Department of Humanities
Institute of Cognitive Science

Bachelor Thesis

Learning And Computing In A Photonic Recurrent Neural Network

by

Paul Liebenow
(pliebenow@uni-osnabrueck.de)

April 16, 2020

First Supervisor:
M.Sc. Pascal Nieters

Second Supervisor:
Dr. Daniel Brunner

Abstract

Understanding the information processing capabilities of the human brain is one of the main scientific challenges of the 21. century. Technologically, we are faced with an overflow of information and there is an urgent need to make sense of the available data. Neural Networks offer a promising solution to both problems: On the one hand, by building more realistic brain-like neural networks we may be able to better understand the human brain. On the other hand, neural networks help to make use of the available data. There exist several differences between the human brain and modern digital computing devices. One of them is parallelism. We know from neuroscience that the brain is intrinsically parallelly structured. The common von-Neumann architecture, however, misses this parallelism. Therefore, the question arises if computing devices can be build which can fill this gap. In this thesis, an approach will be presented that makes use of optical computing. Optical computing devices offer a promising answer to the problem of parallelism. Optical signals can be separated much easier and thus allow parallel processing of many optical emitters. We present an approach to optical computing of a certain type of neural network, named Echo State Network, and simulate the system using Python. Furthermore, an algorithm to improve the learning rule of the system is presented.

Contents

1 Introduction

Since the beginning of human culture there was an interest to build computing devices. One of the oldest evidence is the Antikythera mechanism which can be compared to an analog computer. It was used to predict planetary motion and moon phases according to the lunisolar calendar and has been dated back to around 200 BC (Brunner et al. (2019), p. 1). More than 2000 years later, in 1941, Conrad Zuse developed the Z3 which was built with electromechanical relays and was able to compute floating point numbers with reprogrammable algorithms. The Z3 was already used for applied computations for engineering problems such as the mechanical properties of airplane wings (Brunner et al. (2019), p. 1).
The main conclusion from those examples is that the physical substrate of the computing device changed according to the problem it was used to solve and the technological level of the civilisation which built it (Brunner et al. (2019), p. 1). If we apply this conclusion to the present, it becomes apparent that we face a new computing challenge which is to compute large scale neural networks to predict and to classify the world around us and to understand the brain itself. The common von-Neumann architecture does not suffice for this problem because it misses the parallel nature of the human brain. Most notably is the von-Neumann bottleneck between the CPU and the memory which leads to long CPU waiting times because transfer of data from the memory cannot keep up with processing speed of the CPU (Wulf and McKee (1995)). Also this problem is minimzed by using CPU caches in modern computers, it illustrates well the sequentiel nature of the von-Neumann architecture (Drepper (2007)).
One of the most prominent technologies of the 20. century is photonics. Its development began in 1960 with the invention of Light Amplification by Stimulated Emission of Radiation (Laser) and optical fibres in 1970 (Quimby (2006)). The development of optical computing devices began over five decades ago (Brunner et al. (2019), p. 1). Unfortunately, the transition from electronical to optical computing devices is hard. One reason is that electronics is much cheaper in production. But there is reason to think that optics and electronics fit very well together to compute large-scale neural networks. Electrons are charged particles and interact in a non-linear fashion. Thus, electronics is very well suited to compute non-linear transforms (Brunner et al. (2019), p. 3). In contrast, photons have no charge and do not interact. This enables parallel information processing because signals can be separated using e.g. wavelength or polarization (Brunner et al. (2019), p. 4).
The basis of this thesis is the paper "Reinforcement learning in a large-scale photonic recurrent neural network" by Bueno et al. (2018). The paper in-

troduces an experimental setup which implements a photonic neural network with up to 2025 nodes where each node is a pixel on a Spatial Light Modulator (SLM). The coupling of the nodes is physically generated by a Diffractive Optical Element (DOE) which works in parallel. Maktoobi et al. (2019) showed that the concept is scaleable to up to 30000 nodes connected fully in parallel. The employed learning algorithm uses an evolutionary learning approach. The algorithm iterates over a binary output mapping and optimizes its weights randomly according to some error measure. In contrast to digital implementations of neural networks where the learning algorithm has to cope with noisy input data, non-digital implementations of neural networks are influenced by noise stemming from internal connections or their surrounding. One of the most apparent noise sources is for example the change in room temperature. Semenova et al. (2019) investigated the impact of noise on linear recurrent and multi-layer networks. The authors showed the impact of noise and derived an analytical description.

The first aim of this work is to connect these experimental and theoretical findings by implementing a simulation of the system using the Python programming language. We want to use the simulation to compare learning with and without noise and to understand the effect of noise on the state of the network better. The prediction problem the network has to solve is an one-step-ahead prediction of the Mackey-Glass sequence. The impact of noise can be accessed by e.g. the prediction accuracy of the network. The simulation enables a systematic investigation of this impact by running computer experiments under varying noise conditions.

The second aim is to improve the learning algorithm. The idea we present here is to generalize the evolutionary learning algorithm to a genetic learning algorithm. A genetic algorithm does not only modify one output mapping but a set of the same by recombination.

The work is structured as follows. In the method part, we will first describe the physical laws which are needed to understand the experiment. Then we will introduce reservoir computing and some concepts from dynamical system theory. The main part of this thesis is divided into two subparts. The first part describes the experimental setup and explains the simulation. Then we will present the results of the comparison between learning with noise and without noise. The second part introduces the implementation of the genetic learning algorithm and the results. Finally, we will discuss the obtained results and give a conclusion.

2 Fundamentals of Optical Reservoir Computing

2.1 Optics

Photonics is the technological use of optics. Therefore, we will shortly introduce some basic concepts from optics which are required to understand the physical properties of the experiment. This section follows the book "The Feynman Lectures on Physics" (Feynman et al. (2011)).
Optics is the part of physics which explains the nature of light and phenomenons which result from it. On the one hand light can be understood as a beam of photons. This interpretation is used in geometric optics. Geometric optics applies when the involved wavelengths are very small compared to the size of the equipment (Feynman et al. (2011), p. 353). Light is also an electro-magnetic wave consisting of a magnetic field and an electric field which oscillate perpendicular to each other. This interpretation is used in wave optics. In the following we will ignore the magnetic field because the properties of a light wave can be described by its electric field. The electric field E is a vector with two components E_x, E_y (Feynman et al. (2011), p. 451) and we only consider coherent light waves.

2.1.1 Lenses

Lenses can be grouped roughly into positive and negative lenses. Positive Lenses (biconvex) focus collimated input light, and negative (biconcave) lenses result in diverging light beams. We will focus our considerations on positive lenses since this type is used in the experiment. There are two basic facts about lenses which are schematically illustrated in Fig. 1 (Feynman et al. (2011), p. 374):

1. Every incoming light beam parallell to the VX line will hit the focal point R with distance f from the lense

2. Every light beam crossing the VX line through focal point U will be parallel to the XW line

Since the triangles PVU and TXU are similar, it holds that $y'/f = y/x$. Following the same reasoning for the triangles SWR and QXR it is $y'/x' = y/f$. This allows to derive the lense equation by solving for y'/y:

$$\frac{y'}{y} = \frac{x'}{f} = \frac{f}{x}. \tag{1}$$

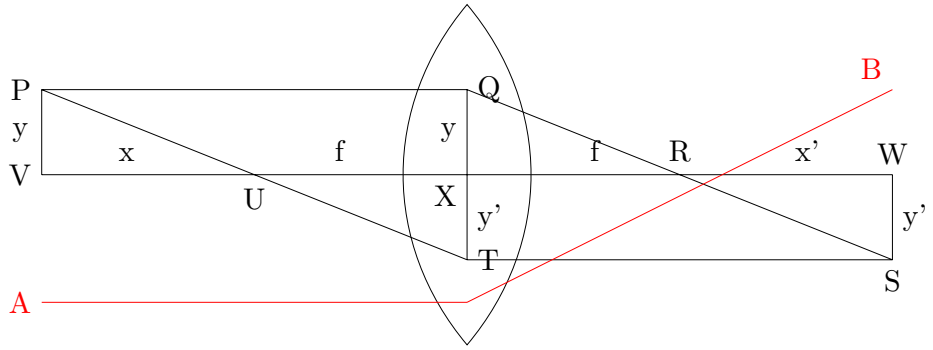


Figure 1: Capital letters denote points, lowercase letters the distance between points. VP denotes the object plane and SW the image plane. The red lines show schematically spherical aberration.

The term y'/y is the magnification of our lense since y is the real size and y' the virtual size of the object. Thus, this equation connects focal length f with magnification y'/y and the focal length with the distances x, x'. We use lenses to collimate the laser beam into parallel beams. Thus, if a laser beam originates from focal point U in Fig. 1, the rays of its beam will be aligned in parallel behind the lens. In the experiment we are not only using one lense but a system of lenses. A system of lenses can be analysed by applying Eq. 1 sequentially for every lense of the system by taking the output of the first lense as the input for the second. Therefore, complicated optical systems can be broken down to simpler individual sections.
One noise source influencing our experiment is spherical aberration. This is schematiclly depicted in Fig. 1 as the red beam. Spherical aberrations occur when light beams far away from the horizontal axes do not intersect the focal point R (Feynman et al. (2011), p. 377). Therefore, we use a second microscope lense to focus the light beam.

2.1.2 Polarizing Beam Splitter

From the perspective of wave optics we can understand the phenomenon of polarization. Polarization describes the geometrical orientation of the oscillation of the electrical field (Feynman et al. (2011), p. 456). Light can be linearly, circularly or elliptically polarized. If it is linearly polarized it means the oscillation of the E_x and E_y components are in phase. A s-polarized light wave has the electrical field vector orthogonal to the plane of incidence. Conversely, a p-polarized light wave has its electric field vector oriented parallel to the plane of incidence. In this context, the plane of

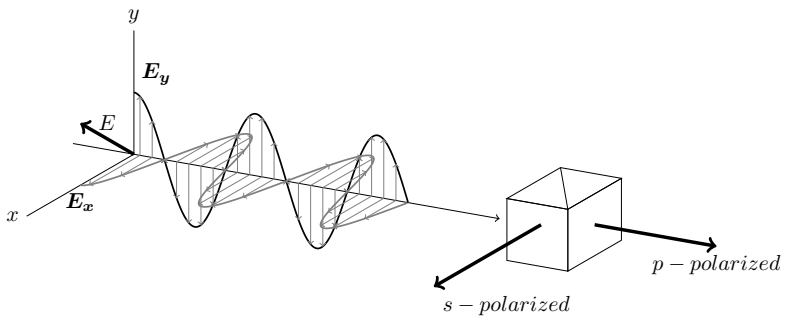


Figure 2

incidence is defined as the sum of the incidence vector and the normal vector on the surface. Polarization is leveraged in our experiment via a polarizing beam splitter (PBS). A PBS is made of two triangular glass prisms which are glued together. At the interface of the two glass prisms birefringent materials are used to split the light wave into two orthogonal parts (Paschotta ("accessed April 3, 2020"a)). In Fig. 2, a linearly polarized light wave travels towards a PBS and is split into a p-polarized and s-polarized part in a certain ratio. The s-polarized part goes through the plane of incidence and p-part is reflected with a certain angle from the plane of incidence.

2.1.3 Diffractive Optical Elements

Wave optics can be used to understand the phenomenon of diffraction. Diffraction occurs when a light wave interacts with an obstacle e.g. a slit. The Huygens–Fresnel principle states that every point in a wavefront hitting a slit is itself a point source of a spherical wave. These spherical waves propagate through the slit and interact with other spherical waves originating from the same slit or other slits in the neighbourhood. At a specific position the phase difference of two waves is a multiple of 2π, we observe a phenomenon called constructive interference. The light waves add up and generate a wave of a higher amplitude. If the phase difference of two waves is an odd multiple of π destructive interference occurs which reduces the amplitude (Feynman et al. (2011), p. 407). The diffractive order describes the maxima of the intensity distribution. In the experimental setup, we approximate a diffractive optical element (DOE) by an optical grating. A DOE is an optical component which can shape a light beam. We use this beam shaping property to establish the optical connections between the nodes of our recurrent neural network. In our case the phase of the light beam was modulated by an optical grating

such that the light waves diffract and physically generate the connections. In Fig. 3, a light beam hits a diffractive grating and point sources are generated. The spherical waves generated by the point sources interact constructively or destructively and thus the phase of the light wave is modulated. A lense projects the light rays onto the focal plane. The point P on the focal plane can only be a constructive maxima if the phase difference of two spherical waves from adjacent slits is a multiple of 2π. Every point satisfying this condition on the focal plane will be a maxima (Harvey and Pfisterer (2019)). The grating equation in the case of an orthogonal incident light beam is:

$$p^{DOE}(\sin(\theta_k^{DOE}) = k\lambda, \tag{2}$$

here p^{DOE} is the width of the grating period and θ_k^{DOE} is the angle of the diffracted ray. The whole number k denotes the diffractive order. The term $p^{DOE}\sin(\theta_k^{DOE})$ denotes the path difference of the light beams. The path difference denotes how much the path of a lightbeam was shifted by the DOE. Thus, the grating spacing of the DOE determines the location of the diffractive orders on the focal plane. In Fig. 4 (a), the emitters in the plane

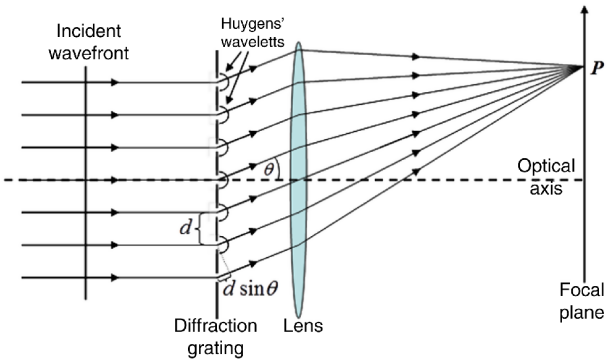


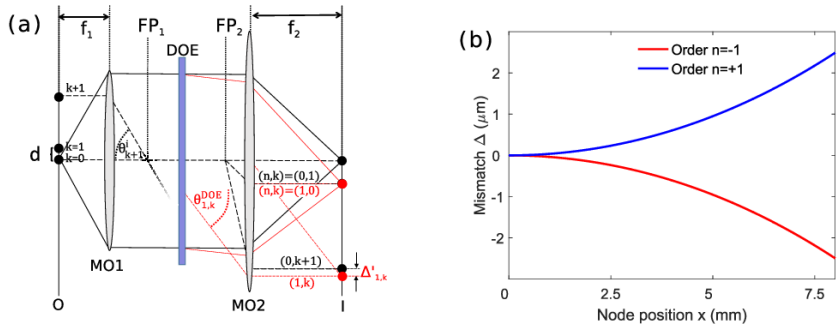
Figure 3: The image is taken from Harvey and Pfisterer (2019). The width of the diffractive grating is denoted with d here instead of p^{DOE} and θ instead of θ_k^{DOE}.

0 are focused by an objective lense of a microscope on the plane I. Emitters are located at position $x_k = kd$. If we want to couple emitters $k = 0$ with $k = 1$ it is required that node 0's first diffractive order overlaps with node 1's zero diffractive order. When a light wave is incident with an angle θ_k^i on a grating the first maximum occurs at (Maktoobi et al. (2019), p. 2):

$$p^{DOE}(\sin(\theta_k^{DOE}) - \sin(\theta_k^i)) = \lambda. \tag{3}$$

Here p^{DOE} is the width of the grating period, the incidence angle is θ_k^i on the diffraction grating and θ_k^{DOE} is the angle of the diffracted ray. The difference to Eq. 2 is the $\sin(\theta_k^i)$ term. Since the lightbeam is not orthogonal

incident anymore, the path difference depends on the angle of incidence, too. Thus, if the wavelength λ and the spacing p^{DOE} is known, the experimentator can predict where the intensity maximum will be. The parameters of



(a) **(b)**

Figure 4: images from Maktoobi et al. (2019)

this setting λ, p^{DOE} were chosen such that coupling for central emitters is optimized (Brunner et al. (2019), p. 85). As can be seen in Fig. 4 (b), for emitter position outside the centre the mismatch between diffractive orders and positions of the emitters slowly increases because of a breakdown in the paraxial approximation (Maktoobi et al. (2019)). If the paraxial approximation holds the propagation direction if a lightwave will be close to the optical axis (Paschotta ("accessed April 3, 2020"b)). Maktoobi et al. worked out the theoretical limitations of using a DOE. They showed that the coupling of up to 30000 emitters located in an array of spacing size 10 μm is possible (Maktoobi et al. (2019), p. 7). Thus, diffraction offers a very promising approach to parallel information processing since the present approach can be potentially extended to over a million elements connected fully in parallel (Maktoobi et al. (2019), p. 2).

2.2 Dynamical Systems

A dynamical system D is a triple (X, T, R) where X is the state space or phase space, T is time and $R : X \times T \to X$ is the operator which defines the rule according to which the system is advancing in time. As an example of a dynamical system, we will now introduce delay systems. A continous delay system with $T = \mathbb{R}$ can be described by a delay differential equation (DDE):

$$\dot{x} = f[t, x(t), x(t - \tau_D)], \tag{4}$$

which represents the operator R. In this setting τ_D is the delay time, $f[\cdot]$ a nonlinear function and $x(t - \tau_D)$ the delayed feedback signal. If we compare

Eq. 4 to an ordinary differential equation (ODE), we added $x(t-\tau_D)$. This small modification makes the system's state space infinite dimensional since intializing in continous time means an infinite amount of initial conditions (Brunner et al. (2018)). Delay Systems are hard to understand. One can often only make qualitative predictions about their behavior in the future. One way of qualitative analysis is to search for attractors in the state space. An attractor, in dynamical system theory, is a subspace of the phase space X to which the trajectories tend as time increases. Furthermore, a fixed point is a point attractor and defined as $x \in X$ such that $f(x(t), \cdot) = x$ for all following iterations. A bifurcation is a qualitative change of the fixed points of the system (Strogatz (2018)). Fixed points can be destroyed and generated by varying a control parameter. We will now introduce a prominent delay system in optics. From Eq. 4 we can derive the equation:

$$T_R\dot{x} + x(t) = f[\beta x(t-\tau_D) + \rho u(t) + \phi_0], \tag{5}$$

by setting $f[\cdot]$ to a $\sin(\cdot)$ nonlinearity. Now Eq. 5 constitutes the Ikeda delay equation (Brunner et al. (2018)). In Fig. 5, we present the bifurcation diagram of the Ikeda map (Ikeda (1979)). In a bifurcation diagram fixed points are mapped against the parameter value. From parameter value $8-14$ there exists only one stable fixed point. At parameter value 14 a bifurcation occurs and from $14-16$ two stable fixed points exist. From parameter value 17 on the system falls in the chaotic regime. The dynamics of the experiment are governed by coupled Ikeda maps (Brunner et al. (2019), p. 99).

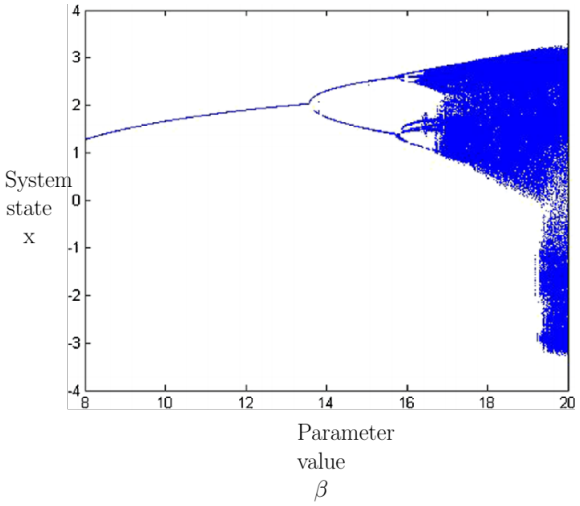


Figure 5: The bifurcation diagram of the Ikeda system. The image taken from Benrhouma et al. (2016).

2.3 Reservoir Computing

In the early 2000s Reservoir Computing (RC) arose in Neuroscience and Machine Learning as a method to predict dynamical systems (Jaeger and Haas (2004)). RC is divided into Echo State Networks (ESN) and Liquide State Machines (LSN). Both are recurrent neural networks with fixed random connections between the input and the recurrent connection inside the reservoir. In Fig. 6, we present the general structure of an ESN. The input signal is injected via an injection matrix W_{in} into the dynamical reservoir. The reservoir consists of a recurrently connected "hidden layer". The connection between nodes in the reservoir is given by the static connectivity matrix W. Output feedback can be installed if necessary for the problem via the matrix W_{fb}. The information inside the reservoir is only accessible by an output mapping and the connection to the output is trainable. The central idea is to project the data into a high dimensional space, the reservoir, where it can be predicted or classified. The method was independently discovered as ESN (Jaeger and Haas (2004)) and LSN (Maass and Markram (2004)).
We will focus on Echo State Networks (ESN) here. Since their introduction ESN were an important bridge between neural networks and unconventional computing approaches. In unconventional computing approaches the reservoir mostly consists of a physical system e.g. in our case a laser system. But the principle works with water waves too (Fernando and Sojakka (2003)). ESN are used for supervised, temporal learning tasks (Jaeger (2007)).

2.3.1 Reservoir computation

In Figure 6 an ESN is schematically depicted. Let $M \in \mathbb{N}$ be the number of input neurons, $N \in \mathbb{N}$ denote the number of neurons in the reservoir and let $O \in \mathbb{N}$ be the number of output neurons. We denote with a discrete time step with $t \in \{1, .., T\}$ where T is the number of datapoints. The input signal $u(t) \in \mathbb{R}^M$ is fed into the network via the random injection matrix $W_{in} \in \mathbb{R}^{N \times M}$. The state of the network $x(t+1) \in \mathbb{R}^N$ is computed by (Brunner et al. (2019), p. 41):

$$x(t+1) = f\left(W \cdot x(t) + W_{in} \cdot u(t+1) + W_{fb} \cdot y(t)\right). \tag{6}$$

The matrix $W \in \mathbb{R}^{N \times N}$ denotes the connectivity matrix of the network. The matrix $W_{fb} \in \mathbb{R}^{N \times M}$ denotes the optional feedback matrix . The state activation function is in our case the $\cos^2(\cdot)$ which is applied element-wise to the current state $x(t)$. After all state vectors are computed the state matrix X is obtained by stacking the single state vectors $x(t)$ column-wise. Therefore, the Matrix X will be of dimension $X \in \mathbb{R}^{N \times T}$ and it contains the activation for all neurons for all timesteps.

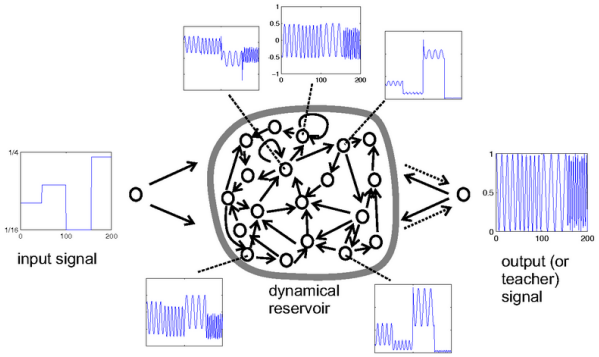


Figure 6: A generic ESN. Dotted arrows indicate trainable connections, solid arrows indicate randomly fixed connections. Image from Jaeger (2007).

2.3.2 Reservoir readout

Now we have set up the reservoir and can use it for prediction or classification. The output $y(t) \in \mathbb{R}^n$ of the Echo State networks is obtained by (Brunner et al. (2019), p. 41):

$$y(t) = W_{out} \cdot x(t). \tag{7}$$

To compute Eq. 7, we need to access the information memorized in the reservoir. For this reason, we compute the readout matrix W_{out}. The matrix $W_{out} \in \mathbb{R}^{n \times N}$ is classically obtained by computing the pseudoinverse $X^\dagger$ of X and multiplying it with Y (Jaeger (2007)):

$$W_{out} := Y \cdot X^\dagger. \tag{8}$$

The matrix Y in Eq. 8 consists of the desired output stacked row-wise. Therefore, W_{out} corresponds to the linear regression weights between the states $x(t)$ and the desired output $y(t)^{target}$. The readout matrix W_{out} is generated to minimize some error measure. In most cases the mean squared error (MSE) is used. The MSE is defined as (Brunner et al. (2019), p. 38):

$$MSE(y, y^{target}) = \frac{1}{T} \sum_{j=1}^{T} (y_j(t) - y_j^{\text{target}}(t))^2. \tag{9}$$

To compute the pseudoinverse of the system we need information about the whole matrix X. This will become important later because in our photonic reservoir we only have access to the current state of the network. Therefore, we can't use global methods as the least square approach of computing the pseudoinverse.

2.3.3 Properties of Reservoir computing

We will now consider the properties which make the ESN so effective. The echo state property or fading memory property states that "the reservoir will asymptotically wash out any information from initial conditions" (Jaeger (2007)). Or in other words: "if the network is undriven, any transient dynamics eventually die out and the network evolves towards an uniquely stable state" (Brunner et al. (2019), p. 42). This property depends on the spectral radius of the connectivity matrix $\rho(W)$. If the spectral radius is too large the reservoir can host several fix points which can lead to chaotic behaviour. The memory about past states and inputs is largest when system dynamics are close to a bifurcation. Therefore, the spectral radius is one of the most important parameters in the reservoir framework and has to be adjusted according to the task (Jaeger (2007)).
The second property is the approximation property. The approximation property states that an ESN can realize every "nonlinear filter with bounded memory arbitrarily well" (Jaeger (2007)). For this to be the case the fading memory and the separation property has to be fulfilled and the reservoir has to be large enough (Brunner et al. (2019), p. 43). The separation property states that differences in the input sequence results in separate trajectories of reservoir states $x(t)$. Or in other words: "if the reservoir is large enough any difference in the input sequence result in a linearly separable difference in the state space." (Brunner et al. (2019), p. 43).
For our photonic reservoir these properties are altered. One difference is that we only have binary entries in our output matrix W_{out}. This is limiting the approximation property since the output will always be strictly positive (Brunner et al. (2019), p. 103).

3 Evolutionary Learning in an opto-electronic Reservoir

3.1 An opto-electronic Reservoir

3.1.1 Why optical neural networks

In the previous chapter we saw that computing in an echo state network mainly involves matrix-vector multiplication, transformation with nonlinear functions and long-ranged connections inside the reservoir. Now we will show why optics is better suited for many of these tasks than electronics.
First of all, electronic circuits have an upper limit response time which is $R \cdot C$ where R is the Ohmic resistance and C is the capacity. In the worst

case this scales quadratically with the length of any transmission line in the circuit (Brunner et al. (2018), p. 3). This makes long-ranged connections impossible and induces delays (Brunner et al. (2018)). Thus, "connections in the von-Neumann architecture are mostly short-ranged" and the switching (charging) of signal wires requires a lot of energy (Brunner et al. (2019), p. 5). But neural networks require highly interconnected and long-ranged hardware connections. Contrary, photonics offers signal transmission with the speed of light. The propagation speed of photons is only influenced by the refractive index of the propagation medium and does not scale with the transmission line's length (Brunner et al. (2018), p. 3).
Secondly, electrical current consists of electrons which are charged particles. Coulomb forces between them make their interaction highly nonlinear (Brunner et al. (2018), p. 3). In contrast, light consists of photons which have no charge. This limits physical interaction between photons (Brunner et al. (2018), p. 3). A drawback of non-interacting particles is that it is not easy to implement purely optical non-linear functions. A very active field of research is the "development of advanced materials and novel waveguide geometries" to compute all optical nonlinearities (Brunner et al. (2019), p. 4). But the linear nature of optical components is also their major advantage. Optical components are very well suited for tasks which can be solved in parallel like matrix-vector multiplication. Parallel computations can be implemented in optics by using components which work on a 2D plane like the DOE or a simple lense (Brunner et al. (2019), p. 8). A good example of the parallel information processing capabilities in optics is the impulse response function of a lense's back focal plane which corresponds to the spatial Fourier transform of an entire image (Brunner et al. (2019), p. 8). We have already seen that the DOE can process light in parallel by the mechanism of Diffraction. In 2015, Google published the Tensor Processing Unit (TPU). The TPU was developed to compute large scale vector matrix products but still is not able to do this fully in parallel. This is known in optics since decades (Brunner et al. (2019), p. 5). To sum up, optics is better suited for fast signal transmission and parallel computations but lacks nonlinear elements. Thus, opto-electronical hybrid approaches unify both worlds. We will now describe how theses approaches can be implemented.

3.1.2 Approaches to optical computing

One of the first implementations of an opto-electronic neural network was done by Farhat et al. (1985). The authors implemented a Hopefield network which is a feedback based neural network. The idea of a Hopefield network is to implement an auto-associative memory to restore patterns of activa-

tion. This is similar to hebbian learning where connection strength between neurons is increased when neurons are activated often at the same time. Patterns v_i^m with $m \in \{1, ..., M\}$, $i \in \{1, ..., N\}$ are stored in a memory matrix T which is generated by the equation (Farhat et al. (1985), p. 2):

$$T_{i,j} = \sum_{m=1}^{M} v_i^m v_j^m. \tag{10}$$

The network is then confronted with input patterns which in their case were generated by an array of LEDs. The LED array corresponds to neurons in an ANN. The binary input vector $\hat{v}$ was multiplied with the memory masks T and collected onto a photodiode array to measure the output signal. If activation excited a certain threshold the signal was amplified or thresholded if this limit was not reached. Then the signal is sent back electronically to the LED array. The system retrieves the pattern by iteratively repeating this process converging to the vector which is closest to one of the stored vectors in the memory matrix (Farhat et al. (1985)). This proved the concept that opto-electronic neural networks are realizable.
The interest decayed in 1980s because of "consumption, volume and scaling issues" (Larger et al. (2012), p. 1) The field has been under reconsideration after the invention of the RC algorithms which has led to the development of photonic delay systems (Larger et al. (2012)).
Larger et al. (2012) proposed a delay-coupled opto-electronic reservoir. We

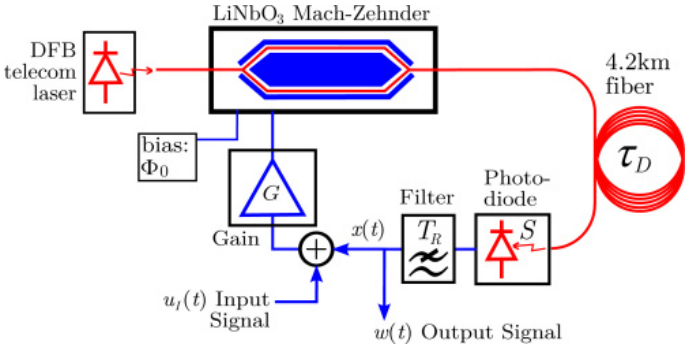


Figure 7: Opto-electronic implementation. Red arrows correspond to optical signal, blue to an electrical signal. Image from Larger et al. (2012).

defined a delay-coupled system in the dynamical systems chapter. In Fig. 7, the experimental setup is presented. A laser beam is injected into a Mach-Zehnder modulator, which physically realizes a phase and amplitude modulation (Brunner et al. (2018)). As a result, the output signal is a $\sin(\cdot)$ nonlinearity. The signal is then injected into a delay line which is a 4.2 km

long optical fibre. After passing the delay line, the signal is detected by a photodiode. Thus, resulting in a $\sin^2(\cdot)$ electrical signal because of the relation:

$$I^0 = |E^0|^2. \tag{11}$$

Physically, the optical field of the laser beam is converted to the intensity of a photon inducing a current through the diode. Furthermore, a lowpass filter is applied to the current from the photodiode to physically realize the characteristic response time T_R (Larger et al. (2012)). Now the input signal $u(t)$ is added. Finally, the feedback gain β is multiplied with the signal. "Parameter β is controlled via the laser diode power" (Larger et al. (2012), p. 4). The multiplication can be realized e.g. by adjusting the laser intensity (Brunner et al. (2018), p. 8). This is summarized in the delay differential equation (Larger et al. (2012), p. 5):

$$\epsilon \dot{x}(s) + x(s) = \beta sin^2[\mu x(s-1) + \rho u_I(s-1) + \phi_0], \tag{12}$$

where ρ is the weight of the input information and μ corresponds to the feedback scaling. The parameter $\epsilon = T_R/\tau_D$ is the oscillator response time normalized to the delay τ_D and $s = t/\tau_D$ is the normalized time. The bias ϕ_0 corresponds to the offset phase of the MZM (Larger et al. (2012)). Eq. 12 corresponds to the state of the system. In the reservoir framework, we defined the network state by Eq. 6. Thus, these equations make the connection between digital reservoirs and delay-coupled opto-electronic reservoirs.
Since a reservoir consists of many nodes, which are not easily implementable in optics, the authors overcome this constraint by using a single nonlinear element, the optical fibre, which is then time delayed. The nodes are placed on the fibre by time multiplexing. Time multiplexing maps the input signal $u(t)$ to the reservoir layer: "we define virtual nodes by dividing the total delay interval of length τ_D , realized by 4.2 km optical fiber, into subintervals of length Θ. At the end of each subinterval we extract the respective virtual node states" (Larger et al. (2012), p. 5). Each node of the reservoir is located with distance θ on the fibre. Thus, mimicking the nodes of a traditional reservoir. The coupling of the nodes is restricted to the nearest neighbouring nodes and is determined by the ratio of the response time and node distance T_R/θ. "The longer (shorter) T_R is relative to θ, the more (less) consecutive virtual nodes are connected" (Larger et al. (2012), p. 5). After passing the delay line, the procedure for time de-multiplexing is to divide the system's output into non-overlapping intervals. This creates the state vector x which is then multiplied by a W_{out} matrix to produce the output. The W_{out} can be computed by the reservoir algorithm described in section 2.3. The system worked well on task such recognizing spoken digits and one-step-ahead time

series prediction (Larger et al. (2012)).
On the way to an all-optical implementation the question arises how the connection between nodes inside the reservoir can be made explicit. In the previous experiment an optical fibre served as the reservoir and the nodes were placed "virtually" on the fibre. But one big advantage of optics is that information in a 2d plane can be processed in parallel. A way to do this is diffractive coupling based on Diffractive Optical Elements (DOE, as explained in the optics chapter).

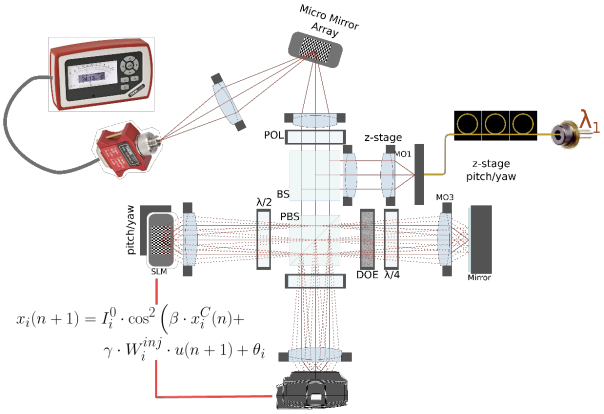


Figure 8: The experimental setup. Image from Bueno et al. (2018)

3.2 The experiment

We mentioned earlier that reservoir computing is used as a bridge between neural networks and unconventional computing approaches. Now we will show in detail how this connection can be implemented. This connection will again be made with Eq. 6 from the reservoir computing section. Every mathemtical operation we will introduce in the following has a counterpart in the physical realization. In Fig. 8, a schematic of the experimental setup is depicted (Bueno et al. (2018)). The main energy source of our opto-electronic network is a laser beam which operates in s-polarization mode (Bueno et al. (2018)). The laser beam can be modelled as a Gaussian beam since it consists of monochromatic light centred around a point of maximal light intensity with uniformly decreasing intensity towards the edges (Quimby (2006)). The laser beam is projected into our opto-electronic reservoir via a system of lenses. The first lense collimates the beam and the second lense focuses it onto the beam splitter (BS) which reflects the light beam to the polarizing beam splitter (PBS). After passing the $\lambda/2$ plate, the light wave is

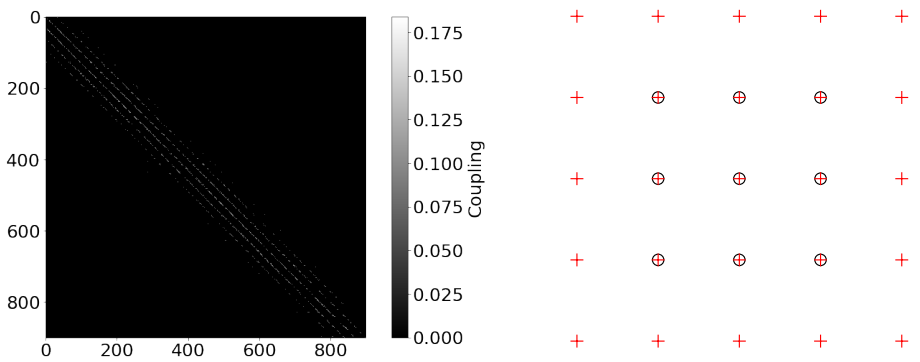


Figure 9: (a) the connectivity Matrix W^{DOE}, (b) black circles denote the first pass through the DOE coupling nearest neighbours, red crosses denote the second pass resulting in the 5×5 convolution pattern.

linearly polarized and aligned to the polarization axis of the SLM. The SLM represents the neurons of our system. Each neuron is a grey valued pixel. Physically, these pixels consist of liquid cristals and modify the phase of the incoming light wave. The SLM is used actively which means that a voltage changes the refractive index of each pixel. In turn, this changes the phase of the incoming light wave. The reflected light from the SLM is led through $\lambda/2$ plate towards the PBS once more. The p-polarized part of the light wave passes the PBS. The optical field after the PBS and before the DOE is given by (Brunner et al. (2019), p 87):

$$I_i(t) = I_i^0 \cdot \cos\left(\frac{2\pi}{\kappa_{SLM}} \cdot (x_i^{SLM}(t) + \theta_i^0)\right). \tag{13}$$

Here $x_i^{SLM} \in \{0, .., 255\}$ is the SLM pixel grey value, κ_{SLM} is a device related constant and θ_i^0 is the SLM offset in grey scale 11.1 ± 1.1. I_i^0 is the initial illumination intensity (Bueno et al. (2018)). After the DOE, the light wave passes a $\lambda/4$ plate, is then reflected by a mirror and passes the $\lambda/4$ plate once more. The connectivity matrix of our reservoir is induced by the DOE. This yields a light wave in s-polarization mode. The PBS reflects the s-polarized optical field again and another microscope images the waves onto the camera (Bueno et al. (2018)). The camera reads the state of the system at time step n (Brunner et al. (2019), p 88):

$$x_i^C(t) = |\sum_{j=1}^{N} W_{i,j}^{DOE} \cdot \sqrt{I_j(t)}|^2. \tag{14}$$

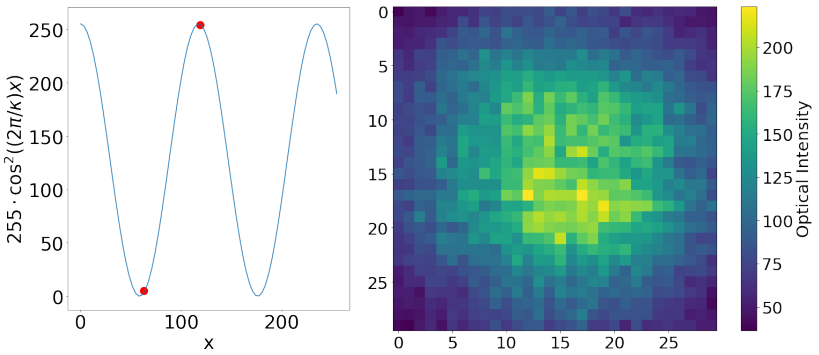


Figure 10: (a) The nonlinear function $\cos(\cdot)^2$ in blue and the two sampling points with $\theta = 55$ and $\delta = 64$ in red and (b) illumination intensity matrix I^0.

The DOE is passed twice with a diffractive order of 3×3. This results in a convolution of diffraction pattern and the resulting pattern is of shape 5×5. W^{DOE} is the coupling matrix created by the DOE. It is squared because the signal is passing twice through DOE and the sum itself is the convolution. Previously we said that the SLM is used actively. This means that the grey values of each SLM pixel x_i are adjusted by an external computer. The reservoir state $x(n+1)$ is defined as the SLM's intensity in p-polarization (Brunner et al. (2019), p. 99):

$$x_i^{SLM}(t+1) = I_i^0 \cdot \cos^2\left(\beta \cdot x_i^C(t) + \gamma \cdot W_i^{inj} \cdot u(t+1) + \theta_i\right), \qquad (15)$$

with $i \in \{1, .., 900\}$ the neuron index. This equation consists of 4 parts. The first part, $\beta \cdot x_i^C(t)$ is the memory of the system. In Fig. 9 (a), we present the coupling matrix induced by the DOE. The white diagonal lines show the $5x5$ diffraction pattern. This pattern results from the convolution which we schematically show in Fig. 9 (b). The black circles denote the first pass through the DOE and the red crosses the convolution resulting from the second pass.
The second part $\gamma \cdot W_i^{inj} \cdot u(n+1)$ is the injection of the input signal. The parameter $\gamma \in (0, 1)$ determines the strength of the injection. The matrix W^{inj} is uniformly drawn from the closed interval $[0, 1]$ but constant during training. The injected signal u is a timeseries generated from the Mackey-Glass equation which is a time-delayed differential equation. We will state how we preprocessed the data when describing how the reservoir framework is applied.
The third part is the $\theta_i \in \{0, ..., 255\}$ with $i = \{1, .., 900\}$ which is the offset

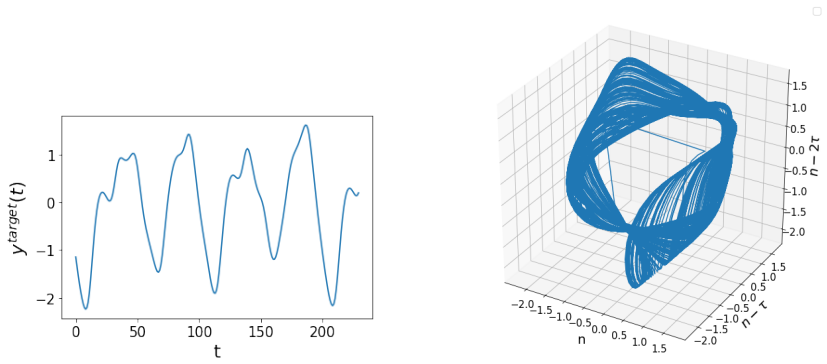
of the system in grey scale. The non-monotonous slope of our nonlinear function was divided by a Bernoulli distribution with probability μ we sample from position θ and with probability $1-\mu$ we sample from $\theta+\delta$ values. In Fig. 10 (a), we present the case with $\theta = 55$ and $\delta = 64$. Since the reservoir state matrix is strictly positive (positive activation function) the approximation capabilities of our system are reduced. We mitigated this problem by introducing the two sampling points and sampling probability which allowed the system to choose from positive and negative slopes by shifting the input of the nonlinear function into a minimum or maximum. "The consequence are node ensembles which exhibit dynamics operating along their negative, others along their positive slope" (Brunner et al. (2019), p 104).
Finally, the nonlinear function $I_i^0 \cdot \cos(\cdot)^2$ governs the dynamics of the system. The illumination intensity I^0 is approximately normal distributed around the centre (Fig. 10 (b)). For the modelling of the experiment we used the intensity unit since a camera can only detect the intensity of the signal and not the complex electronical field. The same applies for the SLM. The relation between the intensity and the optical field is:

$$I^0 = |E^0|^2. \tag{16}$$

We use the $\cos(\cdot)^2$ as an activation function since it characterizes the nonlinear transformation of every SLM pixel. The nonlinear transform is obtained by the projection of the optical signal onto the p-axis because of the polarization by the PBS ($\cos(\cdot)$) and finally the intensity detection (Eq. 15) results in the $\cos^2(\cdot)$ nonlinearity.
Now Eq. 15 corresponds to Eq. 6 in the reservoir framework. For every iteration we use the same 200 points from the Mackey-Glass (MG) sequence with which we generate the reservoir state matrix X. We normalize by subtraction of the minimum of the training set and division of the maximum minus the minimum of the data and finally multiplication with 255. This way, we transformed the data points to the range of $[0, 255]$ such that the signal has the 8 bit range of the SLM. We removed 30 points before starting to compute the state of the system because of their transient nature. We obtain the reservoir state matrix X by stacking the $x(t) \in \mathbb{R}^{900}$ such that we get $X \in \mathbb{R}^{900\times 170}$.

3.2.1 Reservoir readout

Previously we described how the state of the system evolves. Now we will describe how the evolution of the system is used to solve a time series prediction problem. The p-polarized component of the light wave from the SLM is reflected towards the Digital Micro Mirror Device (DMD) by the PBS. After



(a) **(b)**

Figure 11: The pre-processed data y^{target} (a) and its attractor (b).

the beam hits the beam splitter, it is reflected towards a lens which projects it onto the mirrors of the DMD. The experimenters used a simple lens to image a version of the reservoir's state onto the DMD because the nodes of the SLM are spatially distributed (Brunner et al. (2019), p. 101). The DMD consists of mirrors which can be flipped by $\pm 12°$ in either direction. The entries of the DMD are changed electronically. In this setting every mirror of the DMD is connected with their corresponding SLM pixel. Afterwards, the DMD reflects the light onto a detector which only detects light from pixel with $\pm 12°$. Hence, realising our binary readout.
In the Reservoir framework, the desired computational task was now to predict a timeseries. The training of the network was constrained on the W^{out} matrix which corresponds to the binary DMD. We computed the error ϵ_k and compared with the previous state ϵ_{k-1}. If $\epsilon_k < \epsilon_{k-1}$, then modification of the output matrix is kept. Otherwise, it is reversed.
More precisely: at $k = 1$ we measured the 170 points and computed ϵ_1. For $k+1$ and all following learning iterations, we modify W^{out} as follows (Brunner et al. (2019), p 102):

$$W_k^{select} = rand(N) \cdot W^{bias}, \tag{17}$$

$$l_k = max(W_k^{select}), \tag{18}$$

$$W_{l_k,k}^{out} = \neg W_{l_k,k-1}^{out}. \tag{19}$$

In Eq. 17, we create a random vector rand(N) uniformly from the interval $[0, 1]$ and multiply it with the Bias matrix W^{bias} whose entries are also from

the interval $[0, 1]$. In Eq. 18, we select the index of the largest entry. This entry is a number between 1 and 900 and is used in Eq. 19 to negate the W^{out}. We can negate because W^{out} is a binary matrix. Then W^{Bias} is updated according to (Brunner et al. (2019), p 103):

$$W^{Bias} = 1/N + W^{Bias}. \tag{20}$$

$$W^{Bias}_{l_k} = 0. \tag{21}$$

Therefore, the values of W^{Bias} are increased by $1/N$ in every learning iteration (Eq. 18) and the position chosen previously is set to zero (Eq. 20). This secures that we change entries in W^{out} uniformly and are learning away from weights which have been updated previously (Bueno et al. (2018)). The output of the k-th learning iteration becomes (Brunner et al. (2019), p 101):

$$y = W^{out}_k \cdot X \tag{22}$$

In Fig. 11 (a), we present our target data with it's attractor (Fig. 11 (b)). For training our output matrix we normalized y^{target} by subtracting the mean and dividing by the standard deviation. We computed the MSE and because points are z-normalized this gives us the NMSE for the output error. We implement the one-step ahead prediction by shifting the y^{target} . The error in the k-th learning iteration becomes:

$$\epsilon_k = NMSE(y_k, y^{\text{target}}). \tag{23}$$

Then we compare the error with the error in the previous iteration:

$$r(k) = \begin{cases} 1, & \text{if } \epsilon_k < \epsilon_{k-1} \quad (24) \\ 0, & \text{if } \epsilon_k \geq \epsilon_{k-1} \quad (25) \end{cases}$$

This results in the final learning rule (Brunner et al. (2019), p. 103):

$$W^{out}_k = r(k) \cdot W^{out}_k + (1 - r(k)) \cdot W^{out}_{k-1}. \tag{26}$$

This learning rule "resembles a stochastic gradient descent" (Brunner et al. (2019), p. 103).

3.2.2 Types of noise

Another import characteristic of our system, besides its computing and learning capabilities, is that it includes noise. Semenova et al. (2019) derived an analytical description which we use to implement noise in our simulation.

The noise inside the output signal y can be defined through a noise operator $\hat{N}$ and the noiseless signal y' (Semenova et al. (2019), p. 2):

$$y = y' + \hat{N}(y'). \tag{27}$$

Semenova et al. (2019) differentiates between 4 kinds of noise: additive or multiplicative and correlated or uncorrelated. Correlated noise results from parameters with global impact e.g. voltage of a power supply or temperature. Uncorrelated noise on the other side results from local disturbances e.g. detection of the signal by the camera. Noise can be quantified in our case by (Semenova et al. (2019), p. 2):

$$\hat{N}(y') = \sqrt{2 \cdot D^A} \cdot \xi^A. \tag{28}$$

$$\hat{N}(y') = y' \cdot \sqrt{2 \cdot D^M} \cdot \xi^M. \tag{29}$$

where (28) is additive and (29) multiplicative noise. ξ^A (additive) and ξ^M (multiplicative) are vectors of white gaussian noise with zero mean and variance of one with shape $\xi^A, \xi^M \in \mathbb{R}^{900}$. The constants D^A (additive), D^M (multiplicative) are experimental constants which are obtained from the experiment for the noise found at the individual neuron level. In our system we have additive uncorrelated noise (detector, camera), multiplicative uncorrelated noise (camera), multiplicative correlated noise (SLM) (Semenova et al. (2019)). In the next section, we will describe the simulation. It is important to note that during the computer experiments we have to divide all noise constants mentioned above by 4. Otherwise, the system was not able to learn the one-step-ahead prediction effectivly.

3.3 The simulation

We implemented a simulation of the described experiment using python (version 3.6.8) with the standard libraries numpy (version 1.17.2), matplotlib (version 3.1.1) and scipy (version 1.3.1). In this python simulation, the reservoir is a class with associated functions. Thus, we make use of python's object oriented language design. We generate a new reservoir instance by calling the class constructor with the parameter:

1. inSize: integer, default is 1
2. outSize: integer, default is 1
3. resSize: integer, is restricted to N = 900 because of experimentally obtained coupling matrix

4. initLen: integer, default is 30, determines the transient when computing the reservoir state matrix
5. trainLen: integer, default 200
6. γ: float, determines the input strength
7. β: float, determines the memory of the system
8. θ: float, first sampling point from $\cos(\cdot)^2$
9. δ: float, second sampling point from $\cos(\cdot)^2$
10. μ: float, probability of sampling from first or second sampling point
11. noiselevel: float, default is 1, determines the amplitude of the noise
12. noise: boolean, set the noise on or off.
13. Bias: boolean or numpy array of shape $\mathbb{R}^{N\times 1}$, if boolean, we generate new Bias matrix
14. W_{in}: $\mathbb{R}^{1\times N}$, as default, injection matrix is loaded and reshaped

In this context, self denotes the reservoir instance which has all parameters (1-13) associated to it. We can access all parameters (1-14) by calling self.parametername. In the following, we introduce the class methods we implemented. A class method is strictly associated with the instances of the class and can be called by self.methodname(). We first introduce the name of the method with the expected input and after the colon we give the output. After the output we describe the function in some more detail.

- initialize_reservoir(self): normalized coupling matrix. This method normalizes the coupling matrix by taking the square root and division with the absolute value of the largest eigenvalue.
- make_data(self): normalized data. The method loads the data file and normalizes it to the range of [0,255] taking the minimun and maximum. We first subtract the minimum, divide by the difference between maximum and minimum and multiplication with 255, returns the normalized data of length trainLen.
- generate_ bias(self, n, μ, θ, δ), Bias of size $\mathbb{R}^N$. The method is used if Bias is of boolean type. Then it generates a vector with resSize many entries which are zero or one with probability μ for one and $1-\mu$ for

zero. Then we iterate over this vector and if it is one we sample from θ and if it is zero we sample from $\theta + \delta$. We add to both sampling points a random uniform number from the closed interval $[-5, 5]$ and a constant offset θ_0. The constant offset θ_0 equals the offset phase of the SLM.

- compute_state_matrix(self, time_steps): reservoir state matrix X, is presented as Alg. 1. This method is called with the reservoir instance and an integer number time_steps which is optional and for future work if we want to add drift noise. We initialize reservoir state matrix $X \in \mathbb{R}^{N \times T}$ and the state vector $x \in \mathbb{R}^N$ as zero arrays. The vector x is only used for computations inside this method.

Algorithm 1 compute_state_matrix(self, time_steps):

1: **for** $t = 1, ..., T$ **do**

2: $x \leftarrow \sqrt{2D_A}\xi^A + x(1 + \sqrt{2D_M}\xi^M)$

3: $camera \leftarrow (W^{DOE}\sqrt{x})^2$

4: $arg \leftarrow \gamma W^{inj} u(t) + \beta camera + Bias$

5: $x \leftarrow I^0 \cos^2(\frac{2\pi}{\kappa} arg)$

6: **if** $t \geq Transient$ **then**

7: $X[t, :] \leftarrow x$

return X

We start by iterating over the $T = 200$ points of our MG sequence. In line 2, the analog signal of the optical field between PBS and DOE is simulated by adding multiplicative, uncorrelated and additive uncorrelated noise to the state vector. Then, we convert the state vector x from optical field unit to intensity unit by taking the square root because it is used to compute the camera detection. The DOE is passed twice which is simulated by squaring. This implements Eq. 14. The signal is now detected by the camera and converted into a digital signal. The computation in line 4 is done by weighting the camera detection with β, the injected signal with γ and adding the Bias which is described by Eq. 15. In line 6, we compute the new SLM state. The $\frac{2\pi}{\kappa}$ is a

device related constant and the $\cos(\cdot)^2$ is our generated non-linearity. Then we store the state vector x only if we are over the transient of the signal. After each of the computation 3-4 of Algorithm 1, we cut off decimals and clip the resulting integer value by 255 because of the 8 bit range of the SLM. After the reservoir state matrix is generated, it is associated to the reservoir instance.

- train_reservoir_layer(self, learning epochs): trained output $W_{out} \in \{0,1\}^N$. This method uses the reservoir instance to train the output mapping. The pseudocode is presented as Alg. 2. We load the same initial W_{out} from the experiment, and reshape it, to secure equal conditions in experiment and simulation. Then we generate a W_{Bias} matrix with entries in $[0,1]$. Next, we generate the target data for the learning process. We z-normalize the data. We shift this data by one datapoint in the positive direction by shifting the array index by one since we want to learn a one-point ahead prediction. We start the learning process by entering a while-loop which runs the learning epoch counter in line 1 of Alg. 1. In learning epoch 0, we compute the output of the system by Eq. 22. In the noise condition, we add detector noise $\xi^{A'}$ to the output which represents the detection of the signal from the SLM. We z-normalize the output. Then, the mean square error is computed and the error value is stored in the variable prev_epsilon which will in the following only be updated if the error is lower than its value and is not depicted in Alg. 2. For learning epoch k+1 and all following, we start to change the entries in the W_{out} and generate a new reservoir state matrix (line 6) to simulate the noise in the system by calling the compute_state_matrix(self, time_steps) method. The index to change in the W_{out} is chosen by multiplication of W_{Bias} with random vector with entries in $[0,1]$ which we generate in every learning epoch, multiplication with W_{Bias} gives a vector of shape $\mathbb{R}^N$. We take the index of the maximum entry as the position to change W_{out}. Then, we negate W_{out} at this position and compute the error value. If the error is lower than prev_epsilon, we save its value in prev_epsilon . If is not, we undo the negation of our W_{out} matrix.

Algorithm 2 train_reservoir_layer(self, learning epochs):

while $k \leq$ learning epochs **do**
  if $k == 0$ **then**
    $y \leftarrow W^{0ut} \cdot X + \xi^{A'}$
    $\epsilon_0 \leftarrow MSE(y, y^{target})$
  else if $k > 0$ **then**
    $X \leftarrow self.compute\_state\_matrix(k)$
    $W_k^{select} \leftarrow rand(N) \cdot W_k^{bias}$
    $l_k \leftarrow max(W_k^{select})$
    $W^{bias} \leftarrow 1/N + W^{bias}$
    $W_{l_k}^{bias} \leftarrow 0$
    $W_{k,l_k}^{out} \leftarrow \neg W_{k-1,l_k}^{out}$
    $y \leftarrow W^{0ut} \cdot X + \xi^{A'}$
    $\epsilon_k \leftarrow MSE(y, y^{target})$
    if $\epsilon_k \geq \epsilon_{k-1}$ **then**
      $W_{l_k}^{out} \leftarrow \neg W_{l_k}^{out}$
  $k \leftarrow k + 1$
return epsilon

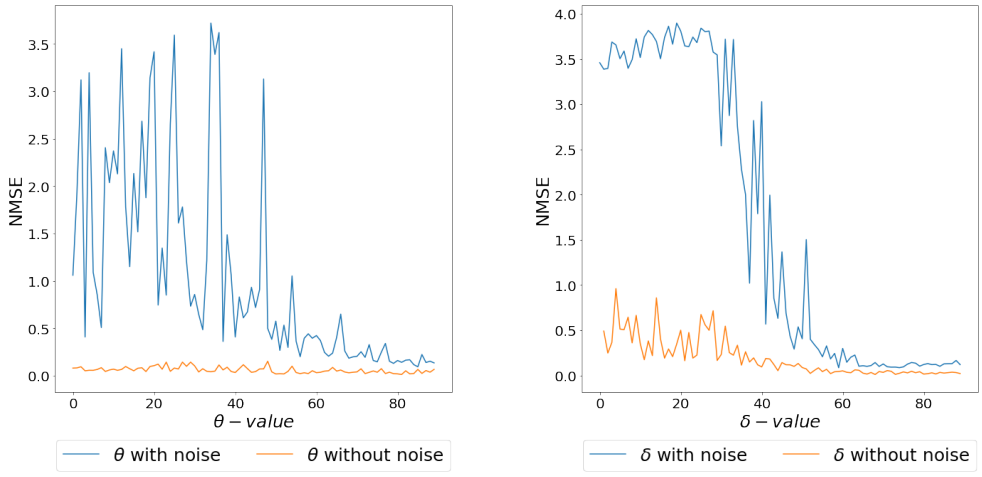
4 Evaluation of the Simulation

The aim of this work was to simulate a photonic recurrent neural network and to compare learning with and without noise. To compare simulation and experiment we compare the optimized parameter values, the NMSE, the bifurcation diagram and the reservoir attractor. We compare learning by considering two different kinds of learning modes.

4.1 Parameter optimization

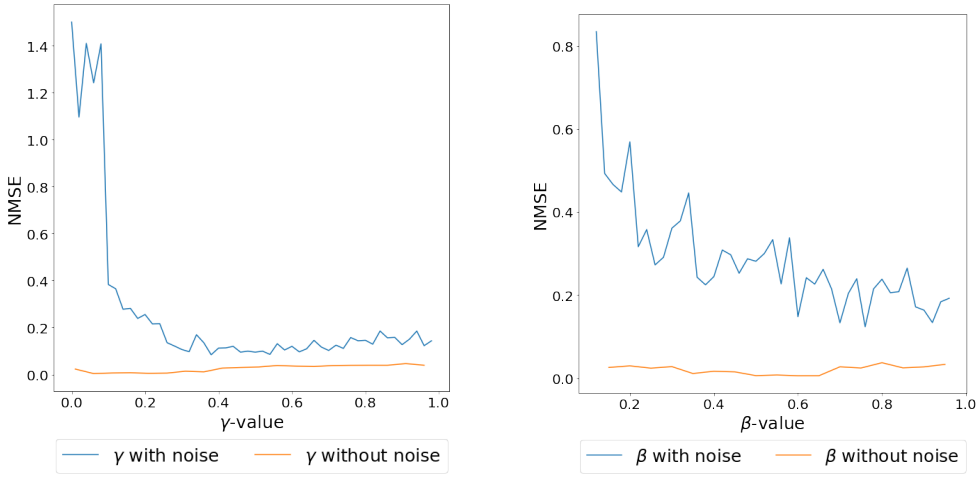
The evaluation of the simulation depends largely on the parameter of the model. In total we have 5 parameters. We have θ value which is the position of the first sampling point for the Bias and δ where $\theta+\delta$ is the second sampling point. Then we have μ which is the probability of sampling from θ and $1-\mu$ is the probability of sampling from $\theta + \delta$. We have the parameter β which weights the memory of our system and the parameter γ which determines the injection strength of the input signal. In the optimization process, one parameter is varied and all the rest are held constant. Thus, we move in one hyperplane of $\mathbb{R}^5$ and compute the NMSE with this parameter. We choose the parameter with the lowest NMSE and this parameter is then used for the rest of the search. We perform the optimization in the following order: θ, δ,

μ, β, γ. We set already optimized parameters to the best found values and initialize parameters with the best found values from the experiment. For the no-noise condition, we scanned every parameter value for 500 learning epoch and for the noise condition for 800 learning epochs. In Fig. 12- 14, we plotted the results of the parameter optimization. The blue curves denote the condition with noise, the orange curves the condition without noise. In Fig. 14 (b), we performed an additional grid search with noise because the system was not able to learn without parameter reconfiguration. The grid search is contrary to the previously performed parameter optimization because now the system finds low β values most desirable.
In the simulation without noise, best parameter values found are $\theta = 81$, $\delta = 68$, $\mu = 0.4$, $\beta = 0.6$, $\gamma = 0.1$.
In the simulation with noise (noise level =0.5), best parameter values found are $\theta = 83$, $\delta = 58$, $\mu = 0.2$, $\beta = 0.8$, $\gamma = 0.4$. This values were updated by the grid search to $\beta = 0.15$ and $\gamma = 0.85$.



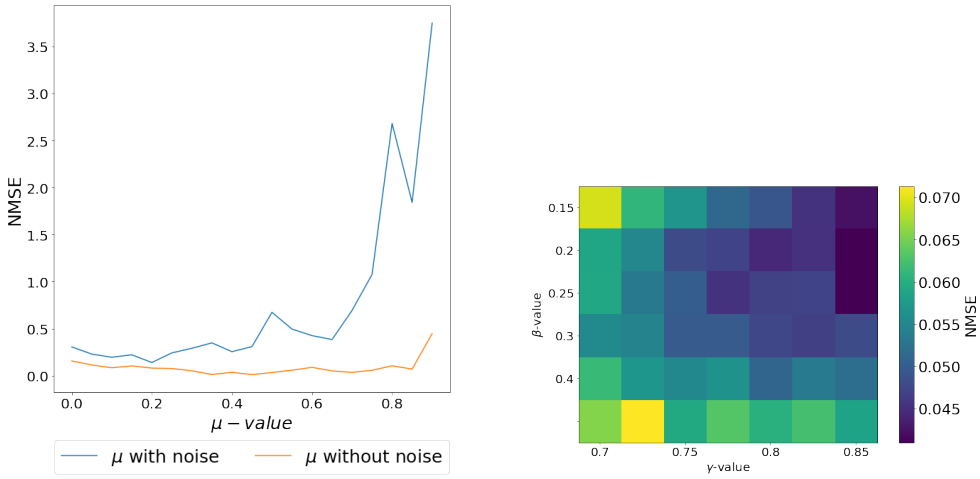
(a) **(b)**

Figure 12: Error with varying theta and delta values with and without noise.



(a) **(b)**

Figure 13: Error with varying beta and gamma values with noise.



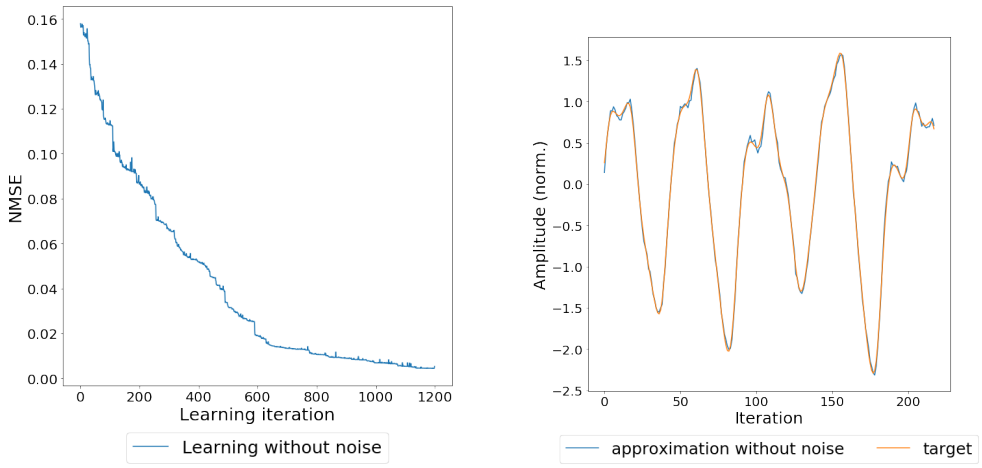
(a) **(b)**

Figure 14: Error with varying μ values with noise and the additional grid search plot.

4.2 NMSE of the experiment and the simulation

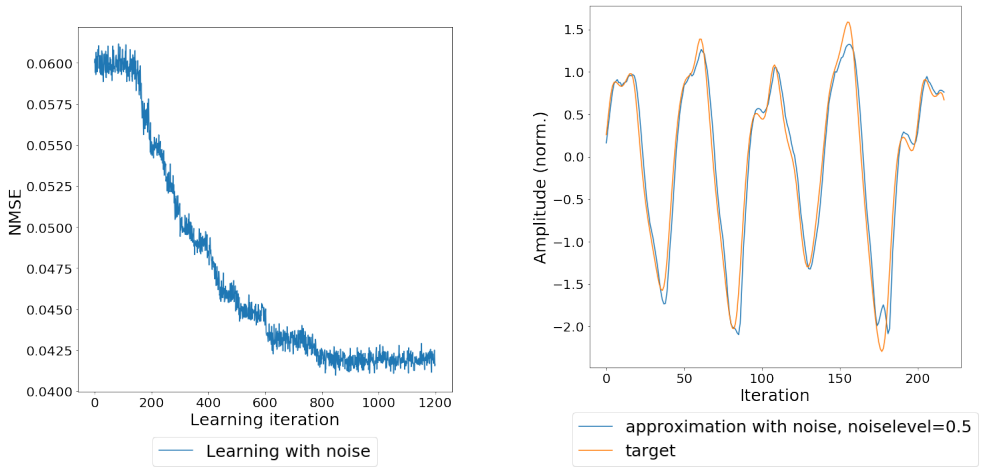
Now we compare the learning curves and the final approximation of the simulation and experiment. In Fig. 15, we plotted the learning curves and the approximation for the no-noise condition and in Fig. 16 for the noise condition. The final error for the condition with noise is 0.042. The final error for the condition without noise is 0.0016. The convergence speed is larger in

the condition without noise. In the simulation with noise, we converge after approximately 900 learning epochs



(a) **(b)**

Figure 15: Learning curves (a) and Approximation (b) for the no-noise condition.



(a) **(b)**

Figure 16: Learning curves (a) and Approximation (b) for the noise condition.

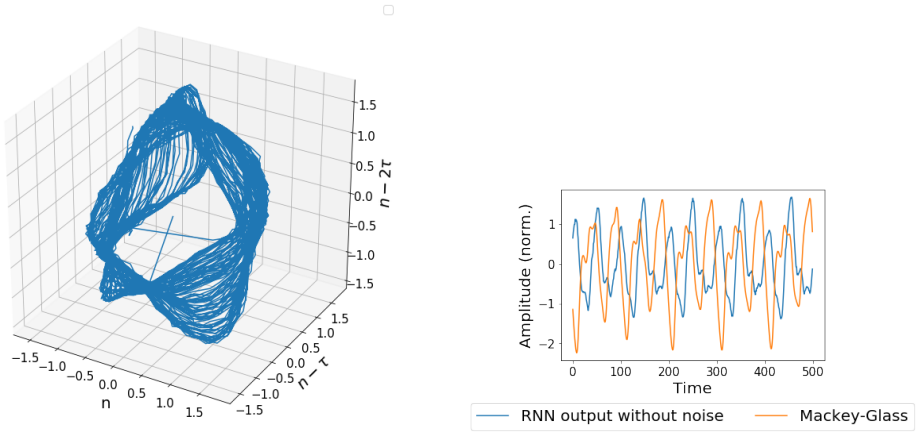
4.3 Autonomous system

An interesting method to evaluate the longterm prediction abilitiies of our system is feedback forcing. We substitute the external input of the time series

with the output of our reservoir. More precisely, we generate a reservoir state matrix X with the original input data and train our W_{out} on this matrix. In the next iteration, we compute the output of the RNN with:

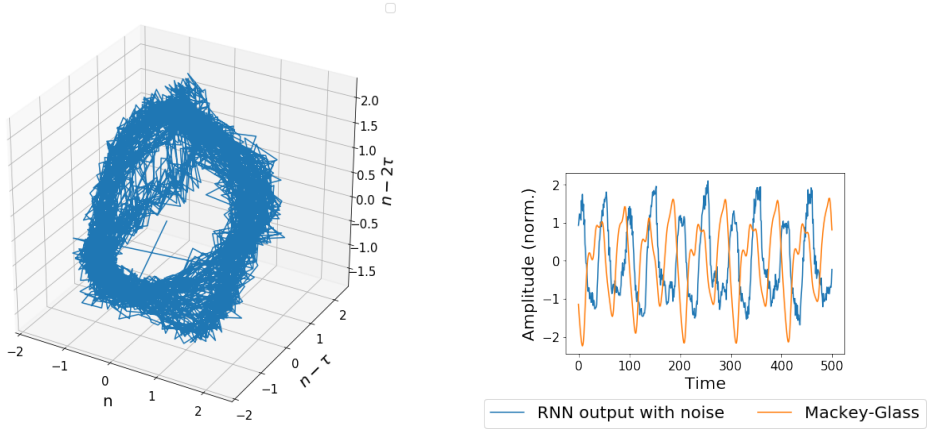
$$Y = W_{out} \cdot X. \tag{30}$$

We generate the next reservoir state matrix with the output Y and do this for 10 iterations in the noise as well as in the no-noise condition. We present the generated attractor by using Taken's theorem. Taken's theorem states that a dynamical system can be recreated from the projection onto its component axes by using a lag time τ (Shalizi (2006), p. 27). We set the lag time equal to the first zero crossing of the autocorrelated signal divided by its length. The resulting attractors are plotted in Fig. 17 (a) without noise and in Fig. 18 (a) with noise. In Fig. 17 (b) without noise and in Fig. 18 (b) with noise we plotted the output Y of the reservoir with the section of the original MG timeseries. If we compare Fig. 17 (a) and Fig. 18 (a) with Fig. 11 (b) we see that the system approximates the topological structure of the target attractor. Furthermore, in Fig. 17 (b) without noise and in Fig. 18 (b) with noise, we plotted a section of the MG sequence and compare it to the output of our reservoir.



(a) **(b)**

Figure 17: The Attractor (a) and the corresponding RNN output without noise in comparison with the original Mackey-Glass sequence (b).



(a) **(b)**

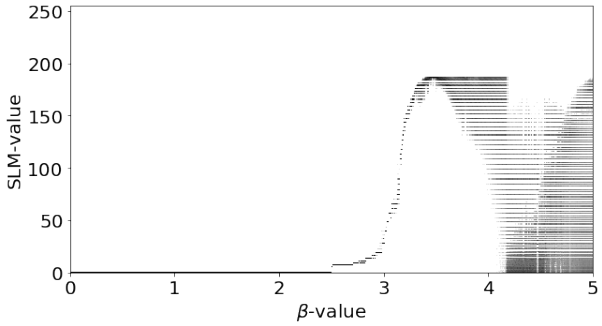
Figure 18: The Attractor (a) and the corresponding RNN output with noise in comparison with the original Mackey-Glass sequence (b).

4.4 Bifurcation diagram of Reservoir activation

Another method to investigate the internal dynamics of our reservoir is to plot the bifurcation diagram. We computed the bifurcation diagram corresponding to the change of beta by choosing one neuron (we chose a neuron in the middle of SLM since activation is largest and most stable), calculating the activation by computing the reservoir state matrix, sorting the neuronal activation into a histogram with bins from 0 to 256 and normalizing by the length of the activation length of 5000. We repeated this while changing the parameter. The parameter γ is set to zero. All other parameters have the best found value. In Fig. 19, we plotted the bifurcation diagram without noise. It is clearly visible that for $0 \leq \beta < 3.2$ one fixed point exists and for $\beta > 3.2$ activation shows chaotic behaviour. In Fig. 20, we plotted the node activation in the same setting with noise. We see two fixed points for $0 \leq \beta < 1$ and one fixed point for $\beta > 1$.

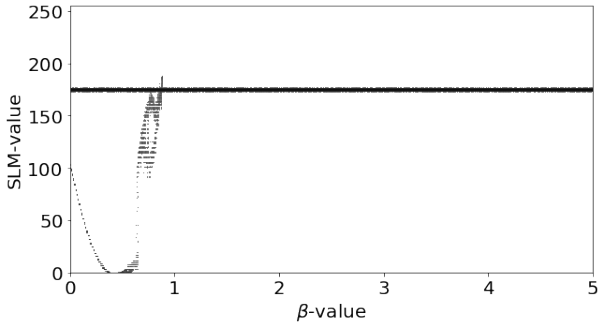
4.5 Readout configuration

Furthermore, we examined the readout configuration of the model with the experiment by comparing two different modes of learning. We used the parameter found by the described procedure at the beginning. We compared the greedy learning method which was used in the paper Bueno et al. (2018) and a "markovian" learning approach. In the "markovian" learning approach each weight of our W_{DMD} was equally likely to be changed in each learning iteration. Another constraint was introduced: we generated one master



(a)

Figure 19: Bifurcation diagram of node 431 without noise.

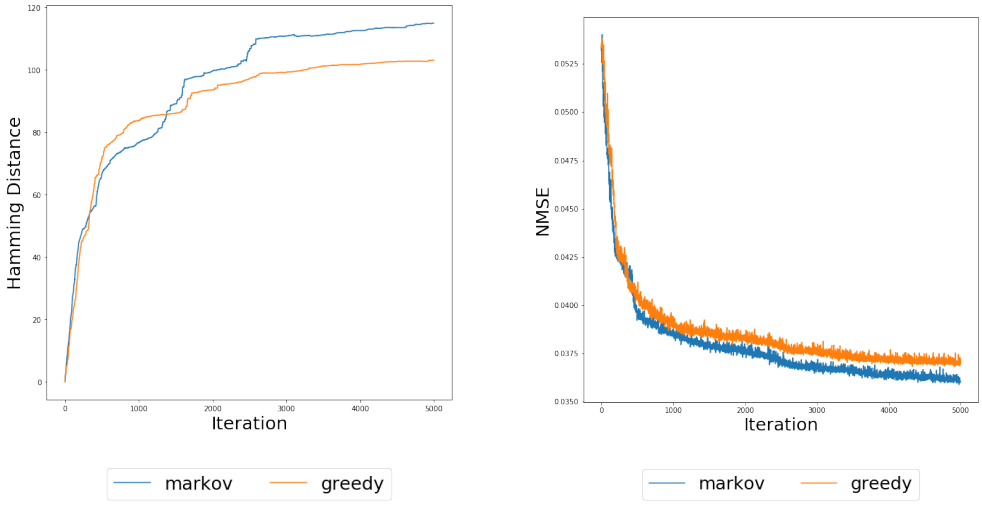


(a)

Figure 20: Bifurcation diagram of node 431 with noise.

learning sequence for each learning mode. The master learning sequences contained the positions (natural numbers between $[0, 900]$) at which W_{DMD} is changed in each training mode. In the n following training epochs we changed weights according to this sequence. Therefore, the algorithm can only reverse the weight changes (negate again since weights are binary). The motivation for this procedure was to eliminate drift noise such as changing room temperature. For the no-noise condition this has no impact because the algorithm follows the master sequence and therefore resulting in identical W_{DMD} configurations. But with our system with noise we observe that W_{DMD} configurations were not identical. We measured this with the Hamming distance. The Hamming distance is defined for binary arrays. It equals the number of entries where the arrays are not identical. We recorded 20 learning curves using one master sequence for the "markovian" and the greedy learning approach. Then we computed the mean hamming distance. The resulting distance functions are plotted in Fig. 21. The greedy learning method converges after approximately 103 weight changes of the W_{DMD} and

NMSE of 0.0375. The markovian learning mode converges after 115 weight changes and a NMSE of 0.036.



(a) **(b)**

Figure 21: The Hamming Distances (a) and the corresponding error curves (b).

5 Genetic Algorithm

5.1 Analysis of the implemented Evolutionary learning algorithm

Previously we described the original evolutionary learning algorithm. This algorithm behaves greedily because it linearly learns away from the previous updated weight by increasing the probability of choosing a weight which has not yet been chosen by $1/N$ in every iteration and setting the probability of weights which have been chosen to 0. Consequently, the entries of our W_{out} are changed uniformly. The error landscape is very high dimensional and depends on the noise. We have in total 900 binary nodes in the W_{out} matrix. Therefore, we have 2^{900} possible W_{out} configurations each associated with an error value. The set of possible W_{out} configurations defines the search space.

To understand the dynamics of the reservoir nodes better, we performed principle component analysis (PCA) on the normalized covariance matrix. We saved the reservoir state matrix for $k = 50$ learning epochs. We normalized the state matrix by taking the mean of every node in every learning epoch

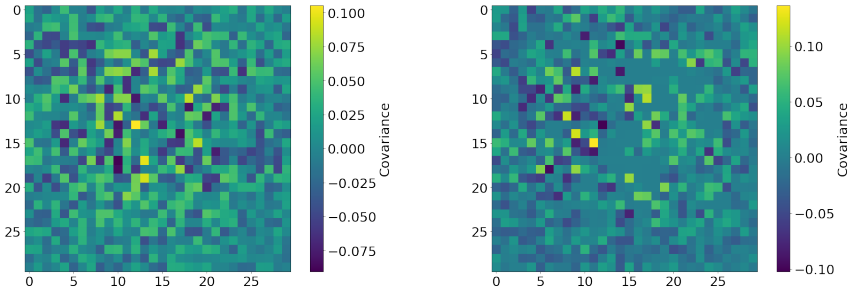


Figure 22: (a) Principle component matrix without noise (a) and with noise (b).

and subtracting the mean activation from the node activation in the specific learning epoch. Finally, we divided each entry of the reservoir state matrix in each learning epoch by the number N of reservoir nodes. Then we concatenated all reservoir state matrix and computed the covariance matrix. From the covariance matrix we computed the principle components. The principle components are the eigenvectors of the covariance matrix and have the same shape as the W_{out} matrix. We sorted after the associated eigenvalues. The eigenvalues denote the variance which is explained by the eigenvector. We plotted the eigenvector with the largest eigenvalue for the no-noise condition is in Fig. 22 (a). As expected, the nodes in the center have the highest covariance. For the no-noise condition in Fig. 22 (b) we observe nodes which have almost 0 covariance in the center. The assumption is that the noise saturates the nodes in the center and then the covariance is averaged out. We infer from the eigenvector in the no-noise condition that we need to make use of the nodes in the centre of the reservoir.

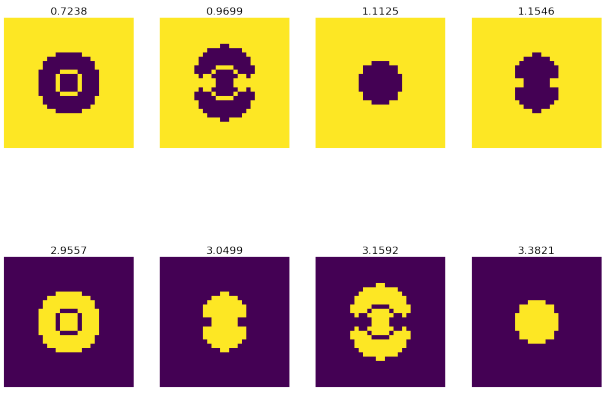


Figure 23: initial chromosome set ordered by error value.

5.2 Genetic Algorithm using the Laguerre-Gaussian modes

The learning rule should take the underlying physical system into consideration e.g. the constraint of our output map W_{out} to be binary. Our approach is to consider the Laguerre-Gaussian modes from Fig. 23. The Laguerre-Gaussian modes are a natural way to represent the intensity distribution of an optical system (Goubau and Schwering (1961)). In this setting, they serve as a heuristic starting point to minimize the number of W_{out} configuration to check for the optimization and, as we saw in the correlation matrix, the influence of nodes on the prediction error depends on their spatial position. The idea is to use a genetic algorithm approach to learn the spatial configuration of the W_{out} matrix. A genetic algorithm is a combinatorial optimization algorithm which uses the mechanisms of genetics (Ashlock (2006)). In this setting, a chromosome is a possible solution to the optimization problem. The key advantages of a genetic algorithm are the ability to make use of world knowledge which for our problem results in using the Laguerre-Gaussian modes as chromosomes and to escape local minima (Ashlock (2006), p. 35). Additionally, we don't need to define a neighbourhood which would be the case for other combinatorial optimization algorithms such as simulated annealing. Genetic Algorithm have been used for optimizing neural network weights and their topologies (Stanley and Miikkulainen (2002)). We can implement a genetic algorithm in the following steps:

1. create initial population of minimal Laguerre-Gaussian modes (select a number of low-order Laguerre-Gaussian modes)
2. evaluate over NMSE-error
3. networks are ranked according to their performance and choose best networks to recombine
4. recombine and mutate chromosomes at random
5. select next generation
6. repeat 2 - 5 until stop criteria is met

5.3 Crossover and mutation

We will now describe the algorithm in more detail. As there exists analytical formulas for the Laguerre-Gaussian modes we implemented these and binarized the resulting images. By this procedure, we initialize the algorithm by generating a starting population of size init size. Now our population

consists of binary Gauss-Laguerre chromosomes of size 900. In all further iterations, we partition the population in elite chromosomes and the average chromosomes. The elite set is defined as a number of chromosomes with lowest NMSE. We generated the elite set by computing the NMSE of all chromosomes and then sort the chromosomes by starting with the lowest NMSE. Next, we apply the crossover operator. The crossover operator takes two parent chromosomes, one or more crossover points and outputs a child chromosome. We define a single point crossover operator by generating a random number between $[0, 900]$ at which two chromosomes were crossed with each other. This means the child chromosome has the values of parent A from the beginning to the crossover point and from the crossover point until the end the child's entries are the values of parent B. The two parent chromosomes to cross were chosen at random from the whole set. The size of the average population determines how many times we apply the crossover operator while the elite chromosomes are not affected by the crossover operator. The elite set is copied to the next generation but used for crossover in every learning epoch. This procedure introduces two parameters: the elite size and the average population size.
Furthermore, we apply the mutation operator. The mutation operator iterates over each chromosome in the average population and negates an entry if a sampled number from a standard gaussian distribution is smaller than the mutation rate. Additionally, the mutation operator introduces two more parameters: the mutation rate and the decay factor. We have in total 5 new parameters introduced:

1. init_size: integer, number of initial chromosomes
2. avg_pop_size: integer, size of average population
3. elite_pop_size: integer, size of elite population
4. mut_rate: float in $[0, 1]$, determines how many entries of each chromosome from avg_pop are mutated
5. decay_factor: float in $[0, 1]$, determines the decay of mutation rate

We evaluate every generation of chromosomes by computing the mean error over the elite chromosomes in every iteration.

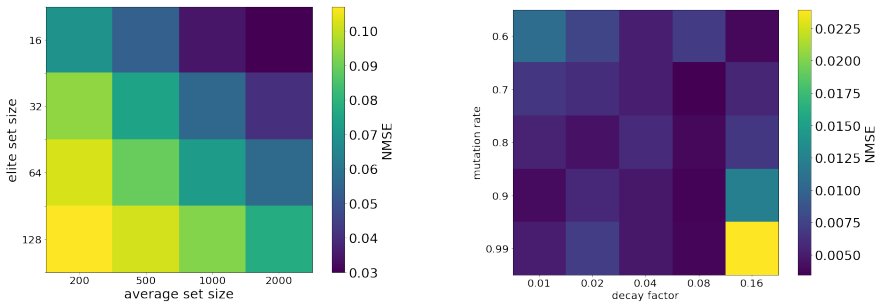
6 Evaluation of the Genetic Algorithm

In total we now have 10 parameters but we will use the already tuned parameters from the evolutionary learning task and focus on the 5 new pa-

rameters from the genetic learning approach. We chose the initial_size of 8 for our consideration here. We start to analyse these parameters by grid search through the parameter space only for the no-noise condition because of time and hardware constraints. Secondly, we present the final error for the noise and no-noise condition. Thirdly, we explore learning with initial Gauss-Laguerre modes and learning with initial random matrices.

6.1 Parameter optimization

We perform grid search by splitting the parameter space in the elite-average set hyperplane and decay-mutation rate hyperplane. First we search the elite-average set hyperplane by setting the decay rate to 0.9 and the mutation rate to 0.03. After, we search through the decay-mutation rate hyperplane by setting the first two parameters constants to the best found values. The underlying assumption is that the decay-mutation-rate plane and the elite-average size plane are mostly independent for a good final error result. In Fig, 24 represents the result of the grid search is presented. In Fig. 24



(a) **(b)**

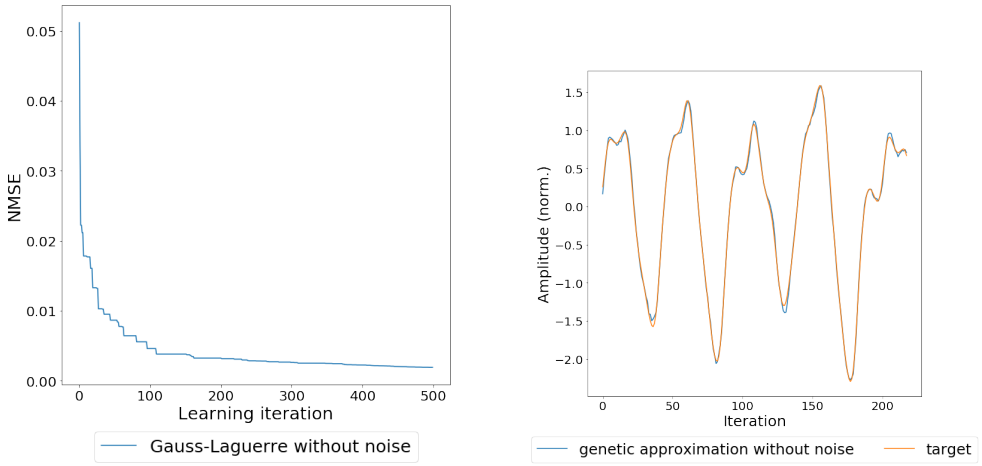
Figure 24: (a) correlation matrix with noise and (b) the correlation matrix without noise.

(a), we chose $200, 500, 1000, 2000$ for the average size and $16, 32, 64, 128$ for the elite size. The system found the average population size of 2000 and an elite size of 16 most desirable. In Fig. 24 (b), we chose decay factors $0.01, 0.02, 0.04, 0.08, 0.16$ and mutation rate of $0.6, 0.7, 0.8, 0.9, 0.99$. The decay factor of 0.08 turned out to be most desirable here.

6.2 NMSE of the genetic algorithm

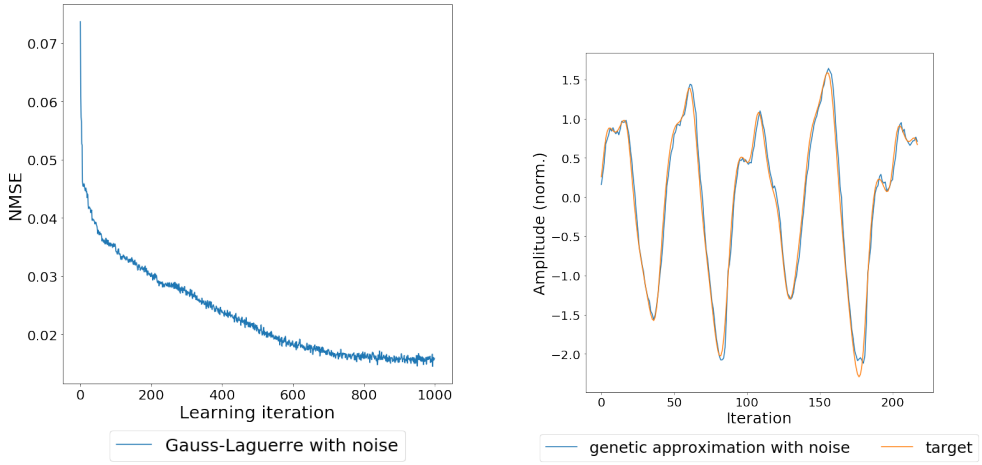
We compare the learning curves and the final approximation of the genetic algorithm for the noise and no-noise condition. In Fig. 25, we plotted the

learning curves and the approximation for the no-noise condition and in Fig. 26 for the noise condition. Because of time and hardware constrainsts, we chose elite set size of 16 and average set size of 200 and T=1000 learning epochs. The decay rate was set to 0.08 and the mutation rate of 0.99. The final error for the condition with noise is 0.015. The final error for the condition without noise is 0.0016. The resulting are plotted in Fig. 24 and Fig. 25.



(a) **(b)**

Figure 25: Learning curves (a) and Approximation (b) for the no-noise condition and the genetic algorithm.



(a) **(b)**

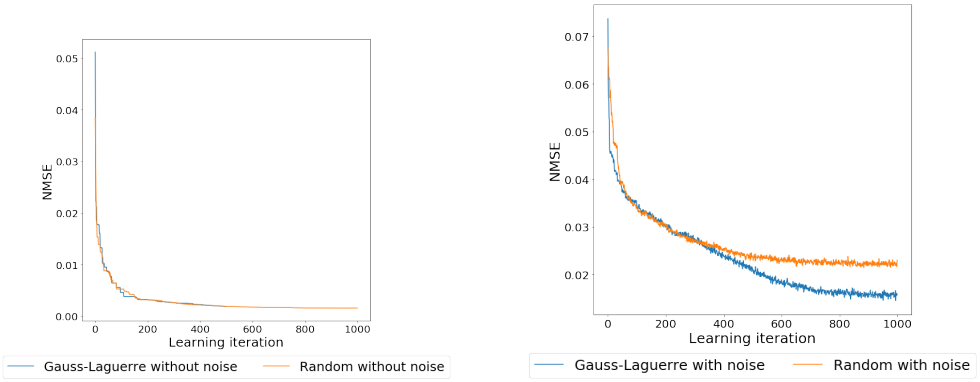
Figure 26: Learning curves (a) and Approximation (b) for the noise condition and the genetic algorithm.

6.3 Comparison between Gauss-Laguerre population and random population

We initialized a random population of 8 W_{out} matrices and compared their learning succes and convergence to the previously presented Gauss-Laguerre initialized population with and without noise. We chose the same parameter set as in the previous section. We perform 1000 learning epochs. The Gauss-Laguerre initialized population performed better in the noise condition. In the noise condition the random population converged with a final error of 0.023. In the condition without noise the random population performed better than the Gauss-Laguerre initialized population with a final error of 0.0015.

7 Discussion

In this thesis, we simulated a photonic neural network with and without noise. In the introduction we stated that the aim of this work is to simulate a photonic neural network and to compare it to the actual experiment with and without noise. Furthermore, we introduced a genetic algorithm to generalize the learning rule which the system uses. We will first discuss the performance of the simulation and compare it to experiment by refering to Brunner et al. (2019). Than we will discuss the genetic algorithm. In both parts of the discussion, we willl focus on 3 main points: parameter optimiza-



(a) **(b)**

Figure 27: Learning curves (a) for the no-noise condition and (b) for the noise condition.

tion,convergence and the readout configuration.
We start our discussion of the parameter optimization for the simulation. The aim of the parameter optimization process was to find the parameter settings which would recreate the behavior of the experiment and allow optimal performance. In the experiment the optimal parameter values obtained were $\theta = 42$, $\delta = 64$, $\mu = 0.45$ (Brunner et al. (2019), p. 104), $\beta = 0.8$, $\gamma = 0.4$ (Brunner et al. (2019), p. 105). We reconsider the equation:

$$x_i^{SLM}(t+1) = I_i^0 \cdot \cos^2\left(\beta \cdot x_i^C(t) + \gamma \cdot W_i^{inj} \cdot u(t+1) + \theta_i\right), \qquad (31)$$

which we used to compute the state of each neuron in our system. The parameter γ controls the injection strength of the input signal. If $\gamma = 0$, the system is only influenced by its previous computation and thus an iterated map of the $\cos(\cdot)^2$ non-linearity scaled by the illumination intensity. For $\gamma = 1$, the input is fed into the system with maximal amplitude. This results in strong perturbation of the reservoir state matrix. In the case of the noise condition the optimal value was twice as high than the value from the experiment. In the no-noise condition of the simulation its value was by a factor of 0.25 lower than what we expected from the experiment. An explanation is that the system chose a very low beta value in the noise condition and thus tried to compensate with a comparatively high γ value. This implies that the reservoir state matrix is merely a result of the input signal and less a result of previous computations. Thus, strong perturbations occcur. A hypothesis is that this effect becomes more apparent when applying higher noiselevel.
Furthermore, we investigated th parameter β which controls the feedback gain. For $\beta = 0$ the system can no more rely on previous computations and

thus has no memory. For $\beta = 1$ the system is in most cases close to the chaotic regime. In the noise condition it was by a factor of 0.18 lower than the experimental value. An explanation for this can be found from the bifurcation diagram of Fig. 20. We conclude from it that the system saturates immediately because of the limitation of the SLM to the range of $[0, 255]$. The node's activation is clipped to 255 if it exceeds this range. Thus, the dynamics of the nodes is stabilized which limits the approximation capabilities of the system. Therefore a low β value is chosen to avoid the saturated state. The value of the parameter β in the no-noise condition was close to the experimentally found value. Additionaly, the bifurcation diagram of Fig. 20 shows the same bifurcation as generated by (Brunner et al. (2019), p. 100) where the node activation falls in the chaotic regime for $\beta > 1.2$. Thus, the obtained data about the parameter β suggest that it is very sensitive to noise and it recreates the experimentally found behavior better without noise. An interesting implication of this finding is that the impact of noise on the experiment is also low. Future research should investigate how the bifurcation diagram of the β parameter is influenced by a varying noiselevel. We mentioned in the section about the experiment that θ and δ constitute our bias term. The bias was constructed such that the nodes of the system can operate along the positive and the negative slope of the nonlinear function $\cos(\cdot)^2$ and thus increase the approximation capabilities of the system. The system is not able to approximate the signal if the bias is not optimally tuned. The nonlinear function was a result of the SLM phase. In both conditions, we obtained values for θ and δ which were much higher then the experimental condition. Thus, the system found the optimal offset independent of the noise. This suggests further that the SLM approximation in our simulation is not comparable to the experimental condition. To improve the SLM approximation we can install drift noise which accounts for e.g. temperature fluctuations. Drift noise results in a very slow, oscillatory drift of the bias offset. This value is the same for every neuron.

Another critical parameter is the probability which determines on which sampling point the system relies. Bueno et al. (2018) showed that it optimal value is obtained for = 0.45 because of symmetry breaking. In the noise condition of the simulation optimal is found at 0.2. Thus relying mainly on the first sampling point. An explanation could be that the system looks for SLM pixels which are not saturated. In the no-noise condition, the system finds = 0.4 which is very close to the experimentally found value. This result again suggests that the impact of noise is low on the experiment.

We control the injected noise into the system by the noiselevel parameter. When we obtained the results it was very difficult to determine a noise parameter on which the system could still evolve because it influences all parameters

simultaneously. If the noise parameter is set to one the system is saturated and does not learn anymore. This is also the case with the noiselevel = 0.5 as can be seen in the bifurcation diagram in Fig. 20. The noiselevel which describes the experiment best is still missing. The data suggest that it should be very low.
We will now continue our discussion with the final error result and the convergence of the simulation. The final error from the experiment is 0.013. The minimum is reached in the experiment after approximately 900 learning epochs. (Brunner et al. (2019), p. 105). For the simulation, we used all parameters from the previoulsy discussed parameter optimization. The convergence of the simulation in the no-noise condition was faster and and the error lower than in the condition with noise and in the experiment which makes the noise condition more comparable to the experiment. This suggest that noise exists. In the noise condition the system learns very slowly and the final approximation error was by a factor of 2 higher than in the experiment. We obtain an already low initial error by optimally tuning the bias off approximately 0.1. The system can only improve this error value by 0.05. We conclude that the saturation of the SLM decreases the system approximation capabilities in the noise condition. Furthermore, we investigated the longterm approximation capabilities of our system by computing the feedback forced attractor. The approximation of the attractor suggests that the system learned the longterm dynamics of the timeseries independent of the noise condition. (Brunner et al. (2019), p. 111) shows very similar results. From a visual inspection, the attractor produced by the noise condition is more similar to the experimentally obtained attractor than the attractor from the no-noise condition which confirms that the simulation can recreate the experimental results.
We compared two modes of learning with noise. The greedy learning mode and the markovian learning mode which allowed the system to randomly chose a weight to switch. From this learning modes we generated a master sequence which the system had to follow and it could only reverse the weight switch. Under experimental conditions the two learning methods converged after a hamming distance of approximately 450. In the simulation the system was saturated and thus the node dynamics were stabilized. Thus, the system learns until it reaches a saturated state and from there on it only reverses the weight switch.
We will now discuss the proposed genetic algorithm which works on a set of W_{out} matricies and recombines them with the point crossover and mutation operator. The genetic algorithm introduced 4 more parameters. We optimized them to outperform the evolutionary learning algorithm. The two parameters which control the size of the population: average_size and

elite_size and the mutation rate and the decay factor which determine the convergence rate of the algorithm. A large average size increases the chance that the population will generate a chromosome which reduces the final error. For the elite set size it is the other way around. A large elite size impairs the search through the space of possible W_{out} configurations because once the population is dominated by a few chromosomes these will produce more children in further iterations which may not include the optimal solution. A low number of elite chromosomes, on the other hand, could "wash out" important chromosomes from the search space. The mutation rate determines how much random change we apply to our population. For example, if we start with a mutation rate of e.g. 0.03 then in the first learning epoch, on average 30 entries in each chromosome is changed. The decay factor slowly decreases the mutation rate to enable convergence. The evaluation of the size parameter showed that for the mutation rate a value from 0.7 to 0.99 performance best with a decay factor of 0.08. A key question is to find an appropriate population size. Since the optimal solution is unknown we can only try further parameter settings and compare their outcome to other algorithms which is what we will do in the following.
The proposed genetic algorithm did perform equally to the evolutionary learning approach. In the no-noise condition the genetic algorithm had a final error of 0.0016. One problem which arises when using the Gauss-Laguerre modes as chromosomes is that the mean hamming distance of the initial population is 450. In this case a genetic algorithm can not evaluate the error landscape well. We compared the Gauss-Laguerre initialized population to a randomly initialized population with the same number of chromosomes. The final error result was even better for the random population and it did outperfom the evolutionary learning. Thus, the performance of the Gauss-Laguerre initialized population can be further increased by mixing the Gauss-Laguerre modes with randomly initialized W_{out} chromosomes. The optimal ratio of chromosomes from both population must be subject to future work.
Maybe a genetic algorithm can also help to charactize the error landscape by looking at the convergence speed and the final W_{out} population. Possibly, a clustering algorithm could be applied to classify the W_{out}. population.

8 Conclusion

In this thesis, we examined a hardware implementation of a neural network. We investigated the physical laws and the neural network architecture which governs it. As adressed in the introduction, we presented a simulation to show the effect of noise on a photonic neural network. The results show that

the simulation recreates the experimental conditions. Furthermore, we succesfully introduced an algorithm which improves the learning of the system under the given constraints. The simulation and the proposed learning algorithm can be used to further investigate the influence of noise and to analyse the underlying error landscape.

9 Appendix

9.1 Noise constants

1. $D^A = 0.11$, additive noise constant
2. $D^M = 0.00005$, multiplicative noise constant
3. $D^{MC} = 0.0001$, multiplicative correlated noise constant
4. ξ^A, additive, uncorrelated noise vector (mean 0, standarddeviation 1)
5. ξ^M, multiplicative, uncorrelated noise vector (mean 0, standarddeviation 1)
6. $\xi^{A'}$, additive, uncorrelated detector vector (mean 0, standarddeviation 0.0008)

References

Daniel Ashlock. *Evolutionary computation for modeling and optimization*. Springer Science & Business Media, 2006.

Oussama Benrhouma, Houcemeddine Hermassi, Ahmed A Abd El-Latif, and Safya Belghith. Chaotic watermark for blind forgery detection in images. *Multimedia Tools and Applications*, 75(14):8695–8718, 2016.

Daniel Brunner, Bogdan Penkovsky, Bicky A Marquez, Maxime Jacquot, Ingo Fischer, and Laurent Larger. Tutorial: Photonic neural networks in delay systems. *Journal of Applied Physics*, 124(15):152004, 2018.

Daniel Brunner, Miguel C Soriano, and Guy Van der Sande. *Photonic Reservoir Computing: Optical Recurrent Neural Networks*. Walter de Gruyter GmbH & Co KG, 2019.

Julian Bueno, Sheler Maktoobi, Luc Froehly, Ingo Fischer, Maxime Jacquot, Laurent Larger, and Daniel Brunner. Reinforcement learning in a large-scale photonic recurrent neural network. *Optica*, 5(6):756–760, 2018.

Ulrich Drepper. What every programmer should know about memory. 2007.

Nabil H Farhat, Demetri Psaltis, Aluizio Prata, and Eung Paek. Optical implementation of the hopfield model. *Applied optics*, 24(10):1469–1475, 1985.

Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. In *European conference on artificial life*, pages 588–597. Springer, 2003.

Richard P Feynman, Robert B Leighton, and Matthew Sands. *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*, volume 1. Basic books, 2011.

George Goubau and F Schwering. On the guided propagation of electromagnetic wave beams. *IRE Transactions on Antennas and Propagation*, 9(3):248–256, 1961.

James E Harvey and Richard N Pfisterer. Understanding diffraction grating behavior: including conical diffraction and rayleigh anomalies from transmission gratings. *Optical Engineering*, 58(8):087105, 2019.

Kensuke Ikeda. Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system. *Optics communications*, 30(2):257–261, 1979.

H. Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007. doi: 10.4249/scholarpedia.2330. revision #189893.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

Laurent Larger, Miguel C Soriano, Daniel Brunner, Lennert Appeltant, Jose M Gutiérrez, Luis Pesquera, Claudio R Mirasso, and Ingo Fischer. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics express*, 20(3):3241–3249, 2012.

Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4):593–616, 2004.

Sheler Maktoobi, Luc Froehly, Louis Andreoli, Xavier Porte, Maxime Jacquot, Laurent Larger, and Daniel Brunner. Diffractive coupling for photonic networks: how big can we go? *IEEE Journal of Selected Topics in Quantum Electronics*, 26(1):1–8, 2019.

Ruediger Paschotta. *Beam Splitters*, "accessed April 3, 2020"a. URL `https://www.rp-photonics.com/beam_splitters.html`.

Ruediger Paschotta. *Beam Splitters*, "accessed April 3, 2020"b. URL `https://www.rp-photonics.com/paraxial_approximation.html`.

Richard S Quimby. *Photonics and lasers: an introduction*. John Wiley & Sons, 2006.

Nadezhda Semenova, Xavier Porte, Louis Andreoli, Maxime Jacquot, Laurent Larger, and Daniel Brunner. Fundamental aspects of noise in analog-hardware neural networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(10):103128, 2019.

Cosma Rohilla Shalizi. Methods and techniques of complex systems science: An overview. In *Complex systems science in biomedicine*, pages 33–114. Springer, 2006.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995. ISSN 0163-5964. doi: 10.1145/216585.216588. URL `https://doi.org/10.1145/216585.216588`.

Declaration of Authorship

I, Paul Liebenow, hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

.....

signature

.....

city, date