6}$$

The pattern recognition problem was proved to be solvable by means of reservoir computing. In (52), the images to be classified were suggested to be projected into a higher-dimensional space of the echo state network before feeding them into a feed-forward neural network that would then generate a prediction to the face recognition problem. In (53), a reservoir-based spiking neural network (r-SNN) was developed for terrain classification, which also promised to be useful for autonomous robot navigation systems. In (54), the authors designed the echo state network with a specific data encoding scheme, which showed the low error in the classification of the digits of the MNIST database (55).
Since an image segmentation problem can also be modeled as a pattern recognition problem (56), we also briefly review achievements in this domain. In (57), the authors proposed the reservoir computing algorithms for efficient and temporally consistent segmentation of spatio-temporal image series, which outperformed the U-Net and performed on par with the convolutional LSTM.

1.7 Aim of this work

We aim to design a reservoir network architecture with phase oscillators as neuron models that solves a temporal classification task. The reservoir architecture consists of data encoding (section 2.1), tuning the dynamics of neurons (section 2.2), state decoding (section 2.3), and readout training (section 2.4). To give rise to the high-dimensional states in the network, which are useful for computation, we find suitable parameter regimes using dynamical systems theory. More precisely, we rely on the studies about the active rotator

(37) and pendulum (49) models of phase oscillators. We evaluate these reservoir architectures by solving the MNIST digits classification task.

2

Methods

In order to describe our reservoir computing setup, the following steps need to be specified:

1. Encoding of image data to the reservoir
2. Dynamics of the reservoir
3. Decoding of the reservoir state and training of the readout

The aim of this chapter is to describe these steps. The reservoir performance is evaluated in Results, chapter 3.

2.1 Encoding of data

An input stimulus can be encoded in the network as initial conditions or (external) perturbations. We encode the stimulus via perturbations, i.e., each oscillator receives a time-

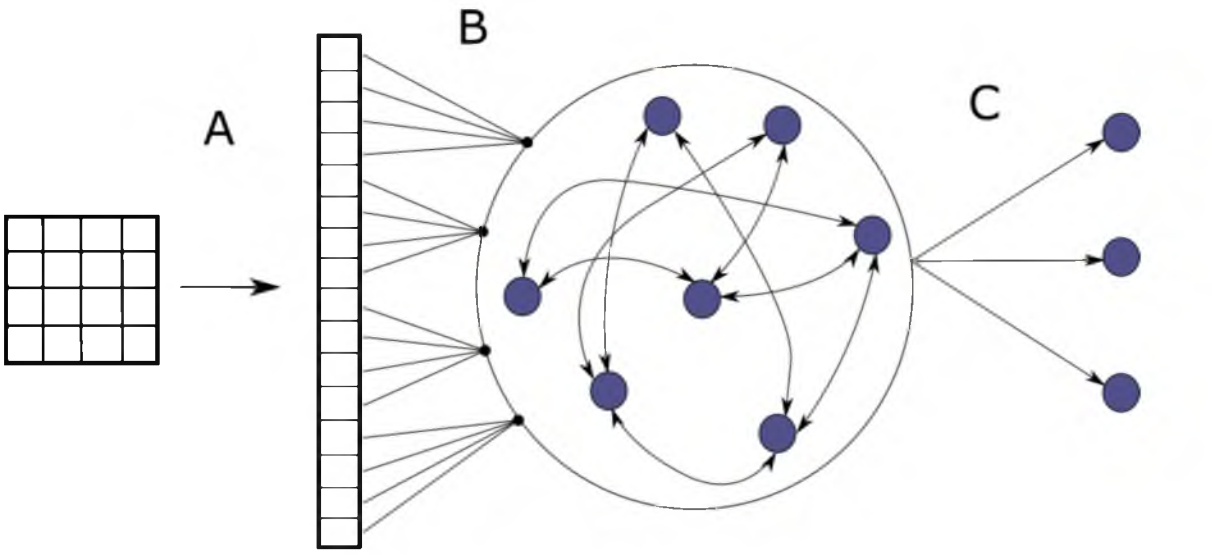

Figure 2.1: Architecture of reservoir. **A**: Transforming static image data into time series. **B**: Encoding data into reservoir. **C**: Decoding reservoir state and training output weights.

varying external input derived from the image data.
The encoding of the data consists of two steps: the preprocessing of the image data and feeding the processed data into the network. We start by describing the preprocessing step. We write

$$\mathcal{I} = \{D \in \mathbb{R}^{L \times L} | D \text{ is image}\} \tag{2.1}$$

for the space of single channel images from the classification task (55). Writing x_{ij} for a pixel of an image, by assumption $x_{ij} \in \{0, 1, ..., 255\}$. In order to control the average velocity across neurons, these values are scaled to a (compact) *target range* $[a, b]$ for usage

as external inputs. We define $\omega_{\text{scale}} = b - a$ and call it the *input scaling factor*. Its value is chosen based on observing the system's response to inputs of different amplitudes with all other reservoir parameters fixed. The choice of the *scaling factor* is primarily related to the oscillator model parameters, described in section 2.2. Denoting

$$x^{\max} = \max\{x_{ij} | (x_{ij})_{ij} = D\} \quad x^{\min} = \min\{x_{ij} | (x_{ij})_{ij} = D\}$$

as the maximum resp. minimum value of an image D, we use the min-max normalization (58) to scale each $x_{ij} \in D$ to the *target range* $[a, b]$:

$$x_{ij}^{\text{new}} = \omega_{\text{scale}} \frac{(x_{ij} - x^{\min})}{(x^{\max} - x^{\min})} + a \tag{2.2}$$

For each $D \in \mathcal{I}$, we obtain a new rescaled image $D_{\text{scaled}} = (x_{ij}^{\text{new}})_{ij}$ according to 2.2. In the following, we replace each image in $\mathcal{I}$ by its rescaled counterpart.
Next, we describe how to feed the data to the network. Let T be the total time of evolution of the network. T is split into L intervals of length S. Roughly, we split the image data into L pieces each of which is fed into the network for a time interval of length S. This means a static image is transformed to a time series which can be used for sequence-based processing. The idea here is to expose only partial information of the image to the network and obtain the corresponding 'computational response' which can be used for classification. To be precise, we first flatten the image matrix via the mapping

$$f : \mathbb{R}^{L \times L} \to \mathbb{R}^{L^2}, \quad \begin{pmatrix} - & v_1 & - \\ & \vdots & \\ - & v_L & - \end{pmatrix} \mapsto \begin{pmatrix} v_1^T \\ \vdots \\ v_L^T \end{pmatrix}. \tag{2.3}$$

Next, we transform the flattened image into a time series:

$$X : \mathbb{R}^{L^2} \to \mathcal{B}([0, T]; \mathbb{R}^L), \quad \begin{pmatrix} v_1 \\ \vdots \\ v_L \end{pmatrix} \mapsto \{t \mapsto \sum_{p=1}^{L} \chi_{[(p-1)S, pS]}(t) v_p\} \tag{2.4}$$

Here, $v_1, \ldots, v_L \in \mathbb{R}^L$, χ_A denotes the characteristic function of the set A and $\mathcal{B}([0,T];\mathbb{R}^L)$ denotes the Borel measurable functions with domain $[0,T]$ and codomain $\mathbb{R}^L$. To feed the time series into the network, we define the *read-in matrix* $W^{\text{in}} \in \mathbb{R}^{N\times L}$, where N is the number of neurons. W^{in} is a sparse matrix (59) the entries of which are i.i.d. and drawn from a Bernoulli distribution with $P[x=1] = P[x=0] = 1/2$. W^{in} remains constant during the entire system evolution, i.e., it depends neither on time nor the input image. Given an image $D \in \mathfrak{I}$ the input to the reservoir now is defined as

$$I(t) = W^{\text{in}} X(f(D))(t) = \sum_{p=1}^{L} \chi_{[(p-1)S,pS]}(t) W^{\text{in}} v_p, \tag{2.5}$$

that is, an input to the i-th neuron is given by

$$I_i(t) = \sum_{p=1}^{L} \chi_{[(p-1)S,pS]}(t)(W^{\text{in}} v_p)_i \tag{2.6}$$

2.2 Dynamics of reservoir

As a trade-off between model complexity and biological plausibility, we suggest using *active rotator* and *pendulum* models as computational units of a reservoir network with N units. Let $\delta\Theta_{ij}$ be defined as

$$\delta\Theta_{ij} = \theta_j - \theta_i \tag{2.7}$$

and the interaction matrix as

$$(\sin\delta\Theta)_{ij} = \sin(\delta\Theta_{ij}) \tag{2.8}$$

For convenience, we will use the following shorthand notation for 2.8

$$(\sin\delta\Theta)_i = \begin{pmatrix} \sin\delta\Theta_{i1} \\ \vdots \\ \sin\delta\Theta_{iN} \end{pmatrix} \tag{2.9}$$

The coupling matrix $W^{\text{res}} \in \mathbb{R}^{N\times N}$ is defined as a fully-connected random matrix, where each element is drawn independently from a uniform distribution on $[0, q]$, where $q > 0$. The matrix defines synaptic strengths of reservoir neurons, which remain fixed.
The *active rotator* reservoir is defined by

$$\dot{\Theta}(t) = -r\sin(\theta(t)) + \frac{1}{N}W^{\text{res}}\sin(\delta\Theta(t)) + w + I(t) \tag{2.10}$$

where r is the gravity term, ω is the intrinsic frequency.
The *pendulum* reservoir is defined as

$$m\ddot{\Theta}(t) + \epsilon\dot{\Theta}(t) = -r\sin(\theta(t)) + \frac{1}{N}W^{\text{res}}\sin(\delta\Theta(t)) + w + I(t) \tag{2.11}$$

where $r = \frac{g}{l}$ is the gravity term, g is the acceleration, l is the pendulum length, ω is the intrinsic frequency, ϵ is the damping coefficient, m is the mass of the pendulum.
The computational properties of the network depend on the interplay of the parameters that control the dynamical properties of neurons, their coupling, the external stimulus. Using the dynamical system analysis findings for the active rotator and pendulum models, we tune the reservoir parameters aiming for rich spatio-temporal activity and optimal performance in the classification task, i.e., to drive a reservoir into a regime suitable for computation. For the *active rotator* reservoir, it is essential to ensure the system's excitability, i.e., it always has to stay near the bifurcation point from equilibrium to the limit cycle. Therefore, the values of intrinsic frequency ω, coupling strength $(\mu_{ij})_{ij} = W^{\text{res}}$, ω_{scale} and the gravity term r are tuned in a way to make the total amount of energy received by an oscillator exceed r when a magnitude of time-varying input is close to its maximum. In this scenario, the oscillator starts rotating, which can be interpreted as neural firing (37). Otherwise, if the energy is less than r, the oscillator enters an equilibrium state, i.e., it stops rotating and goes to the oscillatory mode. From the neuroscientific perspective, this behavior can be interpreted as the transition to the resting state (35).
For the *pendulum* reservoir, the values of the damping coefficient ϵ, intrinsic frequency ω,

gravity term r, coupling strength $(\mu_{ij})_{ij} = W^{\text{res}}$, and ω_{scale} are adjusted. The ratio $\frac{\omega}{r}$ and ϵ are the parameters that control the bistable behavior of the oscillator. They are chosen in accordance with the parameter regions from the bistability diagram that indicates possible dynamics as a function of bifurcation parameters w, r, and ϵ (49).
To obtain the evolution of the reservoir, the system of ODEs in 2.10 and 2.11 are integrated using the Fourth Order-Runge Kutta Method (60) implementation from `scipy` (`scipy.integrate.solve_ivp`) (61). Initial phases of oscillators are sampled from a uniform distribution on $[0, 2\pi)$, initial velocities are sampled from a uniform distribution on $[0, 1]$. **Remark:** To isolate the effect of initial conditions from the other parameters on the performance, we generate them once and fix for the numerical experiments.
We denote the solution of the *active rotator* reservoir as $\Theta_r \in C^1([0,T];\mathbb{R}^N)$ and the solution of the *pendulum* reservoir as $\bar{\Theta}_p = (\Theta_p, \dot{\Theta}_p) \in C^1([0,T];\mathbb{R}^N) \times C^1([0,T];\mathbb{R}^N)$. The solution vectors are the activity of neurons in response to the input $I(t)$. In the following step, we aim to extract the features from these input-driven neural responses, i.e., to decode the reservoir state.

2.3 Decoding reservoir state

The aim of the section is to devise a scheme which allows us to decode the activity of the reservoir and use it in the classification task. For this, we first introduce two readout variables which allow for the decoding of the computation done by the network. In the next step, we describe how to train the *readout weights* on these variables for solving the task.

2.3.1 Kuramoto order parameter

The *complex order parameter* was introduced by Y. Kuramoto (62) to measure the phase coherence in all-to-all coupling models:

$$r(t)e^{i\psi(t)} = \frac{1}{N}\sum_{j=1}^{N} e^{i\theta_j(t)} \tag{2.12}$$

Here, $\psi(t)$ describes the average phase of all oscillators and $r(t)$ the degree of phase coherence. The complex number denotes the centroid of the phases (27). It is a common measure for identifying the synchronous behavior in the network. Motivated by this, we introduce a first readout variable based on the order parameter for the *velocity* of the reservoir averaged over the time intervals of exposition of the chunks of image data.
To be more precise, we define

$$Z : C^1([0,T];\mathbb{R}^N) \to C^1([0,T];\mathbb{R}), \quad \dot{\Theta} \mapsto \{t \mapsto \frac{1}{N}\sum_{j=1}^{N} |e^{i\dot{\theta}_j(t)}|\} \tag{2.13}$$

We call Equation 2.13 the *adapted order parameter for velocities*. Through measuring the system state with such a readout variable, we obtain information about synchronization in neural activity over time. To associate each row of an image with one scalar value, the average synchronization during its exposition time is calculated as follows

$$\overline{Z} : C^1([0,T];\mathbb{R}^N) \to \mathbb{R}^L \tag{2.14}$$

where the q-th entry of $\overline{Z}(\dot{\Theta})$ is given by

$$\overline{Z}(\dot{\Theta})_q = \frac{1}{S}\int_{(q-1)S}^{qS} Z(\dot{\Theta})(t)dt \tag{2.15}$$

The readout vector now is defined as

$$\overline{Z}(\dot{\Theta}_r), \quad \overline{Z}(\dot{\Theta}_p). \tag{2.16}$$

for the *theta* and *pendulum* reservoir respectively.
Thus, an image is represented by a vector of average frequency synchronization of the

oscillators.
The examples of velocity time series and corresponding order parameter values are given in section 5.3.

2.3.2 Frequency clustering

As we observe the emergence of the qualitatively different dynamics in the pendulum reservoir compared to the active rotator reservoir, namely, clustering of oscillators with a common mean frequency, we introduce a second readout variable to decode the reservoir state. It captures the number of clusters and the mean intracluster frequency given the perturbation of the time-varying input. Let K denote the number of frequency clusters at time $t \in [0, T]$; w_k, $k = 1, \ldots, K$ the average frequency in k-th cluster; n_k, $k = 1, \ldots, N$ the number of oscillators in the k-th cluster. The weighted average velocities of the system during input processing is defined as follows

$$V : C^1([0,T];\mathbb{R}^N) \to C^1([0,T];\mathbb{R}), \quad \dot{\Theta} \mapsto \{t \mapsto \frac{1}{N}\sum_{k=1}^{K} v_k(t) n_k(t)\} \tag{2.17}$$

To again associate each row of an image with one scalar value, the average velocity during its exposition time is calculated as follows

$$\overline{V} : C^1([0,T];\mathbb{R}^N) \to \mathbb{R}^L \tag{2.18}$$

where the q-th entry of $\overline{V}(\Theta)$ is given by

$$\overline{V}(\dot{\Theta})_q = \frac{1}{S}\int_{(p-1)S}^{pS} V(\dot{\Theta})(t)dt. \tag{2.19}$$

The readout variable for the pendulum reservoir now is defined as

$$\overline{V}(\dot{\Theta}_p) \tag{2.20}$$

The effect of the decoding via the two readout variables on the performance is studied in subsection 3.2.2.

2.4 Readout training

The final step consists of constructing the readout layer that shall approximate the target output based on the decoded reservoir state. The following steps are to be performed:

1. Task specification
2. Model selection
3. Model fitting (i.e., estimating the parameters of the model)

2.4.1 Task specification

To fit the parameters of the classifier model, a *training set* is required. The training set is defined as a labeled set of input-output pairs $\mathcal{D} = \{(\mathrm{x}_i, y_i)_{i=1}^n\}$, where n is the number of training data points. Each training input x_i is an L-dimensional vector, values of which are called *features*; y_i is a *response variable*, which in our problem setting takes categorical values.
Thereby, depending on the readout variable, a training input x_i is defined as

$$\mathrm{x}_i = \overline{Z}_i(\dot{\Theta}) \in \mathbb{R}^L \tag{2.21}$$

or

$$\mathrm{x}_i = \overline{V}_i(\dot{\Theta}_p) \in \mathbb{R}^L \tag{2.22}$$

where $\overline{Z}_i(\dot{\Theta})$ or $\overline{V}_i(\dot{\Theta}_p)$ is a readout vector of the network dynamics given the i-th image as an input. The response variable y_i representing an image's class is given as follows:

$$y_i \in \{1, \ldots, M\} \tag{2.23}$$

with M being the number of classes.
Solving the classification task via the readout can be then formulated as learning a mapping $W^{\text{out}} \in \mathbb{R}^{L \times M}$ which transforms inputs x_i into outputs y_i, such that an error between the reservoir outputs $\hat{y}_i$ and the target signals y_i is minimized, i.e., $|y_i - \hat{y}_i| \to \min$.

2.4.2 Model selection and fitting

As classification models, we use Softmax regression and Random forest. We estimate their parameters using the previously defined training set $\mathcal{D}$.

2.4.2.1 Softmax regression

Following K. Murphy's definition (10), the *multinomial logistic regression* (also called *maximum entropy classifier*) is a model of the form

$$p(y = m|\mathrm{x}, W) = \frac{\exp(W_m^{\intercal}\mathrm{x})}{\sum_{m'=1}^{M} \exp(W_m^{\intercal}\mathrm{x})} \tag{2.24}$$

2.4.2.2 Random forest classifier

We relate to the concept of classification decision trees explained by Kevin P. Murphy (10). Decision trees are defined by recursively partitioning the input space, and defining a local model in each resulting region of input space.

$$f(\mathrm{x}) = \mathbb{E}[y|\mathrm{x}] = \sum_{k=1}^{K} w_k \mathbb{1}(\mathrm{x} \in R_k) = \sum_{k=1}^{K} w_k \phi(\mathrm{x}; v_k) \tag{2.25}$$

where R_k is the k-th region, w_k is the mean response in this region, and v_k encodes the choice of variable to split on, and the threshold value, on the path from the root to the k-th leaf.

A *random forest classifier* is then defined as a collection of decision trees $\{h(\mathrm{x}, W_k), k = 1, \ldots\}$, where the W_k are independent identically distributed random vectors and each tree casts a vote for the most popular class at input x (63).

3

Results

This chapter aims to evaluate the reservoir's performance in the classification task. We use the MNIST handwritten digits classification as a benchmark problem of pattern recognition (55). As the primary performance measure, we take *accuracy* – the fraction of the correct predictions made by the readout layer trained on the decoded reservoir states.
To save computational resources required for integrating the system of ODEs with the time-varying inputs, we reduced the training data set's size from 60 000 to 12 000 images, which constitute 20% of the original training sample. The distribution of classes in the resulting sample is the same as in the original data set. The full test data set is used for accuracy measurements, i.e., all 10 000 images.

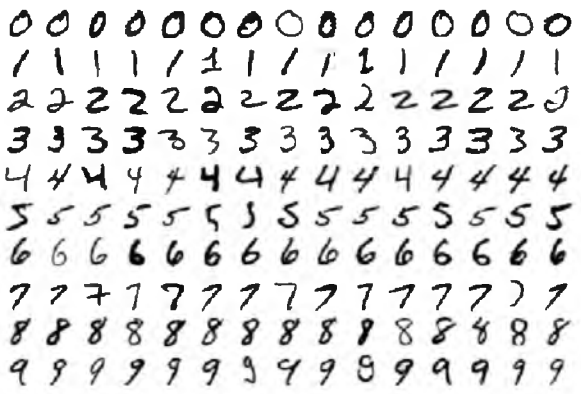

Figure 3.1: Handwritten digit examples from the MNIST database

3.1 Active rotator reservoir

In this section, we aim to evaluate performance of the readout layer trained on the *active rotator* reservoir's decoded states to solve the classification task. The readout variable is fixed to the adapted order parameter for velocities given by Equation 2.15. If not specified otherwise, the input vector length is fixed to $L = 28$.

3.1.1 Fixed network size

We measure the accuracy of image class predictions performed by the readout layer of the reservoir. The network size is fixed to $N = 50$. The results are compared with respect to the classifier model used for optimizing W^{out}. We observe that the Random forest classifier yields better readout weights. The parameters of the classifiers are given in detail in chapter 5.

Classifier model	Test accuracy
Softmax Regression	0.641
Random forest	**0.6869**

Remark: Note that improving the integration accuracy leads to better performance: we can observe a steep increase in performance for the active rotator model:

Classifier model	Test accuracy
Softmax regression	0.69
Random forest	**0.81**

3.1.1.1 Input length effect

Splitting the input data further down yields increased performance as well. We assume that this effect is caused by the higher dimensionality of the readout vector. The data, however, suggest that too short input vectors also impose a penalty on the performance (Figure 3.2). One should opt for intermediate lengths as a trade-off choice.

3.1.2 Variable network size

Comparing the performance depending on the network size, we observe a trend of increased accuracy for larger networks regardless of the estimation method (Figure 3.3). Note that the dimensionality of the readout vector does not change with the increase of the network size. This hints that the increased computational capabilities of the higher-dimensional dynamics are reflected even in the lower-dimensional readout vector.
Remark: The strategy for modifying the network's size is described in detail in subsection 5.2.1.

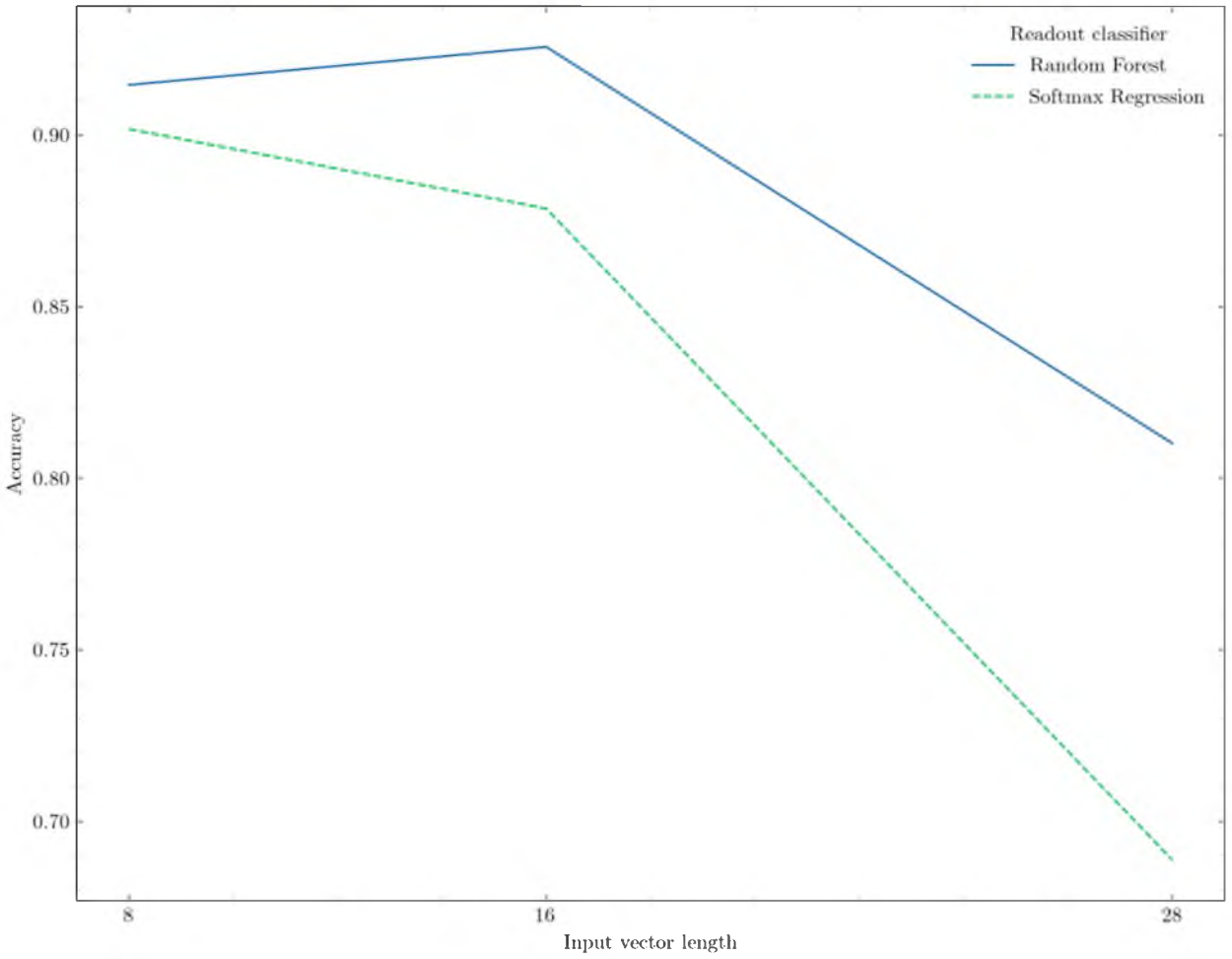

Figure 3.2: Active rotator reservoir: accuracy of image classification w.r.t. input vector length L. Parameters: $\omega = 0.5$, $r = 0.56$, $\omega_{\text{scale}} = 0.1$, $W^{\text{res}} \sim \mathcal{U}([0.0, 0.1])$, $S = 100$.

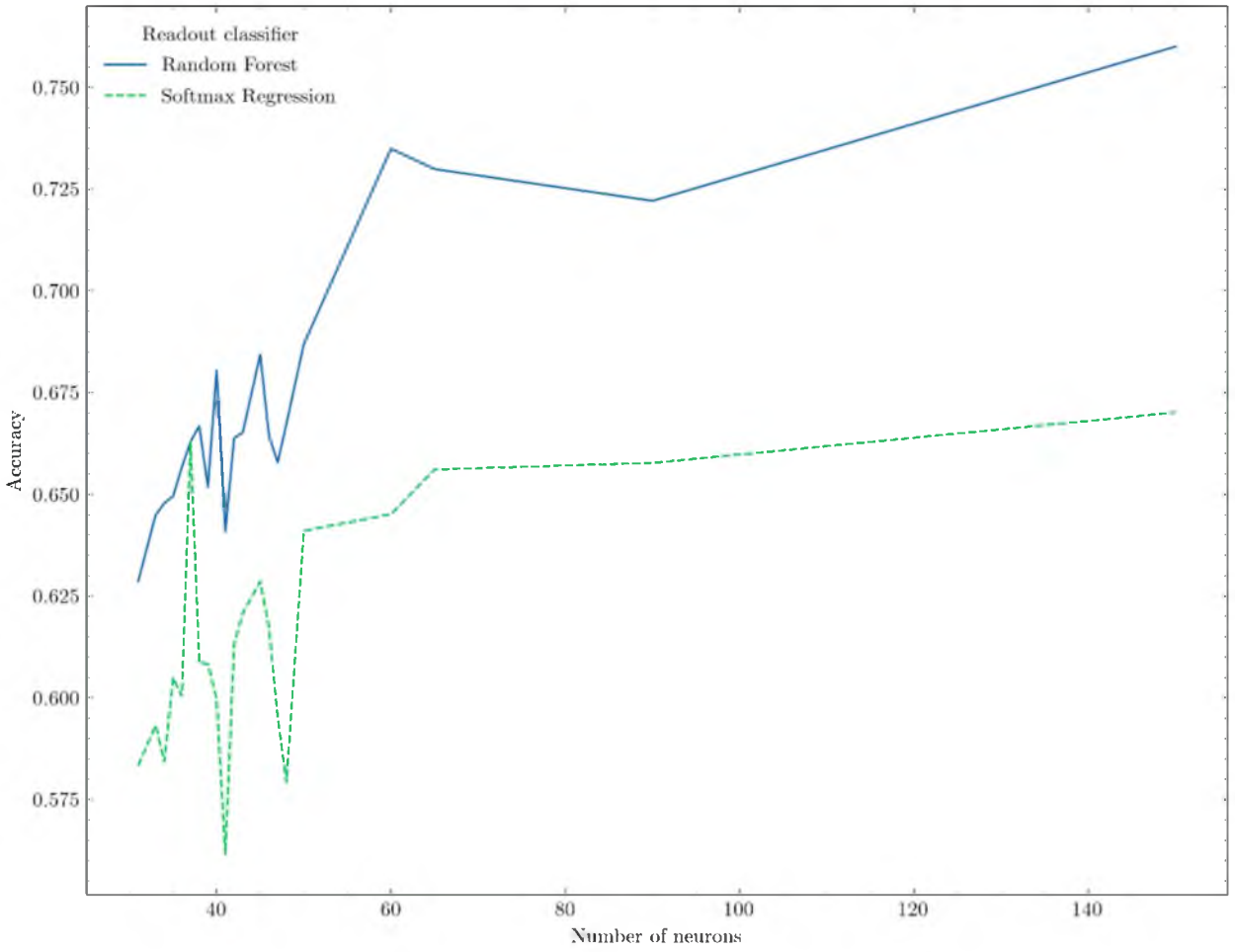

Figure 3.3: Active rotator reservoir: accuracy of image classification w.r.t. the network size. $N_{\text{min}} = 30, N_{\text{max}} = 150$ are the minimal and maximal network sizes correspondingly. Reservoir parameters: $\omega = 0.5, \quad r = 0.56, \quad \omega_{\text{scale}} = 0.1, \quad W^{\text{res}} \sim \mathcal{U}([0.0, 0.1]), \quad S = 100.$

3.2 Pendulum reservoir

In this section, we aim to evaluate performance of the readout layer trained on the *pendulum* reservoir's decoded states to solve the classification task. The exploited readout variables are given by Equation 2.15 and given by Equation 2.19. If not specified otherwise, the input vector length is fixed to $L = 28$.

3.2.1 Fixed network size

We measure the accuracy of image class predictions performed by the readout layer of the pendulum reservoir. The network size is fixed to $N = 50$. The results are compared with respect to the classifier model used for optimizing W^{out}.

Classifier model	Test accuracy
Softmax regression	0.6524
Random forest	**0.7264**

3.2.2 Decoding approaches comparison

We study the effect of the state decoding techniques, i.e., the adapted order parameter for velocities and frequency clustering, on the accuracy of classifications. The classifier model is fixed to Random forest. In the case of the frequency clustering readout variable (Equation 2.19), the clusters of oscillators are determined via the agglomerative clustering method from sklearn (64).

Readout variable	Test accuracy
Order parameter	**0.7264**
Frequency clustering	0.6466

Given the accuracy measurements, we conclude that the adapted order parameter function for velocities better captures the computational capabilities of the higher-dimensional system dynamics compared to the frequency clustering readout variable.

3.3 Reservoir comparison

We compare the performance of active rotator and pendulum reservoirs using the order parameter for velocities (Equation 2.15) as the readout variable.

Neuron model	Softmax regression	Random forest
Active rotator model	0.64	0.69
Pendulum model	0.65	**0.73**

We observe the pendulum model achieves higher accuracy than the active rotator model. We assume it is due to its richer bistable dynamics.

4

Conclusions and discussion

We conclude that the proposed reservoir computing scheme can be harnessed for the image classification task. The results of the work supported our expectations about the suitability of active rotator and pendulum oscillators for use as computational units due to their excitable and bistable dynamics. We saw that the richer dynamics of the pendulum model leads to increased performance compared to the active rotator model. We also observed the dependence of reservoir performance on the network size and input vector length. Despite being a simple measure for describing the dynamical state, the low-dimensional order parameter for velocities demonstrated practicality for capturing the computational properties of high-dimensional spatiotemporal dynamics.
The results of the thesis support the theories of oscillator-based and reservoir computing. The proposed approach is, however, different from the classical reservoir computing methods, where the readout layer is trained directly on the time evolution of the states. Nevertheless, we showed that the averaged transformation of the dynamical state of the network can also be used for solving the classification task with decent accuracy.

We assume our approach generalizes to other neuron models with similar properties but it is beyond the scope of this study to confirm this hypothesis. We presuppose that the reservoir architecture has the potential to be implemented in neuromorphic hardware, e.g., using memristor devices or Mott insulator-based oscillators.
Future studies should take into account the effects of the reservoir topology, input layer sparsity, synapse plasticity, transient dynamics, the ratio of excitatory and inhibitory connections on the reservoir performance. It would be also beneficial to evaluate the reservoir architecture on robustness to noise and deformations in the input data as well as usefulness in real-world applications.

5

Appendix

5.1 Simulations parameters

This section contains technical details of the reservoir network simulations.

5.1.1 Neuron models

5.1.1.1 Active rotator

Parameter	Value/Distribution	Description
ω	0.5	eigenfrequency
r	0.56	gravity
$\Theta(t_0)$	$\mathcal{U}_{[0.0,2\pi]}$	initial phases
W^{res}	$\mathcal{U}_{[0.0,0.1]}$	synapse strength
W^{in}	$\text{Bern}(\frac{1}{2})$	input weights
ω_{scale}	0.1	input scaling factor
L	28	input vector length
S	100	time of one vector exposition

5.1.1.2 Pendulum

Parameter	Value/Distribution	Description
m	1.0	mass
ϵ	0.1	damping
ω	0.6	eigenfrequency
r	1.0	gravity
$\Theta(t_0)$	$\mathcal{U}_{[0.0,2\pi]}$	initial phases
$\Theta(t_0)$	$\mathcal{U}_{[0.0,1.0]}$	initial velocities
W^{res}	$\mathcal{U}_{[0.0,0.1]}$	synapse strength
W^{in}	$\text{Bern}(\frac{1}{2})$	input weights
ω_{scale}	0.4	input scaling factor
L	28	input vector length
S	500	time of one vector exposition

5.1.2 (Hyper) parameters of classifier models

The classifier models for the readout layer were implemented using **scikit-learn** library (64).

5.1.2.1 Softmax regression

Class **sklearn.neural_network.MLPClassifier** has been used for the implementation of Softmax regression. MLPClassifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters (64).

Parameter	Value	Description
activation	'logistic'	Activation function for the hidden layer
hidden_layer_sizes	()	Number of neurons in the ith hidden layer
solver	'sgd'	Solver for weight optimization
alpha	0.0001	L2 penalty (regularization term) parameter
momentum	0.9	Momentum for gradient descent update
learning_rate	'constant'	Learning rate schedule for weight updates

5.1.2.2 Random forest

Class **sklearn.ensemble.RandomForestClassifier** has been used for the implementation of Random forest – a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Parameter	Value	Description
n_estimators	500	Number of trees in the forest
criterion	"entropy"	Function to measure the quality of a split
min_samples_split	2	Minimum number of samples required to split an internal node
bootstrap	True	Whether bootstrap samples are used when building trees

5.2 Algorithms

5.2.1 Network connectivity modification

The scheme of adding neurons to the network is described as follows. Suppose we want to increase the existing network of size $N = n$ by K neurons. The connectivity of the given matrix is defined by $(x_{ij}) = W_n^{\text{res}} \in \mathbb{R}^{n \times n}$. Let $k = 1, \ldots, K$. Then, (x_{ik}) denotes the connections from the existing neurons to the new neurons and (x_{kj}) denotes the connections from the new neurons to the existing neurons. The synapse weights of (x_{ik}) and (x_{kj}) are taken from the same distribution as the entries of W_n^{res}. To extend the original matrix W_n^{res}, the following operations are performed:

$$W_n^{\text{res}'} = \begin{bmatrix} W_n^{\text{res}} & (x_{ik})^\intercal \end{bmatrix}, \quad W_{n+K}^{\text{res}} = \begin{bmatrix} W_{n+K}^{\text{res}'} \\ (x_{kj}) \end{bmatrix}$$

Now, $W_{n+K}^{\text{res}} \in \mathbb{R}^{(n+K)\times(n+K)}$ is the connectivity matrix of the expanded network. Such network's structure modification scheme allows for the comparison of performance of differently sized networks by ensuring that for every given network with $N = n$ neurons, the following holds: $W_n^{\text{res}} \subset W_{n+1}^{\text{res}} \subset \ldots \subset W_{n+K}^{\text{res}}$, i.e., each smaller network's connectivity

is a subspace of a larger one. The given scheme is also applicable to the corresponding $W_n^{\text{in}}, W_{n+1}^{\text{in}}, \ldots, W_{n+K}^{\text{in}}$ matrices, with the difference being that only rows are to be added to each new matrix.

5.3 State decoding: visualization

To illustrate the decoding step of our reservoir computing scheme, we visualize velocity time series and readout vectors given images of different classes as inputs.

5.3.1 Pendulum model

Reservoir parameters: $N = 30$, $\omega = 0.6$, $r = 1.0$, $\omega_{\text{scale}} = 0.4$, $W^{\text{res}} \sim \mathcal{U}_{[0.0,0.1]}$, $S = 100$, readout variable is given by 2.13

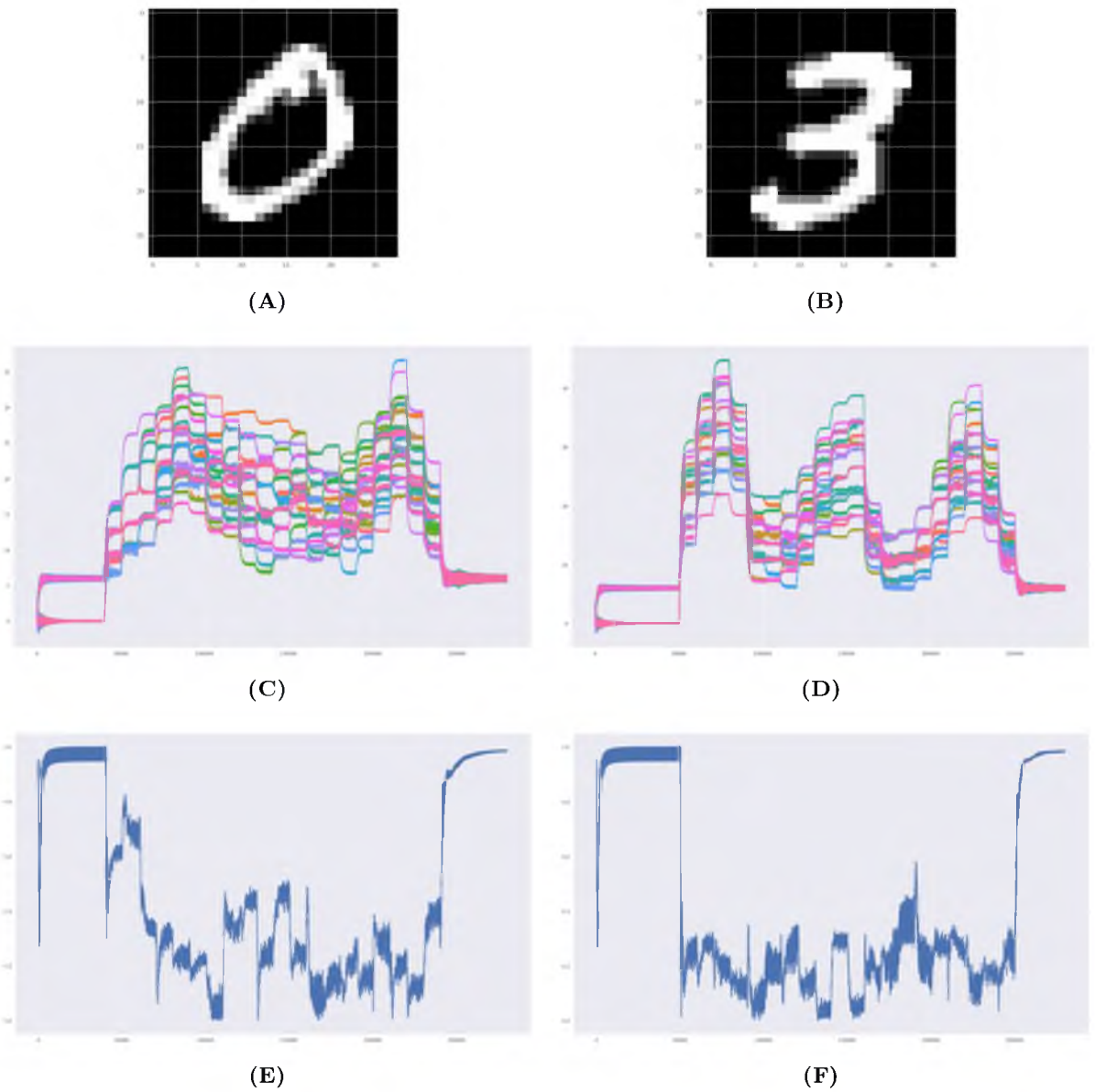

Figure 5.1: Reservoir dynamics in response to the images of different classes. **C-D**: velocity time series given **A-B**. **E-F**: frequency synchronization time series given **A-B**

References

[1] Pedro Mateos-Aparicio and Antonio Rodríguez-Moreno. **The Impact of Studying Brain Plasticity**. *Frontiers in Cellular Neuroscience*, **13**:66, 2019. 1

[2] Alex Graves and Jürgen Schmidhuber. **Framewise phoneme classification with bidirectional LSTM and other neural network architectures**. *Neural Networks*, **18**(5):602–610, 2005. IJCNN 2005. 3

[3] Ruhi Sarikaya, Geoffrey E. Hinton, and Anoop Deoras. **Application of Deep Belief Networks for Natural Language Understanding**. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, **22**(4):778–784, 2014. 3

[4] Sai Zhang, Jingtian Zhou, Hailin Hu, Haipeng Gong, Ligong Chen, Chao Cheng, and Jianyang Zeng. **A deep learning framework for modeling structural features of RNA-binding protein targets**. *Nucleic Acids Research*, **44**(4):e32–e32, 10 2015. 3

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **Imagenet classification with deep convolutional neural networks**. *Advances in neural information processing systems*, **25**:1097–1105, 2012. 3

[6] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org. 3, 4, 5

[7] Kurt Hornik. **Approximation capabilities of multilayer feedforward networks**. *Neural Networks*, **4**:251–257, 1991. 3

[8] D. H. Hubel and T. N. Wiesel. **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex**. *The Journal of Physiology*, **160**(1):106–154, 1962. 3

[9] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. **Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition**. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. 3

[10] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2012. 3, 23

[11] Gyorgy Buzsáki. *Rhythms of The Brain*, pages xiv, 448 p. 01 2009. 3

[12] Yunbo Wang, Haixu Wu, Jianjin Zhang, Zhifeng Gao, Jianmin Wang, Philip S. Yu, and Mingsheng Long. **PredRNN: A Recurrent Neural Network for Spatiotemporal Predictive Learning**. 2021. 3

[13] Alex Graves and Navdeep Jaitly. **Towards End-To-End Speech Recognition with Recurrent Neural Networks**. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, **32** of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR. 3

[14] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. **On the Properties of Neural Machine Translation: Encoder-Decoder Approaches**, 2014. 3

[15] Hava T. Siegelmann and Eduardo D. Sontag. **On the Computational Power of Neural Nets**. COLT '92, page 440–449, New York, NY, USA, 1992. Association for Computing Machinery. 5

[16] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. **Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets**. *CoRR*, **abs/1901.09049**, 2019. 5

[17] Wolfgang Maass, Thomas Natschläger, and Henry Markram. **Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations**. *Neural computation*, **14**:2531–60, 12 2002. 5

[18] Herbert Jaeger. **The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'**. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, **148**, 01 2001. 5

[19] Tadashi Yamazaki and Jun Igarashi. **Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit**. *Neural Networks*, **47**:103–111, 2013. Computation in the Cerebellum. 6

[20] Charis Mesaritakis, Adonis Bogris, Alexandros Kapsalis, and Dimitris Syvridis. **High-speed all-optical pattern recognition of dispersive Fourier images through a photonic reservoir computing subsystem**. *Opt. Lett.*, **40**(14):3416–3419, Jul 2015. 6