On-Chip Photonic Recurrent Neural Networks for Time Series:
A Dynamical Exploration and Application Search

MSc Thesis

by

Bastiaan Ketelaar

Student Number:	4592174	
Thesis advisor:	Dr. R.A. Norte,	TU Delft, 3mE faculty
	Dr. M.A. Bessa,	Brown, School of Engineering

ABSTRACT

Artificial intelligence has a strong need for faster and more energy-efficient solutions, especially for computation performed at the sensor edge. On-chip photonic neural networks (PNNs) offer a promising solution for high speeds and energy efficiency. A less explored side of PNNs is their application to time-series data, which is often the case for real-world sensor applications. While PNNs promise high speeds and energy-efficient solutions, no good use cases have been proposed. This report will first review the state of the art of PNNs. It will be seen that to solve time-series tasks, photonic continuous-time recurrent neural networks (CTRNNs) are required. The dynamics of CTRNNs are thoroughly explored to leverage obtained insights to recommend novel practical applications. This is done through simple examples, and applied to a real machine learning task. It was seen that on a real classification task the network learned two distinct fixed points corresponding to the classes. A link between the time constant of the continuous-time neurons and the temporal dynamics of the task is also found. Two general directions for novel applications are then proposed. Firstly, photonic PNNs can be slowed down to match the task. Opto-electronic PNNs allow for more control of the time constant, and on-chip photonic filter neurons are suggested. Secondly, the high speeds of photonic neural networks can also be directly leveraged. Extremely fast convergence of photonic CTRNNs can be utilized for Modern Hopfield networks, pathfinding algorithms, and time-dependent optimization problems such as for example Model Predictive Control.

Contents

List of Figures

List of Tables

1

INTRODUCTION

The rapid advancement of artificial intelligence (AI) has necessitated the development of faster and more energy-efficient computational solutions. Particularly at the sensor edge where often real-time data processing is critical, novel computing paradigms can be of interest [1], [2] . Traditional electronic neural networks, while powerful, often fall short in meeting the stringent speed and energy efficiency requirements of modern applications. This has led to an increasing interest in photonic neural networks (PNNs), which leverage the unique properties of light to perform computations at unprecedented speeds with lower energy consumption [3].

On-chip PNNs have emerged as a promising technology capable of addressing these challenges. PNNs can perform complex neural network computations with high throughput and minimal power dissipation. Despite the potential advantages, the application of PNNs to time-series data—a common requirement in real-world sensor applications—remains underexplored.

This report aims to answer the following question: 'Can we find applications for on-chip photonic neural networks for time series that naturally couple the benefits of photonic neural networks to the learning task?'. This is divided into the following two sub-questions: 'What are the dynamics of photonic recurrent neural networks for time series, and how can we understand them?' and 'How do we use the insight from the understanding of the dynamics to match with applications?'.

Chapter 2.1 will review the state-of-the-art in the field of on-chip photonic neural networks with a focus on the desired on-chip time series application. Chapter 2.2 will review the influence of the time constant in continuous time recurrent neural networks (CTRNNs) with respect to the learning task, and Chapter 2.3 will discuss what has been learned, what gaps are in the literature, and will address the research question. Chapter 3 explore the dynamics of CTRNNs examining the relation between the internal dynamics and the input. Chapter 4 introduces several tasks and benchmarks the CTRNNs against a standard RNN. On of the tasks is examined more in depth in order to find out what and why the network learns, how this couples to the inputs, and how this can inspire novel applications. Chapter 5 concludes the research with a discussion and conclusion.

2

State of the Art

2.1. (On-chip) Photonic Neural Networks

This section will cover the current literature on photonic neural networks. First, it will describe the basics of a neural network and how these components can be made photonically 2.1.1. Then photonic neural networks for time series, in particular, are discussed 2.1.2. Lastly, a short discussion will be given on how the photonic neural networks, in particular the relation to time, relate to practical tasks 2.1.2.

2.1.1. An Introduction to Photonic Neural Networks

Feedforward Photonic Neuron

The fundamental building block of all neural networks is the neuron. The most commonly used in machine learning is the simple feedforward model of the neuron. Despite the existence of more neurologically correct models (such as spiking neural networks), this simple model is the basis of most artificial neural networks (ANNs). Building a photonic neural network (PNN) starts with three basic operators, namely: multiplication of inputs by weights, the addition of the inputs (and possibly a bias), and a nonlinear activation function. An introduction to certain machine learning concepts can be found in append A.

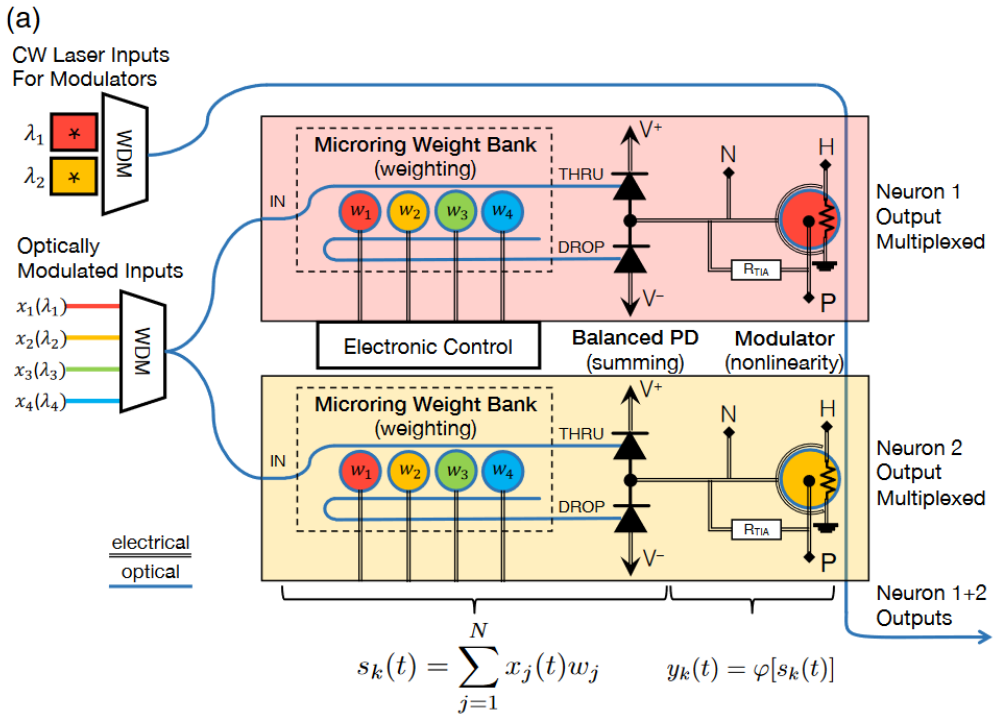
Multiplexing, Weighing, Summing

In broad terms, an artificial neuron can be split up into two main components: a linear- and a nonlinear part. This section will focus on the linear part which consists of the input to the network, the weighting for each input, and the addition of the summed input (and possibly an additional bias).

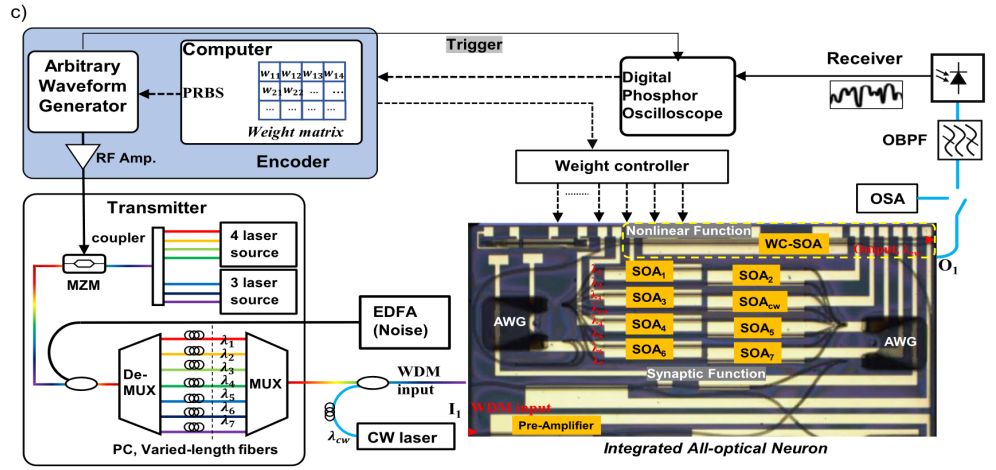
Light possesses many degrees of freedom that can be leveraged for computation. PNNs can be classified by their multiplexing techniques. The main approaches are wavelength division multiplexing (WDM) and space division multiplexing (SDM).

In WDM each input to the NN has its wavelength of light. The inputs are then multiplexed onto a single waveguide. Weighing of the inputs can be done in different ways. Mourgias et al. weighed their inputs with optical attenuators before the multiplexing stage [4]–[6]. Shi et al. weighed the inputs using semiconductor optical amplifiers (SOAs) for each input [7]–[9]. The broadcast and weight (BnW) technique uses micro ring modulators (MRMs) tuned to the specific wavelengths present in the inputs to perform the weighing operation [10]. The total power of the weighted input signals then corresponds to the sum of these signals. If not already, the signals are multiplexed onto the same waveguide where some device detects the total power[4], [7], [11]. WDM allows the reduction of the physical footprint since all signals are present on a single waveguide instead of spatially separated. Light does not self-interact which makes it possible to multiplex many frequencies on a single waveguide without reducing the bandwidth. However, Shi et al. mention noise due to optical cross-talk when implementing WDM via arrayed waveguide gratings (AWG) on-chip [9]. Further, WDM requires a separate wavelength for each input, and also for the output, of each neuron. While on-chip light sources exist, a practical

implementation of a larger on-chip PNN remains to be demonstrated. A demonstration using on-chip light is given by Shi et al. who used an Indium Phosphite (InP) platform to implement an on-chip laser for PNN purposes [8]. Another demonstration is given by Feldman et al. who use a laser and a soliton frequency comb as light source on-chip [12]. Although integrating on-chip light sources would enhance the feasibility of WDM techniques, it is uncertain whether such on-chip light sources can effectively enable larger on-chip PNNs at the present moment.



(a) Shows the broadcast and weight method using OEO nonlinear activation for two neurons [13].



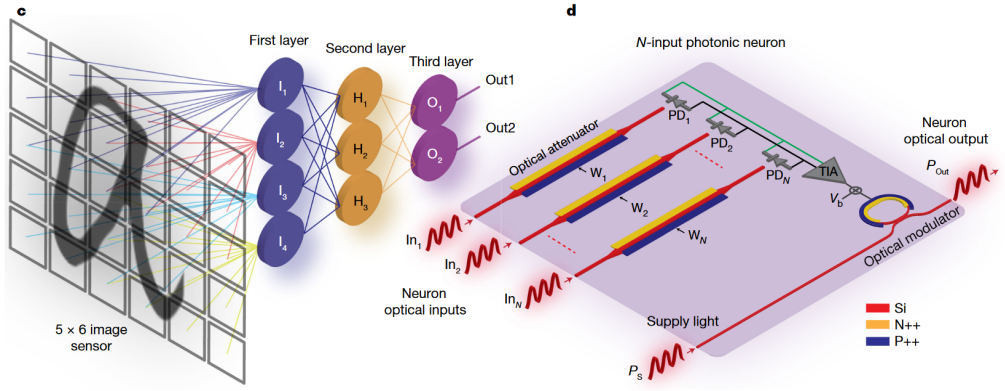
(b) Shows a neuron using AO nonlinear activation on an InP platform including the experimental setup [14].

Figure 2.1: Selection of wavelength division multiplexing (WDM) methods.

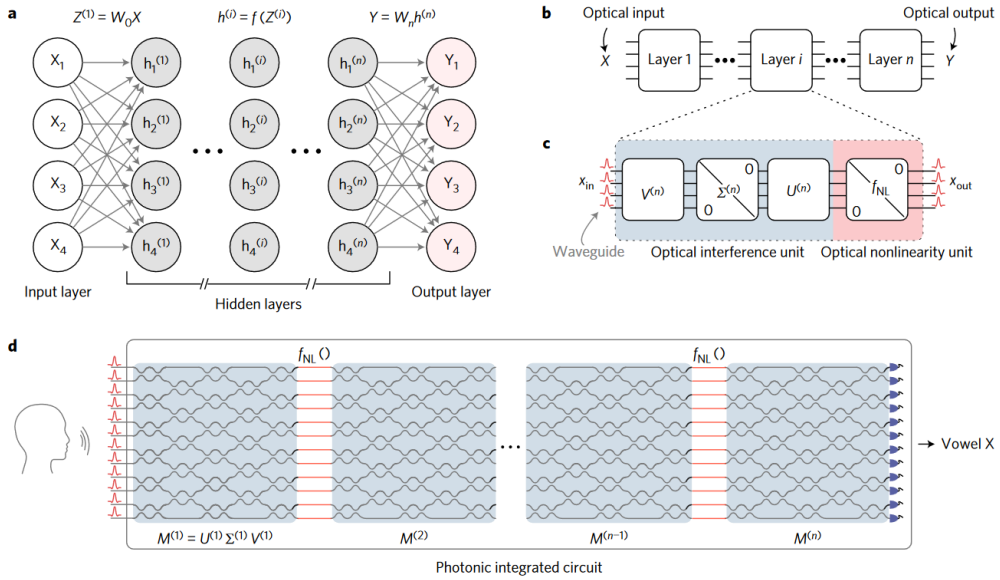
Space division multiplexing (SDM) techniques leverage some spatial features of light to perform weighing and addition. This category holds Fourier optics, diffractive units, MZI-meshed, and a few other methods that do not belong to a general class [15]. Fourier optics will not be further investigated

since these are often very bulky. The first on-chip demonstration of SDM is a MZI mesh which uses a collection of phase shifters and beam splitters to perform vector-matrix multiplication. MZI meshes are commonly constructed by applying singular value decomposition to the matrix formalism used in NNs where the component matrices can be physically implemented using photonic components [16], [17]. Due to the spatial distribution of the MZI mesh a larger physical footprint than the WDM techniques is required. Also, the MZI mesh requires the light to be coherent. Another approach is based on multiple plane light conversion (MPLC), such as diffractive and scattering-based methods. These demonstrations show by far the greatest number of neurons due to the many tuneable parameters, in for example a diffractive plane, but these demonstrations are often in free space and can therefore not be implemented on-chip. Integrated on-chip solutions have been demonstrated, where most recently Zhu (2022) et al. demonstrated an on-chip diffractive neural network based on a series of multimode interferometers, MZIs, and diffractive cells [18]. Since these methods are often more sensitive to manufacturing errors, and more complicated to control, these are not further discussed. The method proposed by Ashtani et al. provides a more neuron-like structure and differs strongly from the MZI approach - although still classified as SDM [15]. In this demonstration, all inputs use the same wavelength. The inputs are located in different waveguides and optical attenuators weigh each input separately. The addition is done via a series of connected diodes. Each input has a diode which generates a photocurrent proportional to the intensity of the input light. A downside of implementing this method directly is that each input channel requires a diode which introduces possible scalability issues such as reduced bandwidth or electronic noise. The authors themselves address this issue and provide a possible solution by using something similar to the MZI-mesh approach to sum the inputs. Since the same light source is to be used for all inputs (and outputs) it is possible to leverage coherent addition and multiplication techniques similar to the MZI-mesh. It remains to be seen however if this scales well and if this does not introduce other sources of error.

Other multiplexing techniques exist to further leverage the versatile nature of light. For instance, data throughput can be increased or simultaneous parallel computation can be performed by leveraging different modes or polarization present in waveguides; the complex nature of light can be used to reduce certain computational costs; time division multiplexing techniques are used for multiple reasons, for example to enhance data throughput or to facilitate certain types of computations such as time-delaying signals for convolutional purposes; and many more application exist. Most implementations however use either SDM or WDM as the main multiplexing method. Other multiplexing methods are mainly used to increase data throughput, parallelism, or for specific computations, and can be incorporated into the WDM or SDM architecture if desired [15], [19], [20].



(a) Shows a three-layer PNN for image classification using OEO nonlinear activation [3].



(b) Shows the MZI approach. Nonlinear activation is performed on-chip[16].

Figure 2.2: Selected examples of space division multiplexing (SDM) methods.

NONLINEAR ACTIVATION

In silicon photonics, there are two main approaches for achieving a nonlinear activation for the photonic neurons, namely opto-electro-optical (OEO) or all-optical (AO) [21].

As the name implies OEO first requires a conversion from the optical to the electrical domain. As described before certain methods sum the weighed signal via a photodiode (or some other conversion method from optics to electronics) generating a photocurrent where the current is proportional to the intensity of the light. This current can then in turn be used to modulate light in some nonlinear fashion. Ashtani et al. convert the photocurrent from the diodes into a voltage using a transimpedance amplifier (TIA) after which the voltage modulates an MRM and achieves a ReLu activation [3]. Importantly, the TIA is located off-chip which is due to the difficulty of monolithic integration of photonics and electronics. They note that this will change in the future. Tait (2019) et al. directly use the photocurrent summed with some bias current to affect the effective refractive index of the MRM [22]. In this fashion, ReLu, Sigmoid, and peaking activation functions are demonstrated by changing the bias on the MRM. It should be noted that photonics seems to be somewhat more limited in the type of nonlinear function that can be created. It is for example difficult to create a function such as the tanh() function due to the negative region.

Another approach is to stay in the AO implementation of nonlinearity. A commonly used approach

is the semiconductor optical amplifier (SOA). These implementations are shown to be relatively energy inefficient as can be seen from the results of Shi et al. and Mourgias et al. [6], [7]. Other all-optical solutions have been thought of such as InP on Silicon, resonant cavities, and saturable absorbers [7], [16].

Currently it is an open discussion about which method, OEO or AO, is best suited for photonic neural networks. Proponents of OEO conversion mention the decoupling of input and output benefits cascadability. They argue that wavelength constraints and sensitivities vanish and that the electrical domain offers strong nonlinearity and amplification [22]. It is also mentioned that in general the bandwidth bottleneck is not found in this conversion so the attractive high-inference of photonic neural networks is still achievable. Further, it is often mentioned that a prerequisite for cascadability is a larger than unity gain at the output of a neuron. This is easily achievable for OEO systems, but it is said that the AO conversion often cannot meet this requirement and requires extra amplifiers to achieve this goal.

Proponents of AO mention that the OEO conversion does limit the ultra-high inference promised by photonic neural networks. This is for example the case of Ashtiani et al. where the inference was determined by the operating speed and power consumption of the TIA. Further, demonstrations have been given where multiple AO neurons are cascaded demonstrating the possibility for AO methods.

2.1.2. PHOTONIC NEURAL NETWORKS FOR TIME SERIES

FEED FORWARD PHOTONIC NEURAL NETWORKS

Feedforward PNNs have been demonstrated for a range of different classification tasks [3], [7], [16], [23]. Usually, the PNNs show good results on the presented tasks, but the practicality is worth some discussion. For example, Shen et al. perform vowel recognition by having the power in certain frequency bands of the recording as an input [16]. While vowel recognition at first glance seems to be a time-series task, the heavy pre-processing of the data circumvents this. Further, the nonlinear activation is performed via a CPU. Lastly, the authors mention that only part of the matrix decomposition can be performed on the demonstrated chip. They argue that the full network can easily be implemented on a larger chip, yet scaling of PNNs in general remains a challenge.

Shi et al. (2020) demonstrate an on-chip PNN with 4 neurons where the nonlinear activation is also performed on a CPU [7]. The network is then shown to work on a classification task using multiple layers, but only a single chip was used where the weights were reset, and the output of the first layer was stored on a computer and then used as an input to the same chip. In a later paper, the same group implements the nonlinear activation using an on-chip semiconductor optical amplifier (SOA) and an on-chip laser on an InP substrate which is experimentally demonstrated. These results are then used to simulate a larger network to classify the MNIST dataset, but the whole network was not experimentally validated [8].

One of the interesting achievements of Ashtani et al. is that they directly use light from the image as the input to the neural network [3]. Further, in contrast to most other papers, a larger -albeit still relatively small- network of 3 fully connected layers was experimentally demonstrated proving the performance on the ML task. Especially the end-to-end processing, without data pre-processing, is an interesting aspect of this study.

The feedforward PNN applications discussed were all some form of classification task on a static 'image', while we are interested in data processing of temporal sensor data directly after acquirement, i.e. time series. Even the vowel prediction task, which seems like a time-series task, leaned on the pre-processing of the data to make it possible for the feedforward architecture to perform the classification.

PHOTONIC RECURRENT NEURAL NETWORK

The simple feedforward neuron as previously described is not well suited for temporal sequences. No internal state is present and therefore the NN has no recollection of past inputs. A simple solution is to create a connection between the output and input of the neuron. In this manner, an internal state is created which is updated with the changing input. To this end, it has been demonstrated that there is an isomorphism between a photonic integrated circuit (PIC) and a continuous time RNN (CTRNNs), this implies the theory of CTRNNs can be applied to PICs [11]. Interestingly, it has been shown that CTRNs are universal dynamics approximators [24], [25]. A photonic equivalent of the recurrent neuron, or

CTRNN, can be described by the following differential equation, with a schematic shown in figure 2.3:

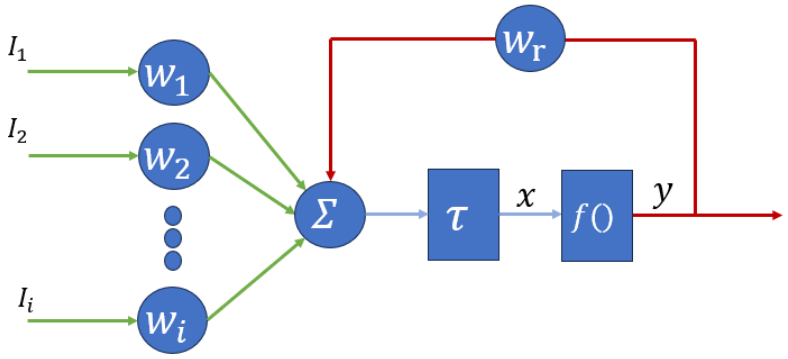


Figure 2.3: Schematic of the setup for a continuous time recurrent neuron.

$$\tau \frac{dx(t)}{dt} = -x(t) + \sum_i w_i I_i + w_r y(t) + b$$
$$y(t) = f(x(t)) \tag{2.1}$$

Were $x(t)$ is the internal state, $\frac{dx(t)}{dt}$ is the analog evolution of the internal state in time, τ represents the time constant of the neuron, $y(t)$ represents the output, $I_i(t)$ the input, b the bias, and W_i and W_r the weight of the inputs and the recurrent loop, respectively.

Different versions of the photonic recurrent neuron have been designed. Tait, de Lima, and Peng have demonstrated several versions of the recurrent neuron based on the broadcast and weight architecture [22], [26], [27]. Two of these neurons have also been implemented on-chip showing cascadability, albeit with no recurrent connection [13].

Multiple attempts have been made to show how the recurrent neurons could work in a larger network by finding the neuron transfer function from experiment. The transfer function is then implemented in a simulation of a larger NN. While these results are valuable in terms of demonstrating the potential capabilities of these designs, Often the reports do not mention physical considerations such as noise term or coupling of the time constant to the machine learning task. Similar demonstrations have been given by Mourgios et al for photonic RNNs where single recurrent neurons with multiple inputs have been demonstrated, but tasks where larger networks are required are only given through simulation [4], [6]. Shi et al. also mention the possibility of investigating the recurrent link but have not yet demonstrated this through either simulation or experiment [7].

THE TIME-CONSTANT IN PHOTONIC NEURAL NETWORKS

PNNs promise ultra-fast inference and low energy consumption, making them promising for various time-series tasks. The suitability of PRNNs for such tasks has been explored in several studies. For example, de Lima et al. (2019) suggested possible applications of PRNNs in model predictive control, scientific computing, and nuclear fusion [27]. Several studies have demonstrated the effectiveness of PRNNs in time-series tasks such as stock market prediction [5], RF fingerprinting [28], chaotic dynamics prediction, and engine anomaly detection [26].

It is stated by several of the authors that the PRNNs are limited to short-term correlations. However, what does a short-term memory mean? Here the difference between discrete time (DT) and continuous time (CT) RNNs is of importance. A DTRNN processes data in fixed steps. This also means that the recurrent input also operates with these steps. For a standard DTRNN, it is sometimes stated that the

memory capacity is somewhere around 10 to 20 samples in the past [29]. Hence, what the samples are determines what short- or long-term is. For example, if the RNN is used as a language model the steps of previous samples could be the words or letters [30]; if the RNN processes a sampled time series, the samples would refer to how many time steps it can remember.

As mentioned, a photonic neuron is modeled as a differential equation with a time constant, namely a CTRNN [24]. The time constant determines not only how fast the neuron reacts to external inputs, but also how long the activation state remains. A larger time constant takes a longer time to accept new inputs, but also retains historical information longer [31], [32]. In other words, a shorter time constant leads to a shorter memory span, but a faster reactivity. Further, the time constant has a strong influence on the performance of the CTRN, and CTRNNs have been shown to match the performance of LSTMs and require fewer neurons for higher-level learning [31]. When the CTRNN is emulated on digital hardware the computational cost is high and CTRNNs are therefore rarely implemented [33]. When the computation is implemented physically, such as in PNNs, this computational burden is not present. The time constant can be included as a trainable parameter similar to the weights and biases [34], and importantly, the time constant can be different for each neuron. In this fashion, multiple timescale (MT) CTRNNs can be created where one can learn on different timescales[32], [35]–[38].

PRNNs have thus far only been shown to work on tasks requiring short-term correlation, as high-speed processing is given as one of its main strengths. This can be attributed to the extremely fast operation speeds of these PRNNs, in other words, the extremely short timescales of the photonic neurons. For example, Ashtiani et al present a 570 ps end-to-end inference in their 3-layer PNN (not recurrent). On the other hand, however, we have the dynamics of the input signal. While often the simulated PRNNs are demonstrated on tasks obtained from sensory data [39], [40], or other benchmarks, the data is passed to the network in sped-up fashion, i.e. the PNN could not be used on these tasks in real-time. It turns out to be difficult to find processes where such short-term correlations are of actual interest.

This poses an interesting challenge: is it possible and/or useful to use PRNNs on tasks with much slower time constants? One interesting question is if one can use physical hardware with a much shorter time constant than the slow signal processing task which requires long memorization. This last sentence is also a paraphrasing of the starting motivation of a large European project called MemScales [41], [42].

One approach is constructing more complex PRNNs, such as photonic LSTM. These have been proposed, but they have not been validated experimentally [43]. Gated recurrent neurons have also been proposed and experimentally validated, but only on digital signals and with unrealistic on-chip delays, making their performance unclear [6]. Investigating how the operators in GRUs and LSTMs could be implemented and incorporated in on-chip photonics is an interesting research direction, but not further investigated in this report.

Another approach could be to slow down the time constant of the photonic neuron such that we match the dynamics of interest. In principle, this seems possible. Before I mentioned that the time constant of the photonic neuron by Ashtiani et al. was on the order of hundreds of picoseconds. However, in the paper, they explicitly mention that the bandwidth of the TIA can be designed based on the application. They mention that at low-speed operation the end-to-end classification time is under 1 μs. It shows some controllability of the time constant. Similarly, in Tait et al. (2017) the time constant was controlled by an RC low pass filter. Increasing the time constant would however also negate one of the benefits of photonics, namely the ultra-fast inference. It should be noted that the time delay cannot be placed in the recurrent loop, but must be added as the internal time constant of the neuron. A time delay in the loop leads to so-called time-delayed dynamics, which are more difficult to analyze [39]. In passing I mention that these types of dynamics have been used for time-delayed photonic reservoir computing. Time Delayed photonic reservoir computing has been demonstrated experimentally on many occasions [40], [44], [45], but is outside the scope of this report. Regardless, it remains to be investigated if the time constant of an on-chip photonic NN can be practically slowed down enough to match the time scale of the sensor and the dynamics of the movement. Possible challenges are monolithically integrating the electrical circuit (such as the TIA or the RC low pass filter) on the chip and controlling the stability of the time constant within the desired range, among others.

2.2. TIME CONSTANTS IN CONTINUOUS TIME RECURRENT NEURAL NETWORKS

As has been mentioned in the previous chapter there exists a mathematical isomorphism between on-chip photonic integrated circuits and the continuous time recurrent neuron (CTRNN). Such a CTRNN introduces the notion of a time constant to the standard recurrent neural network equations, which alters its behavior. This section will look at the literature on CTRNNs, and in particular the influence of the time constant. There is much to say about this topic, therefore we will restrict ourselves to what is deemed important for a physical photonic realization.

2.2.1. INFLUENCE OF TIME CONSTANTS ON MEMORY AND DYNAMICS

Recurrent neural networks (RNNs) obtain a form of memory from the recurrent connections which allows them to retain their state. What is meant by memory in this case does not have a good definition, but broadly speaking we can look at transient- and attractor-based memories [46]. Very briefly: transient memory stems from the transient dynamics that decay with time; attractor memory stems from the formation of fixed (and potentially slow) points in the network. Further, we can distinguish between node- and network memory [47]. Where node memory is about the dynamics of the node (i.e. the time constant), and the network memory stems from the network's connectivity and increases with the size of the network. As said before, these concepts are not always well defined, mean different things depending on the author [41], and can quite possibly affect one another. The report will focus on the time constant, and the duality/link to attractor-based memory. Interesting research on attractor-based memory, including fixed- and slow points, has been done by Barack and Sussillo [48], [49].

The time constant has been found to strongly impact the performance of recurrent neural networks [33], [35], [47], [50], [51]. At this point, it is important to make a distinction between echo-state networks (ESN), a form of reservoir computing (RC), and RNNs. ESNs are essentially (CT)RNNs where the network, or reservoir, is not trained but (pseudo) randomly initiated following some conditions called the reservoir conditions [52]. The ESN is then trained via the linear readout layer. This concept was introduced to alleviate the difficulties and computational burden of training RNNs.

Before starting my research in the report, I was unaware of the research on the time constant for ESN and regarding the effect of the time constant in CT-ESNs literature is more extensive as compared to the CTRNN literature. The effect of the time constant, and the network size, has been examined by Verstraeten [47]. As is commonly done for time series tasks, the input has been pre-processed with some frequency domain techniques. It is uncommon and sometimes discouraged [53], to directly input a time series to an RNN without pre-processing. However, for a direct end-to-end on-chip application as is the topic of this report, this is the most straightforward approach without adding additional components. Further, a direct link to the temporal characteristic of the input signal is not made. To the best of my knowledge, this is only examined in more detail through the means of the auto-correlation function by Manneschi (2021) et al where they examined the influence of multiple time constants in hierarchical ESNs [54]. They found a relation between the time constant of the neurons in different reservoirs and the auto-correlation of the signal. It should be noted that relatively large networks have been used and that reservoir computers do not learn the reservoir weights and thus not form task-specific attractor-based dynamics. Another interesting and more elaborate (and arguably more elegant) approach to incorporating time into the neurons is done with filter-neurons, an early example of these in the form of bandpass neurons is given by Wyffles et al (2008) [55]. As with many of these concepts, these were applied to ESNs and not CTRNNs. An interesting future endeavor would be applying these filter neurons in networks with trainable weights.

2.2.2. DISCREPANCIES IN METHODOLOGICAL APPROACHES

Another distinction typically seen between the ESN literature and the CTRNN literature is the specific formulation of the neuron model. A continuous time neuron can be viewed as a standard recurrent neuron, with the addition of a low-pass filter. Typically, in the ESN literature, the low pass filter is located after the nonlinear activation function of the neuron. However, a similar and comparable equivalent

can be obtained by locating the low-pass filter before the nonlinear activation function. This version is more commonly seen in physical applications of the CTRNN [11], [33], however, this is rarely elaborated on. A third option is locating the filter over the activation function [47]. From a computer science perspective, the location of the low pass filter is only important due to the different dynamics. From a physical viewpoint, however, the location matters not only due to the dynamics but also in what is physically realizable. Most of the information on the time constant in relation to the task is found in models where the filter is located after the nonlinear activation, as this was mostly done from a computer science perspective. While there are examples of physical CTRNNs where the filter is located before activation, a more thorough investigation of how the time constant affects the results has not been performed. Appendix B shows a simple demonstration that the location of the low-pass filter matters for the dynamics.

2.2.3. TIME CONSTANT IN PHOTONIC IMPLEMENTATIONS

As mentioned in chapter 2.1.2 there are physical implementations of CTRNNs. Some of these have been physically implemented [22], [26], and some have been emulated on different hardware such as FPGAs[28]. Mostly, however, the relation of the time constant is not mentioned. For example, in [26] the PRNN is applied to the fordA time series as a pre-processing stage before a CNN, but the specifics of the implementation are lacking. Further, no mention is made of how one would match the time constants of the tasks. This is fair for a demonstrative purpose when artificial data is used as an input, but in any real-time operating conditions, such as for direct on-chip end-to-end photonic processing using PRNNs, one needs to take this into account. Similar on-chip demonstrations have been given using the RC paradigm, but in these cases, the data is almost always augmented with a pre-processing stage, and used with 'sped-up' data. A nice example is Vandoorne et al (2010) who use a photonic RC using semiconductor optical amplifiers on speech data but also mention that for real-time applications the audio data is much too slow for the typical time scales in the photonic RC [56]. Apart from applications in digital computing to speed up some parts of the computation (for example the matrix-vector multiplication), no mention is made for which applications PRNNs (or photonic RCs) can be used. Typically a mention is made of a mismatch, or that these networks are useful for short-term correlations, but are left unexplored.

Another consequence of a physical implementation comes in the form of noise. This can be thermal noise, electrical noise, lossless, manufacturing inequalities, etc. This has been discussed for PNNS [57], [58], but I have not seen these noise components mentioned in relation to CTRNNs, nor how this might affect the required time constants or performance. Further, there are more considerations one needs to include in a physically plausible model regarding for example the input, weights, and activation function [27], which could have implications for the training of the model, the performance, and the time constant.

2.3. CONCLUSION

Photonic neural networks come in many varieties, all with their strengths and weaknesses. The literature is moving quickly, possibly due to an increased interest due to better manufacturing techniques, a rise in the popularity of AI, and a requirement for more computational power with lower energy consumption. For the interest of the report, end-to-end time-series processing using photonic recurrent neural networks, many questions lie in the open. Some of these questions remain in general for photonic neural networks, such as how to deal with inaccuracies, errors stemming from the analog nature of the computation, and how and when to leverage the strong points of optical computations, mainly the very fast potential speeds. Others are more specific to the end-to-end processing of time series using CTRNNs, such as the relation between the task and time constant of the neurons, the necessity of pre-processing, and the broader question of memory in such networks. Many of these questions came together in the original proposition of this project: 'using a photonic neural network to process inertial measurement unit data'. Before tackling that question, however, I believe it is necessary to first answer a few more fundamental questions and challenges.

I believe the most important, question how the time constant of the photonic neurons relates to

the task. Can we understand the dynamics of CTRNNs, and couple this information to make informed decisions on how to design, and for which applications to use, photonic CTRNNs.

3

Dynamics of Continuous-Time Recurrent Neural Networks

A continuous-time recurrent neural network is a network of neurons where all the neurons are described by differential equations, and where the inputs to the neuron are low-pass filtered. In general, all neurons are recurrently connected to every other neuron in the layer, including a self-connect, as can be seen in figure 3.1.

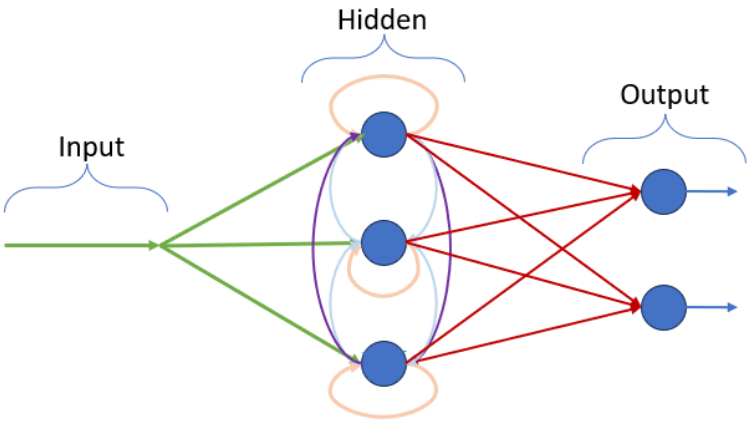


Figure 3.1: Example schematic of a fully connected continuous-time recurrent neural network. Schematic shows a single input for all neurons, three neurons in the recurrent hidden layer, and 2 neurons in the output layer. In general, these can be of arbitrary size.

Mathematically the system may be described as:

$$\begin{aligned} \tau \frac{d\mathbf{x}(t)}{dt} &= -\mathbf{x}(t) + \mathbf{w_I} \cdot \mathbf{I}(t) + W_r \mathbf{y}(t) + \mathbf{b} \\ \mathbf{y}(t) &= f[\mathbf{x}(t)] \end{aligned} \tag{3.1}$$

Where τ are the time constants of the neurons, $\mathbf{x}(t)$ denotes the state of the neuron, $\mathbf{w_I}$ the weight vector connecting the input(s) to the neurons, $\mathbf{I}(t)$ the time-varying inputs to the neurons, $\mathbf{b}$ a bias vector, $\mathbf{y}(t)$ the output of the neuron which is a nonlinear transformation $f[\cdot]$ of the neurons state, and W_r a matrix

connecting the output of the neurons recurrently.

For the following chapter I will assume a zero bias ($\mathbf{b} = \mathbf{0}$), a single input ($\mathbf{I}(t) = I(t)$, and an input connected to each neuron without weighting ($W_I = \mathbf{w_I} = \mathbf{1}$). I will also assume the nonlinear function to be a hyperbolic tangent. These assumptions are made to avoid adding too much complexity, while still demonstrating the main points. To understand the system better, we will first examine the most simple case.

3.1. CONTINUOUS-TIME FEEDFORWARD NEURON

The simplest starting point when viewing neural networks through the lens of dynamical systems is the single continuous-time feedforward neuron. This can be obtained from the general CTRNN model with a scalar version of equation (3.1) and by setting the recurrent weight to zero ($w_r = 0$), as seen in figure 3.2.

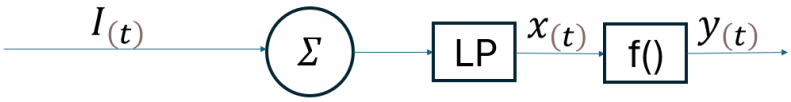


Figure 3.2: Example schematic of a single continuous-time feedforward neuron. I(t) denotes the time-varying input to the neuron, x(t) the low-pass filtered input, i.e. the state of the neuron, and y(t) the output of the neuron after nonlinear activation.

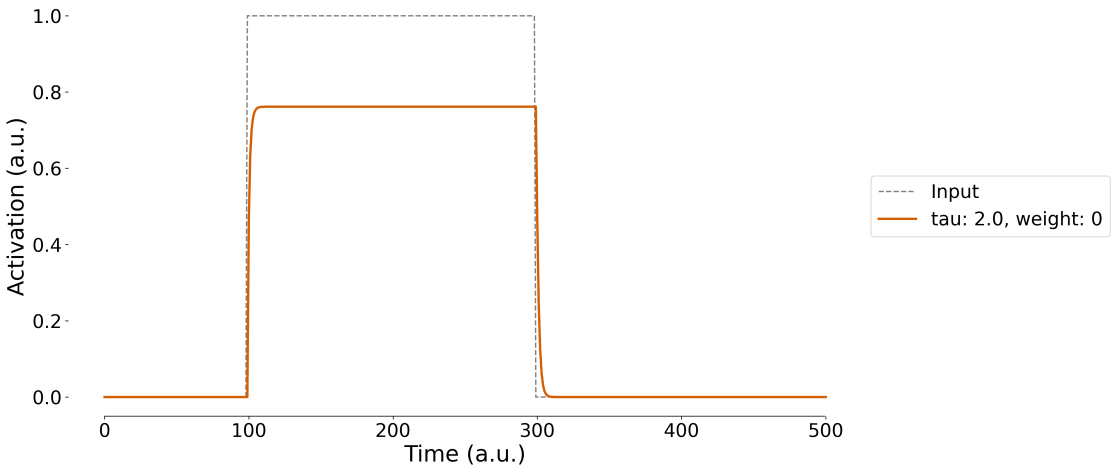
This results in the following equation:

$$\begin{aligned} \tau \frac{dx(t)}{dt} &= -x(t) + I(t), \\ y(t) &= f[x(t)], \end{aligned} \tag{3.2}$$

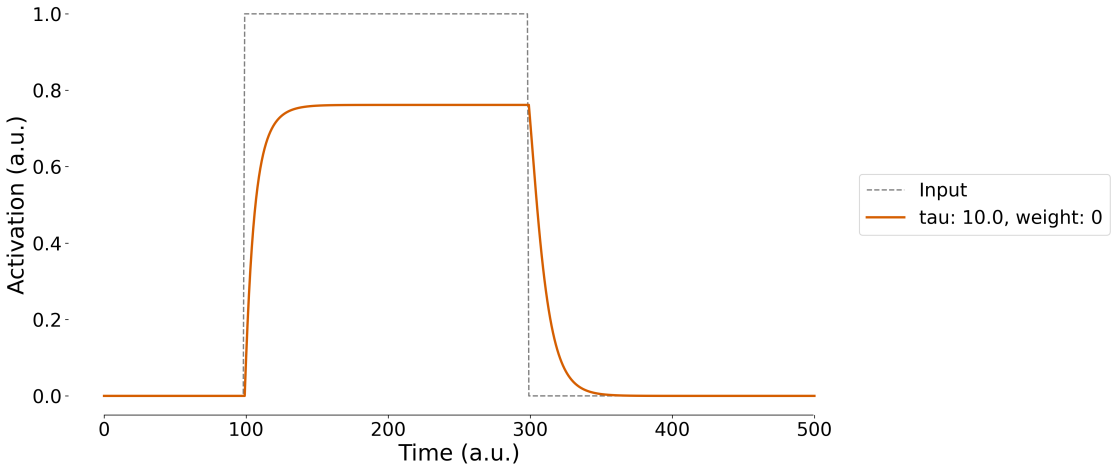
Where $I(t)$ is the input to the neuron, $x(t)$ is the state of the neuron, $f[\cdot]$ is a nonlinear activation function, $y(t)$ is the output of the neuron, and τ is the time constant of the low-pass filter in the neuron. We will look at these neurons (and networks) through the lens of dynamical systems. If for the time being we assume a stationary input ($I(t) = I$) we can observe that this system will always evolve towards its stable fixed point ($\frac{d\mathbf{x}(t)}{dt} = 0$):

$$\begin{aligned} x^*(t) &= w_I \cdot I \\ y^*(t) &= f[x^*(t)] \end{aligned} \tag{3.3}$$

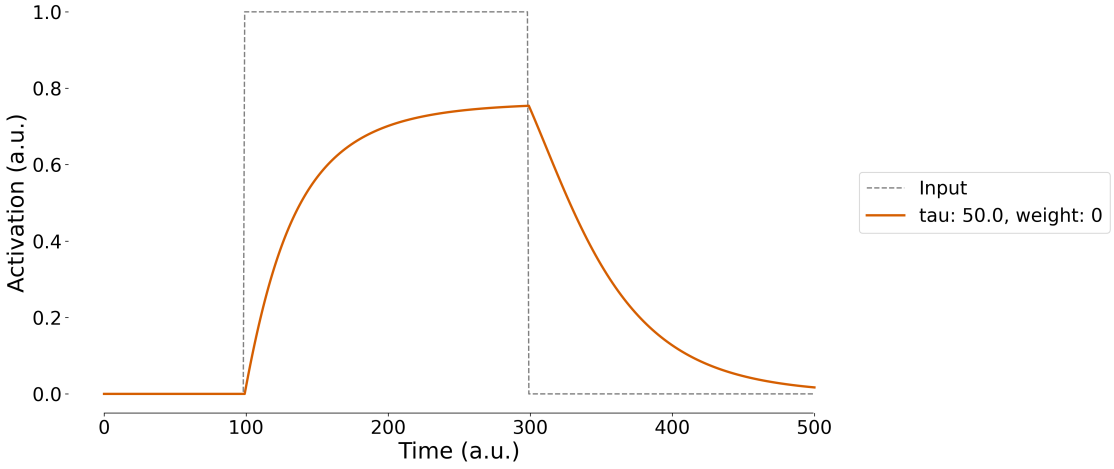
Where the star ($\cdot^*$) denotes the state at the fixed point. It may be observed that the fixed points are independent of the time constant of the neurons. The time constant scales the rate at which the state changes. Interestingly, even without any recurrent connection the feedforward neuron, which typically is agnostic to time, now has a notion of time via the low-pass filter. While not within the scope of the report, this can be interesting for future study.



(a) Continuous-time feedforward neuron with time constant $\tau = 2$ excited with an input unit pulse. Input pulse in grey; orange denotes the output of the neuron.



(b) Continuous-time feedforward neuron with time constant $\tau = 10$ excited with an input unit pulse. Input pulse in grey; orange denotes the output of the neuron.

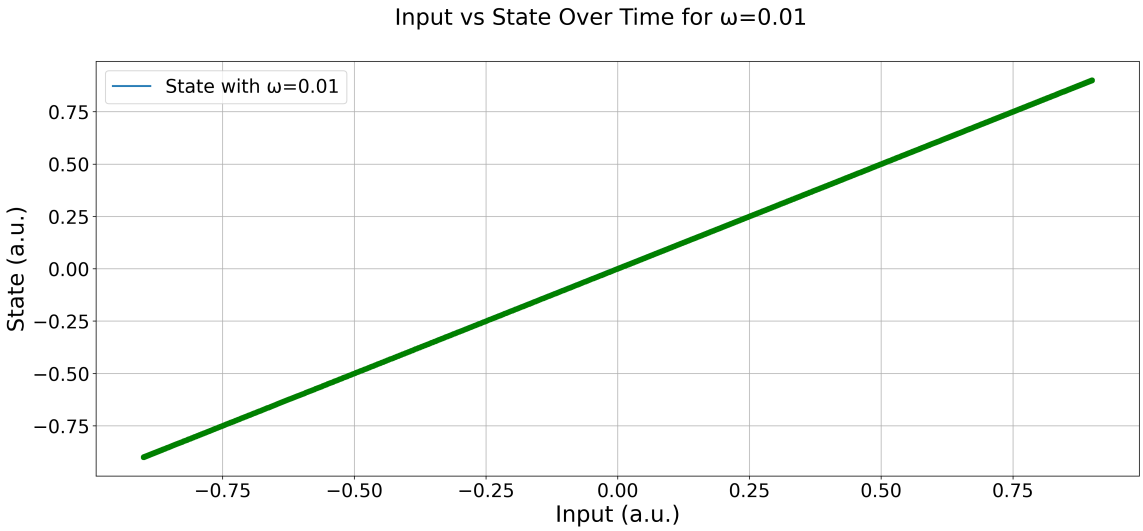


(c) Continuous-time feedforward neuron with time constant $\tau = 50$ excited with an input unit pulse. Input pulse in grey; orange denotes the output of the neuron.

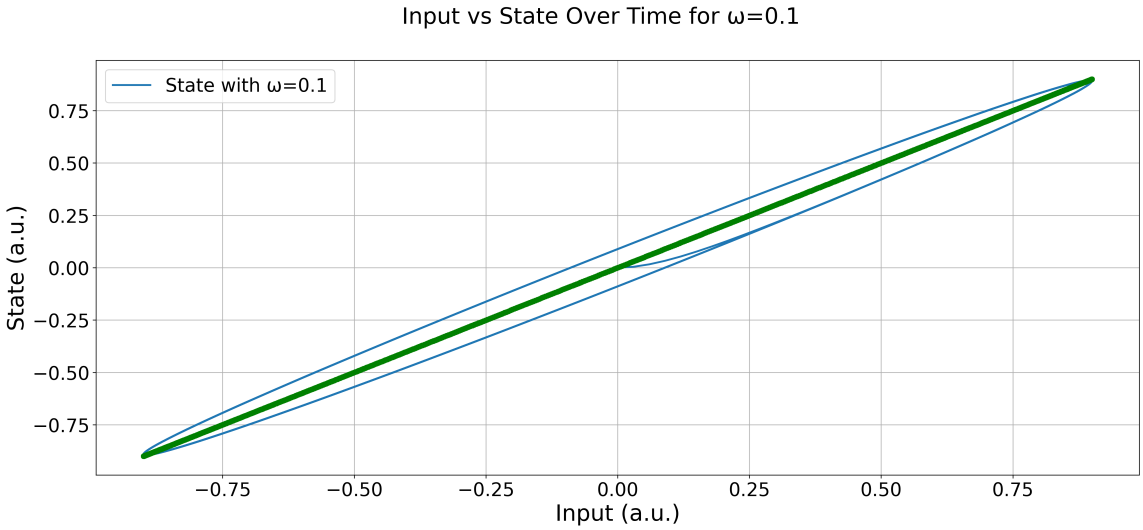
Figure 3.3: Output of a continuous-time feedforward neuron with different time constants. A larger time constant means a slower response and a slower decay of the neuron's output.

Figure 3.3 shows the behavior of the feedforward neuron for different time constants when a step input is applied. We notice that indeed the output converges to a steady state and that the time constant determines how fast the neuron reacts and decays.

Note that thus far we have assumed a stationary input in our description of the fixed points. However, the inputs we will consider are typically not stationary but time-varying. We can numerically investigate how the neuron behaves for a time-varying input. We choose a sinusoidal input. We first plot all the stationary fixed points (green in figure 3.4 and 3.5) for the entire feasible input range. We then compare these points to the actual state under the influence of the time-varying sinusoidal input with different frequencies (blue in figure 3.4 and 3.5).

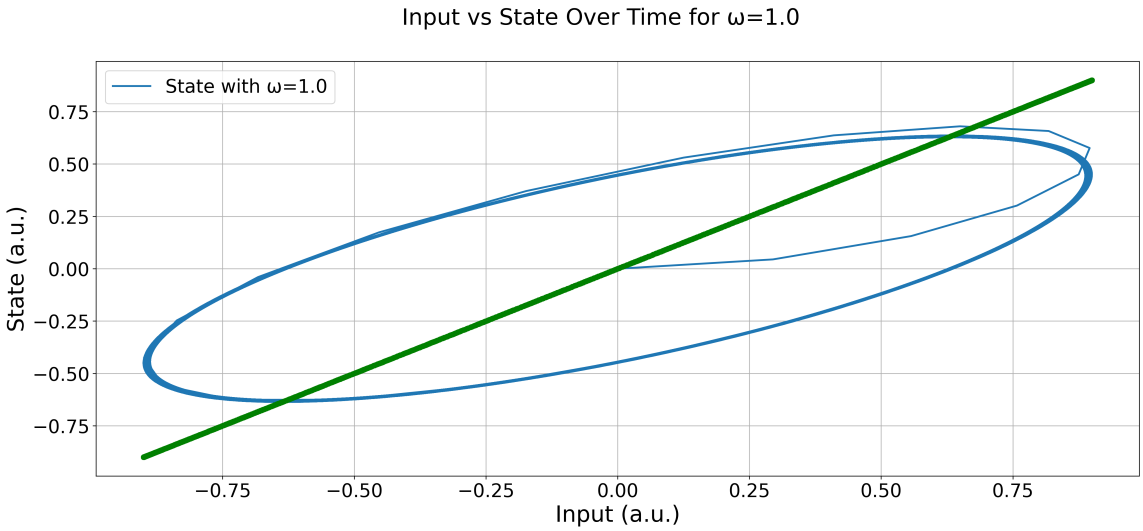


(a) State exactly follows the stationary fixed points for very slow input: $\omega = 0.01 << \frac{1}{\tau} = 1$.

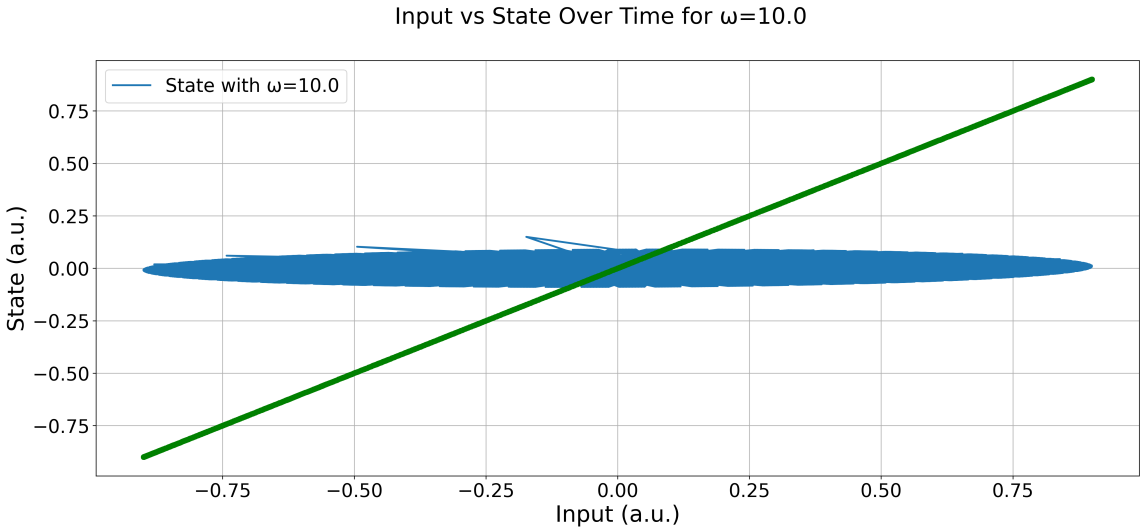


(b) State almost follows stationary fixed points, but diverges slightly, for slow input: $\omega = 0.1 < \frac{1}{\tau} = 1$.

Figure 3.4: Output of a continuous-time feedforward neuron with fixed time constant $\tau = 1$ for slow input frequencies. Green dots (line) show the stationary fixed points for the input range. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.



(a) State no longer follows the stationary input when the input is fast compared to the change of state of the neuron: $\omega = 1 = \frac{1}{\tau} = 1$.



(b) State shows almost no movement under the influence of the very fast input due to the low-pass filtering nature of the neurons: $\omega = 10 >> \frac{1}{\tau} = 1$.

Figure 3.5: Output of a continuous-time feedforward neuron with fixed time constant $\tau = 1$ for (relatively) fast input frequencies. Green dots (line) show the stationary fixed points for the input range. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.

Figure 3.4 and 3.5 show several cases for the feedforward neuron with different frequency inputs. The time constant is fixed to $\tau = 1$. The figure shows 4 distinct cases:

- (1) $\omega << \frac{1}{\tau}$ (Figure 3.4a)
- (2) $\omega < \frac{1}{\tau}$ (Figure 3.4b)
- (3) $\omega = \frac{1}{\tau}$ (Figure 3.5a)
- (4) $\omega >> \frac{1}{\tau}$ (Figure 3.5b)

It can be seen that when the input is very slow (Figure 3.4a) the state almost exactly follows the stationary fixed points. As we increase the frequency the state starts lagging behind the input, but still follows the general trend of the fixed points (Figure 3.4b). Increasing it further we note that the state shows much different behavior as compared to the stationary fixed points, and follows its own trajectory (Figure 3.5a). Interestingly, as we increase the input frequency even more the influence of the input diminishes as the frequency starts to get filtered out (Figure 3.5b). At this point, the state simply converges towards the stable fixed point corresponding to the mean of the input signal ($I = 0$).

3

3.2. CONTINUOUS-TIME RECURRENT NEURONS

The continuous-time nature of the neuron already adds to the typical behavior of a feedforward neuron by adding some notion of time through the low-pass filtering behavior. Adding a recurrent connection allows for even more complex dynamics and in that sense computational variety.

3.2.1. A SINGLE NEURON

As we did with the feedforward neuron the starting point will be a single neuron. Figure 3.6 shows a schematic of such a neuron.

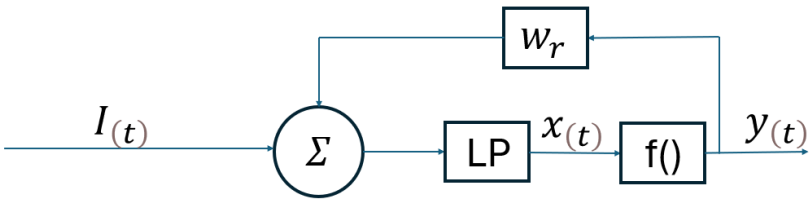


Figure 3.6: Example schematic of a single continuous-time feedforward neuron. I(t) denotes the time-varying input to the neuron, x(t) low pass filtered input, i.e. the state of the neuron, and y(t) the output of the neuron after nonlinear activation.

Mathematically the single neuron is described by:

$$\begin{aligned} \tau \frac{dx(t)}{dt} &= -x(t) + I(t) + w_r y(t), \\ y(t) &= f[x(t)], \end{aligned} \tag{3.4}$$

Similar to what we did with the continuous-time feedforward neuron we can investigate the fixed points by setting $\frac{dx(t)}{dt} = 0$. If we do so we obtain:

$$\begin{aligned} x^*(t) &= I(t) + w_r y^*(t) \\ y^*(t) &= f[x^*(t)] \end{aligned} \tag{3.5}$$

Again it may be observed that the fixed points do not depend on the time constant of the neuron. It can also be observed that in contrast to the feedforward case, the fixed point is now a function of a nonlinear transformation of the state. As such, instead of having a single fixed point, we can now have more than one. Generally, this depends on the specific nonlinearity used. In certain cases, we can find these points analytically by approximating with a Taylor series and finding the roots of the equation. The order of the approximation then shows the number of possible fixed points. The actual fixed point(s) found depend on the specific bifurcation (the input, and/or recurrent weight) parameters [24]. We can also find the roots numerically, which is what we will do for the following.

As mentioned, we can observe in equation 3.5 that the recurrent weight w_r and the input $I(t)$ affect the fixed point(s). At first, it is most convenient to look at the system when the input is stationary, i.e. $I(t) = I$. If we set the input to zero $I = 0$ we can plot the fixed points as a function of the recurrent weights and show how the fixed points are destroyed/created as the recurrent weight varies. This so

called bifurcation diagram is shown figure 3.7.

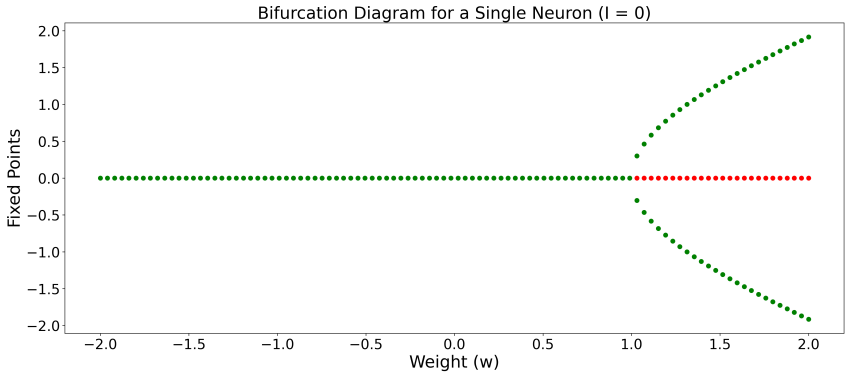
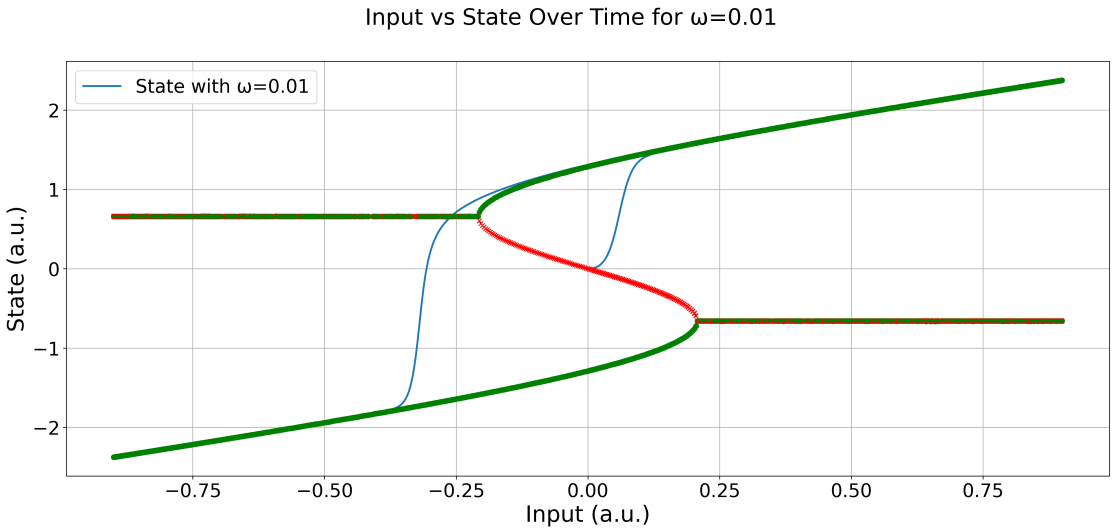


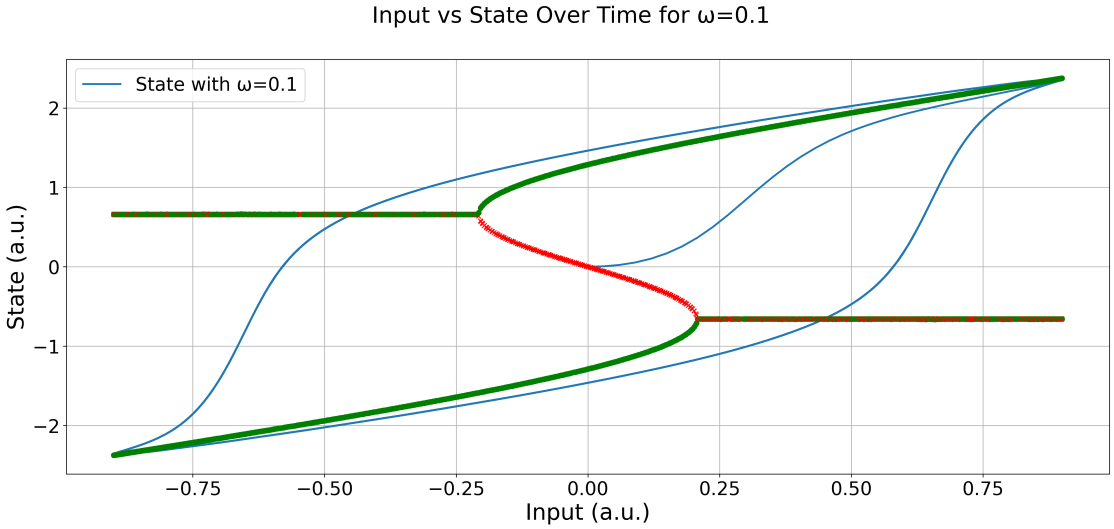
Figure 3.7: Bifurcation diagram for a single continuous-time neuron with no input. At $w_r^* = 1$ the single stable fixed point (green) bifurcates into an unstable fixed point (red) and two stable fixed points emerge.

We notice that before some critical weight value ($w_r^* = 1$) the network has a single stable fixed point (green balls). If we increase the weight beyond this point the stable fixed point becomes unstable (red balls) and two new fixed points emerge and move further apart as we increase the weight further.

As before, we are more interested in the cases of non-stationary input and how the neuron behaves for different input conditions. As we did for the single feedforward neuron we can create a plot where we plot all the fixed points for the entire range of stationary input conditions ($I = [-1, 1]$). We then pass a non-stationary (sinusoidal) input to the network and see how the state under the non-stationary input matches or differs from the stationary fixed points. We do so for different input frequencies and keep the time constant the same throughout. This can be seen in figures 3.8 and 3.9:

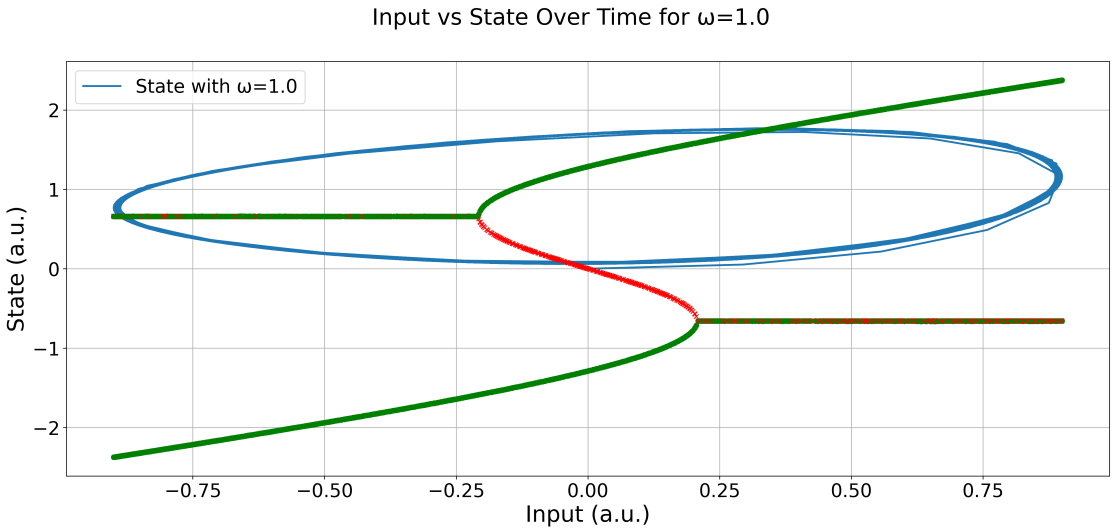


(a) State almost exactly follows the stationary fixed points for very slow input: $\omega = 0.01 << \frac{1}{\tau} = 1$.

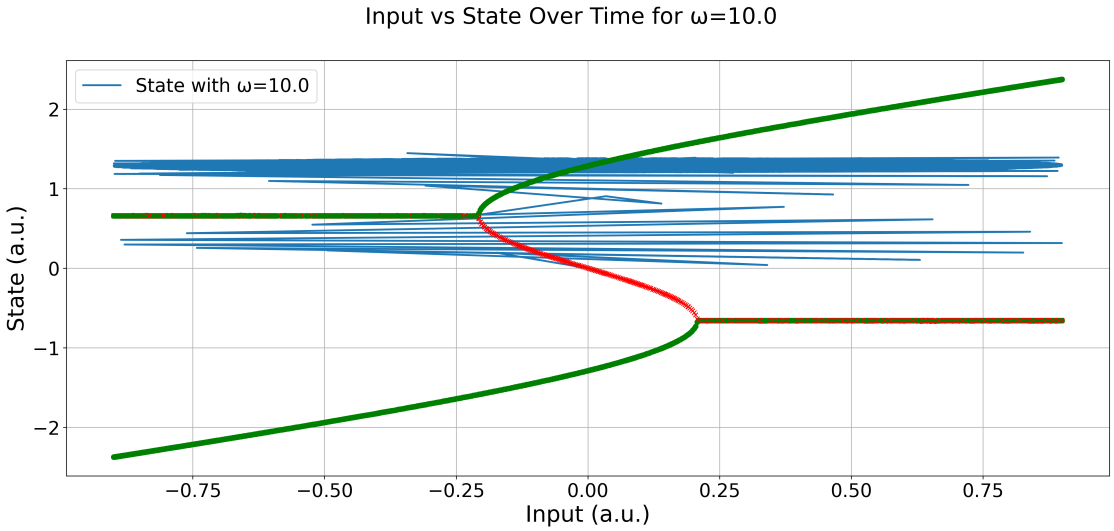


(b) State almost follows stationary fixed points, but diverges slightly, for slow input: $\omega = 0.1 < \frac{1}{\tau} = 1$.

Figure 3.8: Output of a continuous-time recurrent neuron with fixed time constant $\tau = 1$ for slow input frequencies. Green dots/lines show the stable stationary fixed points for the input range, and red dots/lines the unstable fixed points. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.



(a) State no longer follows the stationary input when the input is fast compared to the change of state of the neuron: $\omega = 1 = \frac{1}{\tau} = 1$.



(b) Under the influence of the very fast input: $\omega = 10 >> \frac{1}{\tau} = 1$. The state moves towards the fixed point corresponding to the mean of the input (in this case I=0) and only oscillates lightly around that point, showing the dominant low-pass filtering behavior.

Figure 3.9: Output of a continuous-time recurrent neuron with fixed time constant $\tau = 1$ for fast input frequencies. Green dots/lines show the stable stationary fixed points for the input range, and red dots/lines the unstable fixed points. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.

Again we can discuss the four different cases:

- $\omega << \frac{1}{\tau}$ (Figure 3.8a)
- $\omega < \frac{1}{\tau}$ (Figure 3.8b)
- $\omega = \frac{1}{\tau}$ (Figure 3.9a)
- $\omega >> \frac{1}{\tau}$ (Figure 3.9b)

We observe similar behavior as in the feedforward case. When the frequency of the input is sufficiently slow as compared to the intrinsic dynamics of the network, the state follows the stationary fixed points

(Figure 3.8a). Increasing the frequency but staying slow compared to the network dynamics we see the behavior qualitatively following the fixed points, but the state starts to lag the input and the state only reaches the fixed points when the input changes more slowly (Figure 3.8b.) Further increasing the frequency of the input where $\omega = \frac{1}{\tau}$ we notice that the behavior differs strongly from the stationary fixed points and the state no longer follows a trajectory that much resembles the fixed points. The behavior does still seem to be somewhat influenced by the fixed points but less so. In this specific case, the input is unable to drive the neuron to the lower state. Interestingly this seems to be right on the edge when the bifurcation is about to happen, and this also corresponds to the cutoff frequency of the low-pass filter (Figure 3.9a). When we increase the frequency even further, we see that the low-pass filtering behavior of the neuron is taking over. The input amplitude is strongly dampened and we see a low amplitude oscillation around the fixed point associated with the mean of the input (Figure 3.9b).

3.2.2. A SMALL NEURON NETWORK

The smallest possible network is that containing two neurons. Figure 3.10 shows a schematic of a 2-neuron network:

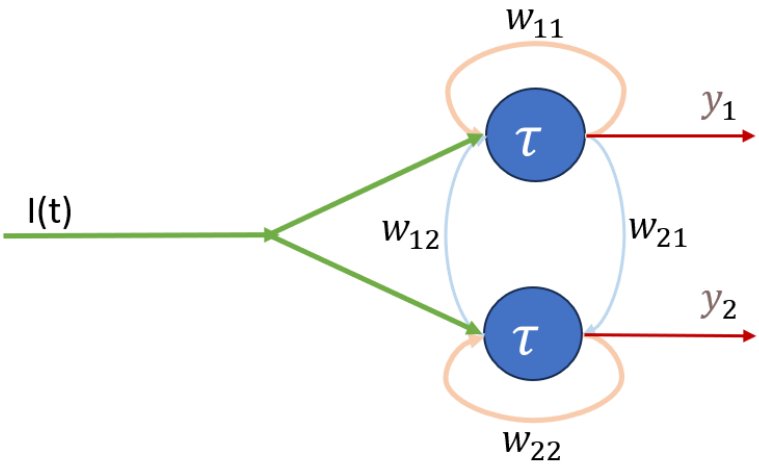


Figure 3.10: Simple schematic of a 2 neuron continuous-time neural network.

The general form of the equation is the same as equation 3.1 where now each of the vectors is 2D and the recurrent weight matrix a 2x2 matrix., in general:

$$W_r = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$$

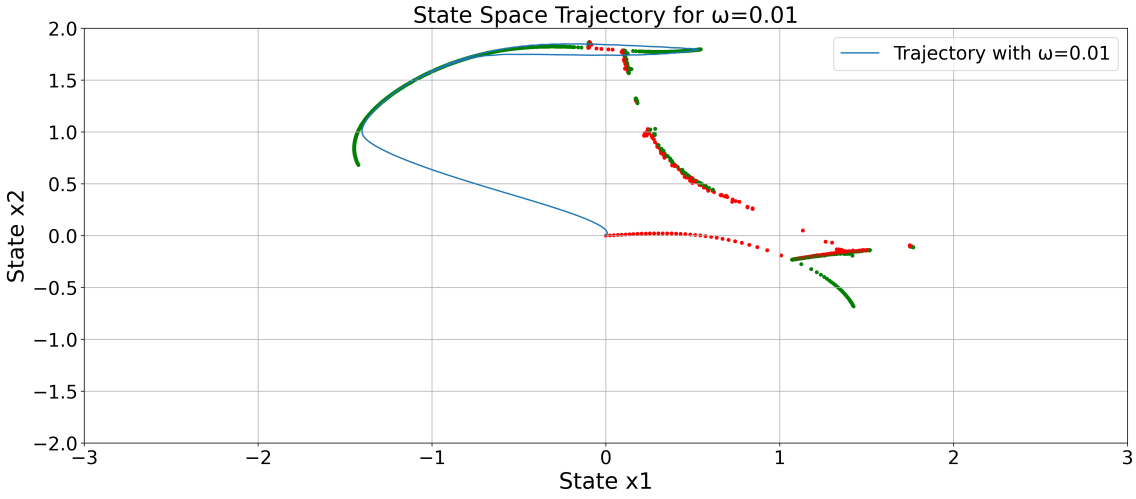
In a similar fashion to the previous analysis, we can again find the fixed points. For a network of two neurons there are 9 qualitatively different fixed points that can be found for a sigmoidal activation function, which is in essence a shifted and scaled hyperbolic tangent [24]. A full demonstration of all the types of behavior and typical examples is out of the scope of this report.

I will examine a single example in more detail in order to demonstrate that the qualitative behavior for time-varying inputs seems to remain similar as we increase the network size. I use the recurrent matrix:

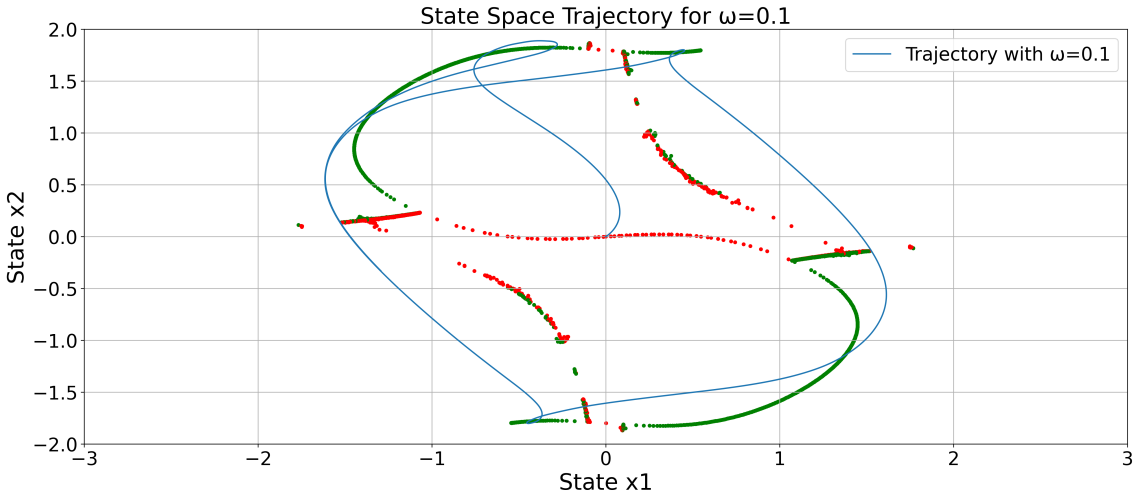
$$W_r = \begin{pmatrix} w_{11} = 1 & w_{12} = -0.9 \\ w_{21} = -0.1 & w_{22} = 1 \end{pmatrix}$$

As we did before I plot the stationary fixed point for the entire input range which is denoted by green (stable) and red (unstable) balls. Since we now have two neurons I opt to only plot the states of the two

neurons, and leave out the input in the figure (in contrast to the single neuron case). Figures 3.11 and 3.12 show the results.

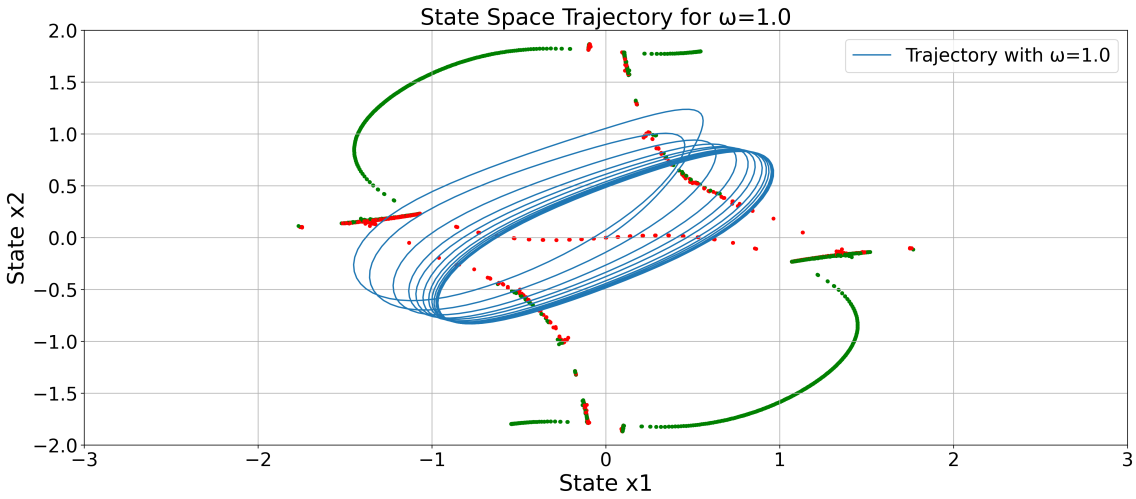


(a) State almost exactly follows the stationary fixed points for very slow input: $\omega = 0.01 << \frac{1}{\tau} = 1$. Note that the input is not sufficient to drive the input beyond the bifurcation point.

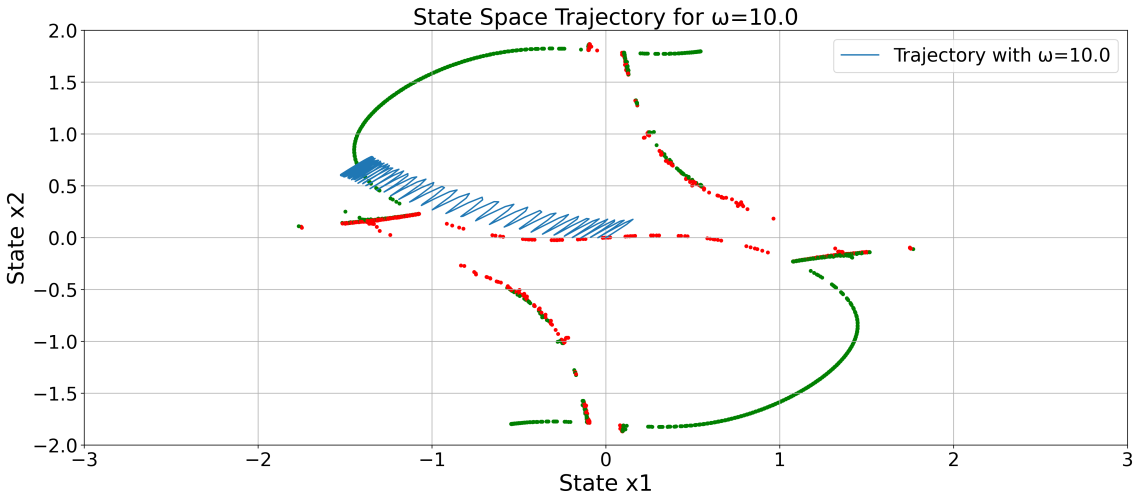


(b) State almost follows stationary fixed points, but diverges slightly, for slow input: $\omega = 0.1 < \frac{1}{\tau} = 1$.

Figure 3.11: Output of a 2D continuous-time recurrent neuron with fixed time constant $\tau = 1$ for slow input frequencies. Green dots/lines show the stable stationary fixed points for the input range, and red dots/lines the unstable fixed points. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.



(a) State no longer follows the stationary input when the input is fast compared to the change of state of the neuron: $\omega = 1 = \frac{1}{\tau} = 1$.



(b) Under the influence of the very fast input: $\omega = 10 >> \frac{1}{\tau} = 1$. The state moves towards the fixed point corresponding to the mean of the input (in this case 0) and only oscillates lightly around that point, showing the dominant low-pass filtering behavior.

Figure 3.12: Output of a 2D continuous-time recurrent neuron with fixed time constant $\tau = 1$ for fast input frequencies. Green dots/lines show the stable stationary fixed points for the input range, and red dots/lines the unstable fixed points. Blue lines show the actual state trajectory of the neurons under the influence of the time-varying inputs.

We again see a similar relation between the input and the state. Most notably, when sufficiently slow the state follows the stationary fixed point trajectory, when too large the input gets filtered out, and in between a more complex behavior emerges.

3.2.3. LARGER NETWORKS

When dealing with neural networks, especially those modeled as continuous-time recurrent neural networks, the state space of the system can become quite high-dimensional. Visualizing this space directly can be challenging due to the complexity and the number of dimensions involved. To address this, Principal Component Analysis (PCA) is a powerful tool that can be employed to reduce the dimensionality of the system while retaining the most significant features [48].

PCA works by identifying the directions (principal components) in which the data varies the most. By projecting the high-dimensional data onto these principal components, we can visualize the structure and dynamics of the system in a lower-dimensional space.

To illustrate how this can be used for a CTRNN, consider a scenario where we observe the states of a larger CTRNN over time. Applying PCA allows us to reduce the high-dimensional state vectors $\mathbf{x}(t)$ into a two- or three-dimensional space defined by the first two or three principal components. This reduction can help us visualize patterns, trajectories, and attractors within the network's dynamics.

As an example we can set up a network of 10 neurons to have two distinct attractors (this can be done for example via a Hopfield Network [59]). We initialize the 10-neuron CTRNN with random states and observe that these converge to two distinct fixed points. This can be seen in figure 3.13

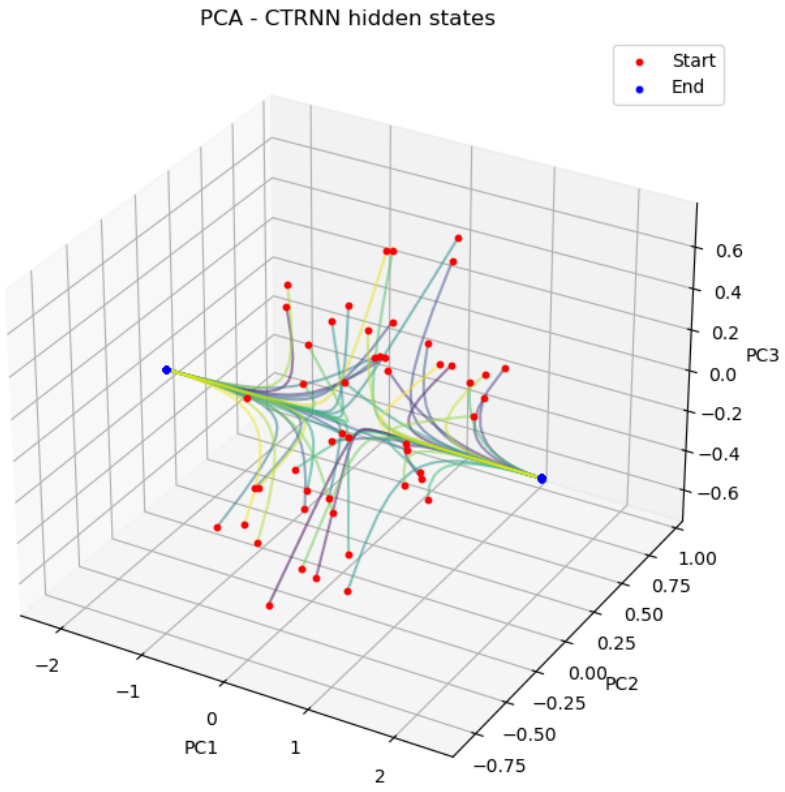


Figure 3.13: Visualization of a 10-neuron system reduced to three dimensions with a PCA transform. Red balls denote randomly initialized starting points, blue balls end points of the trajectories. The three principal components account for 99 percent of the variability in the data. The network was trained to have 2 attractors. Random states are initialized and it can be seen that over time the network indeed converges to one of the two fixed points, depending on the starting location.

PCA is particularly useful in the analysis of neural systems as it allows us to gain insights into the collective behavior of neurons. By examining the reduced-dimensional representations, we can identify dominant modes of activity, distinguish between different regimes of neural behavior, and better understand the underlying dynamics of the neural network and what it has learned, as we demonstrated with the 1- and 2-neuron systems.

3.3. CONCLUSION

This chapter examined the dynamics of continuous-time recurrent neural networks through the lens of dynamical systems. We have done so by examining the fixed points for stationary inputs, and compared these results to the state of the network when a time-varying input is applied. This has been done for a feedforward, a single recurrent, and a small network of two neurons. Some general conclusions seem to emerge.

In all cases for slow varying inputs compared to the network dynamics the state follows the stationary fixed points. Similarly, when the input is too fast the input is filtered out and the neurons no longer react to the input. The region in between shows more complex behavior, where there seems to be a transition from slow enough inputs which still follow the general trend of the stationary fixed points, to a region where the inputs are sufficiently fast and the stationary fixed points are no longer dominant for the state behavior. These results are summarized in table 3.1

Table 3.1: Summary of Network Dynamics

Region	Description
Slow Varying Inputs	The state follows the stationary fixed points. (quasi-steady-state)
Fast Varying Inputs	The input is filtered out and the neurons no longer react to the input.
Transition Region (Slow to Fast)	Complex behavior; transition from slow inputs ($\omega << \frac{1}{\tau}$) generally following fixed points to faster inputs ($\omega = \frac{1}{\tau}$). In this region, a complex mixture of fixed points and transient dynamics determines the behavior.

I hypothesize that all of these regions can be used for computational purposes. Slow varying inputs where the system directly converges to the fixed points is beneficial for example in Hopfield Networks [60], where the desired behavior is derived purely from the fixed point and no timing or transient dynamics are desired. For timing-related tasks, the transition region is most likely the most interesting. Which specific part of the region is most computationally powerful will most likely depend on the task, as it can be imagined that some tasks will lean more on timing, and others more on general fixed point behavior. The specifics remain however for future research. The region where the neurons intrinsically filter out the input of certain frequencies can also be leveraged for practical applications. For example for denoising, or perhaps with more complex filter-neurons which are tailored to remove certain parts of the frequency spectrum.

Further, the chapter briefly discussed a method of visualizing larger network dynamics through principle component analysis. This can in general be used to understand what networks learn, remove the black-box nature, and potentially aid, the design of new architectures. This technique can also be combined to gain insight into the influence of timing versus fixed point dynamics, and thus help in designing physical neural networks as well as finding appropriate tasks. An example will be given in the next chapter.

4

TASKS AND RESULTS

I will evaluate the CTRNN and the standard RNN against a set of sequence modeling classification benchmarks. The idea is to find out how a continuous time model compares against the basic RNN, and choose a suitable task for further analysis. The choice to compare against the standard RNN, and not against more advanced models such as the LSTM and GRU, is done because these have additional trainable parameters that complicate the analysis, while the main difference between the RNN and the CTRNN comes only in the form of the time constant. As such the reported accuracy's are by no means state of the art or meant to show that a CTRNN is better than for example an LSTM at all these tasks. A comprehensive comparison between CTRNNs and LSTMS/GRUs for a wider variety of tasks (classification, prediction, etc) is left for future research.

4.0.1. SEQUENTIAL MNIST

The MNIST task is a well-known benchmark in machine learning. The dataset consists of 60 thousand training, and 10 thousand testing images of handwritten digits with 28x28 pixels ranging from 0-9, as can be seen in figure 4.1. To speed up training a subset of 20 percent of the total dataset is used.

Figure 4.1: Examples of the images in the MNIST dataset [61]

The objective is to classify these digits. In sequential MNIST instead of having a 28x28 image the image is flattened to create one 784-long sequence as can be seen in 4.2. This task is often used as a benchmark for time series classification [62].

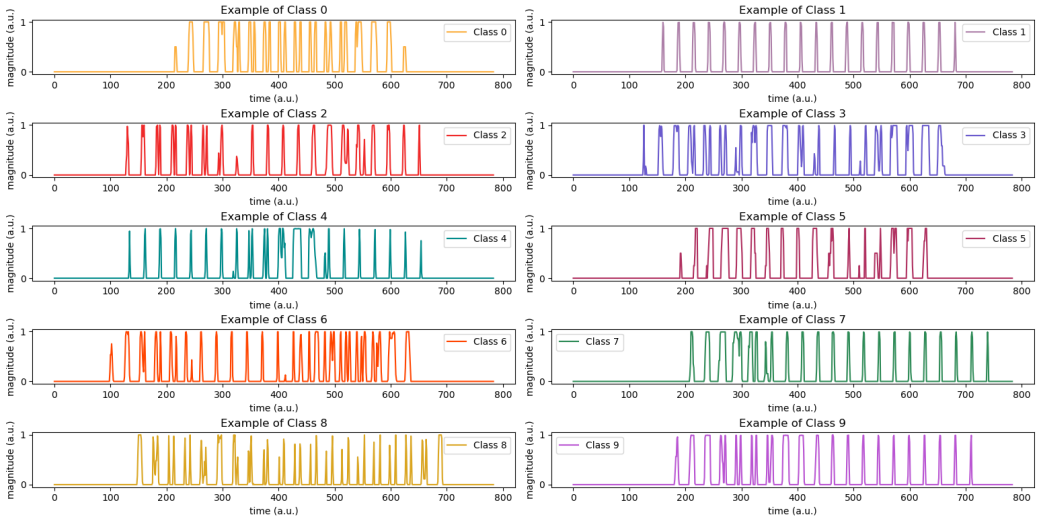


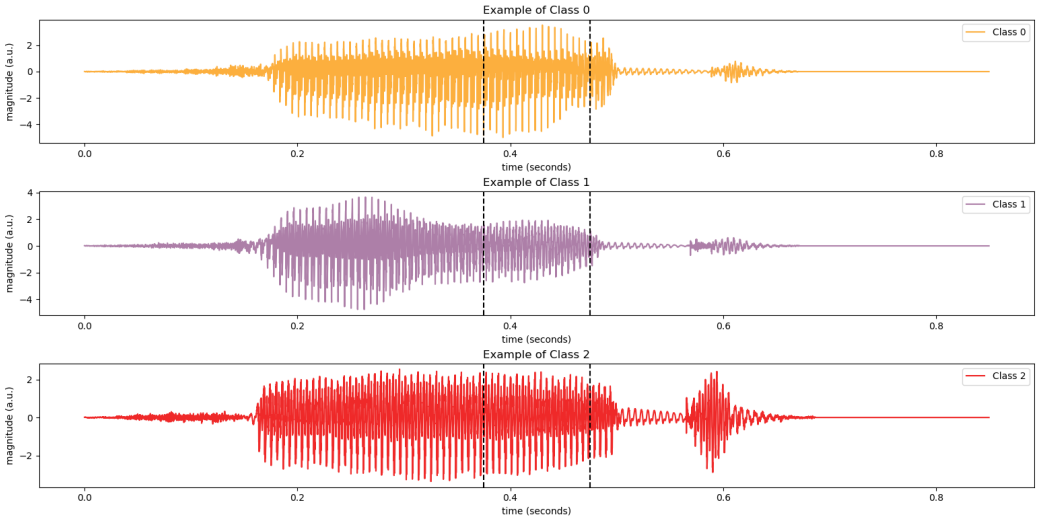
Figure 4.2: Examples of all the input digits after the 28x28 image is flattened into a 784 long sequence.

The task is to classify the 10 different output digits. To do so the neural networks will have a linear output layer of 10 neurons which represent the classes. To do so the cross-entropy loss is determined over the classes based on the last entry in the output sequence of the model.

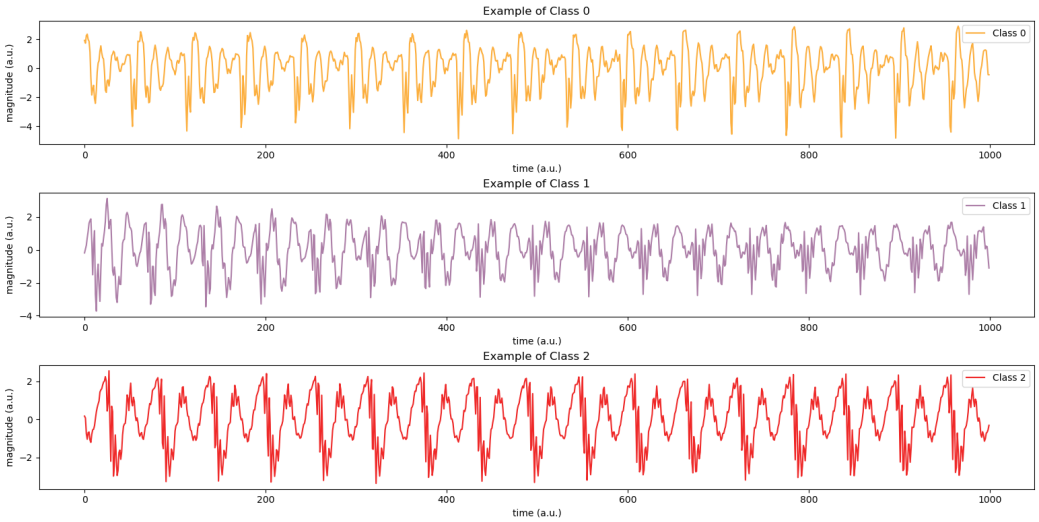
A downside of using a dataset such as MNIST is that it is not an actual time series obtained from sensor data. However, since the benchmark is very commonly used for recurrent neural networks it will be interesting to see how the CTRNN compares to the RNN on this benchmark.

4.0.2. VOWEL

In this task from 45 male and 48 female speakers 10 different vowels are uttered. The original sampling rate is 16kHz but to ease the computational burden this is downsampled to 10kHz using Librosa. The goal of this task is to classify which sequence belongs to which vowel. In this report, a subset of three vowels is chosen. Because the sequences are long, to speed up training a window from the middle of the samples is chosen, which can be seen in figure 4.3 for testing the full (unwindowed) sequences are used following [63].



(a) Example per class of the entire dataset used during training.



(b) Example per class of the windowed samples of the entire dataset used during training.

Figure 4.3: Comparative visualization of the dataset used during training, showcasing full-scale and windowed samples per class.

Three vowels correspond to three classes and therefore we use three output neurons. We use cross-entropy loss to determine which class the model predicts, but for this task do so by integrating over all the steps in the sequence as opposed to only using the last output. This changes what the model needs to learn. In the previous two tasks the model needed to learn the class, but also to ensure the prediction could be read out at the end of the sequence. Using the entire sequence does not put that burden on the model.

4.0.3. FORD A

In this task, a single input sequence is given to the model which comes from a sensor measurement of an engine. The dataset consists of a set of 4801 sequences of length 500. These sequences correspond to one of two classes: 1) the engine is working; 2) the engine has an issue. The objective of the model is to classify which sequences belong to which class, an example of this is shown in figure 4.4. The FordA data was collected in a typical setting and contained little noise. The dataset is supplied normalized with zero mean and unit variance, as can be seen in figure 4.4

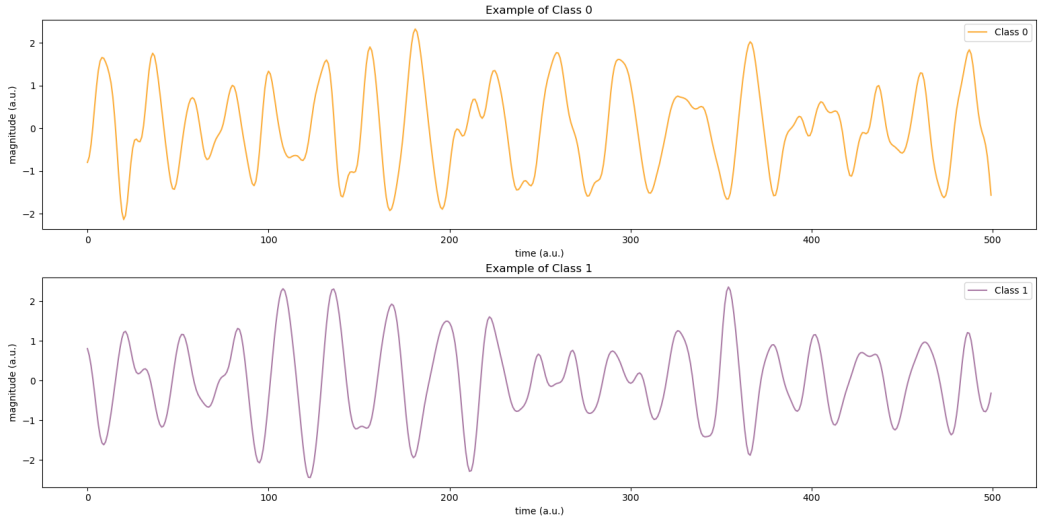


Figure 4.4: Example of the FordA input sequence for the two classes.

When an input sequence is fed into the model the task of the neural network is to learn which class it is. Since there are only two classes, this was done by using only a single output. The last entry in the output sequence was then passed through a Sigmoid function and a binary cross-entropy loss. Unfortunately, no sampling rate or other specifics of the sensor where given. It should also be noted that the zero mean and unit variance is a form of preprocessing which would not be present for direct on-sensor applications.

4.1. RESULTS

The vanilla RNN and the CTRNN are compared against one another. In all experiments in this section, the input sequence is fully connected to the neurons. Similarly, the recurrent connections are also fully connected, meaning each neuron is recurrently connected to itself, and all the other neurons. The outputs of the neurons are connected to a linear output layer which reduces the dimension to the desired output class. A schematic of the architecture can be seen in 4.5.

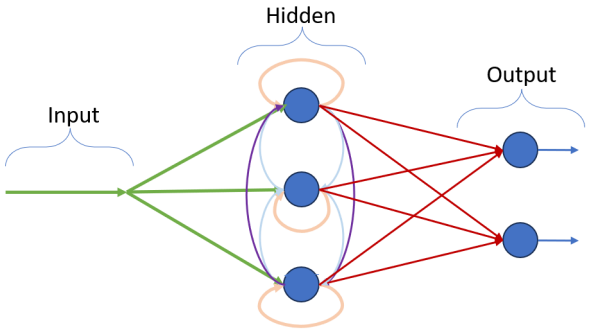


Figure 4.5: Schematic of a fully connected RNN with a hidden size of 3 neurons, and an output layer of two neurons. Green represents the input, orange recurrent self-connections, light blue and purple recurrent connections to other neurons in the layer, red the hidden output layer, and blue the output.

The hyperparameters of the models are optimized using Optuna [64]. The optimization is performed on DelftBlue, the TU Delft supercomputer [65]. During optimization, the time constant was optimized for the range [2, 300]. These time constants are taken assuming a sampling rate of $dt = 1$ for comparability

between tasks, thus the reported time constants are a relative measure. The full list of hyperparameters and ranges can be found in appendix C.

4.2. OVERVIEW RESULTS

Table 4.1: Evaluation of Sequence Modeling Performance

Task	Accuracy %	# Neurons	τ
Seq. MNIST			
RNN	61	100	n.a.
CTRNN	87	100	92
FordA			
RNN	69	75	n.a.
CTRNN	93	75	2
Vowel			
RNN	100	25	n.a.
CTRNN	97	25	2

An overview of the best-performing models can be found in 4.1. It should be noted that none of the results are state-of-the-art, and are solely reported as a comparative measure against a simple RNN.

It is interesting to note that for sMNIST and FordA the CTRNN outperforms the vanilla RNN greatly. For the vowel task, however, the RNN performs best, which is unexpected. This could be a sign that the task is trivial.

4.3. BENCHMARK RESULTS ON TASKS

4.3.1. SEQUENTIAL MNIST

The convergence plots of the best performing models (RNN, CTRNN) can be seen in figure 4.6.

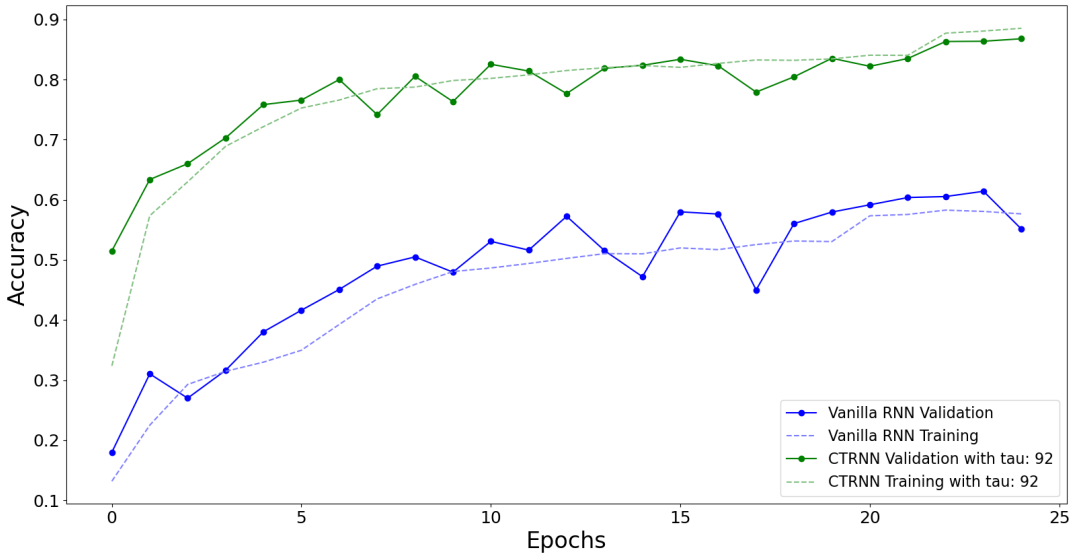


Figure 4.6: Convergence plot for the MNIST task. Plotted are the vanilla RNN and the CTRNN..

The CTRNN has better performance as compared to the standard RNN. At the same time, we can note that a relatively high time constant shows optimal performance. This is in direct contrast to the other two tasks. The most likely cause is that the sMNIST dataset represents more pulse-like data, rather than the continuous nature of the other two datasets. It seems that this promotes slower neurons for better performance. A thorough investigation of why this happens is however outside of the scope of the report and is left for further research.

4.3.2. VOWEL

The convergence plots of the best-performing models can be seen in figure 4.7.

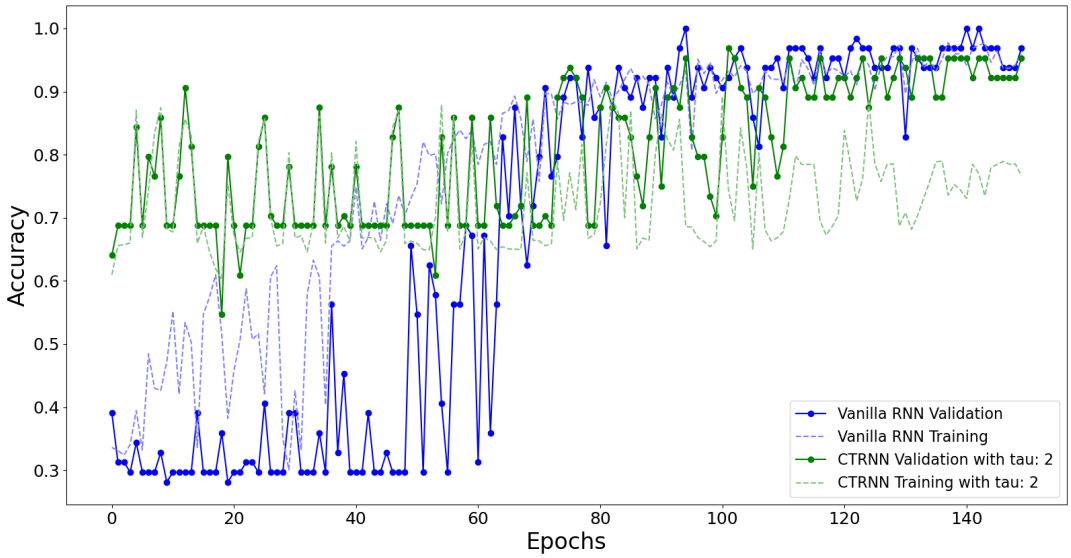


Figure 4.7: Convergence plot for the vowel task. Plotted are the vanilla RNN and the CTRNN

It is interesting to note that here the vanilla RNN shows the best performance and that it shows more stable convergence than the CTRNN. This is unexpected. A potential cause could be due to the class prediction being done via the integration of all time steps to perform classification. This could promote shorter-term correlations but should be further investigated by training the model with the prediction at the last step and comparing results. However, one should take care of the variable sequence lengths and how that affects the results of such an experiment.

Further, it can be observed that for the CTRNN the training and validation curves separate after approximately 80 epochs. In contrast to what one would expect with overfitting, the validation accuracy is higher than the training curve. This might suggest a simpler validation set, possibly a result of the small dataset or other issues with the model. If the cause is a too-small dataset, this should decrease with a larger dataset size or with data augmentation techniques.

4.3.3. FORD A

The convergence plots of the best-performing models can be seen in figure 4.8.

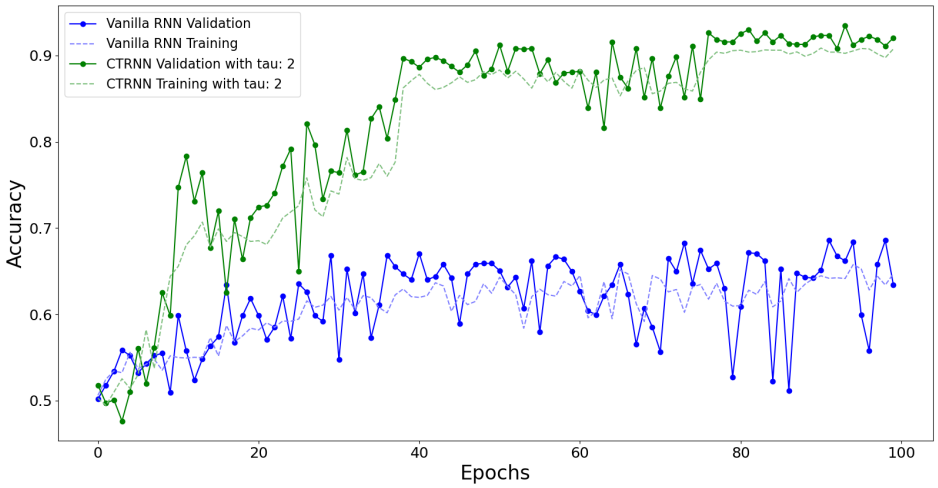


Figure 4.8: Convergence plot for the fordA task. Plotted are the results for the vanilla RNN and the CTRNN.

The CTRNN shows the best performance. While it seems that the standard RNN can learn to predict some examples correctly, it fails to predict more than 70 percent correctly. Since from the three tasks, FordA shows the most consistent results and is also a suitable practical example for a direct on-chip sensor application, we will examine this task in more detail.

4.4. IN-DEPTH ANALYSIS FORDA

We will use the insights and techniques from chapter 3 to better understand how the CTRNN works when trained on an actual task. We will aim to use those insights to further make design and application choices for on-chip photonic CTRNNs.

First, we investigate the performance of the network for different time constants. The entire hyperparameter optimization was run where the constant was also optimized for. Figure 4.9 shows these results.

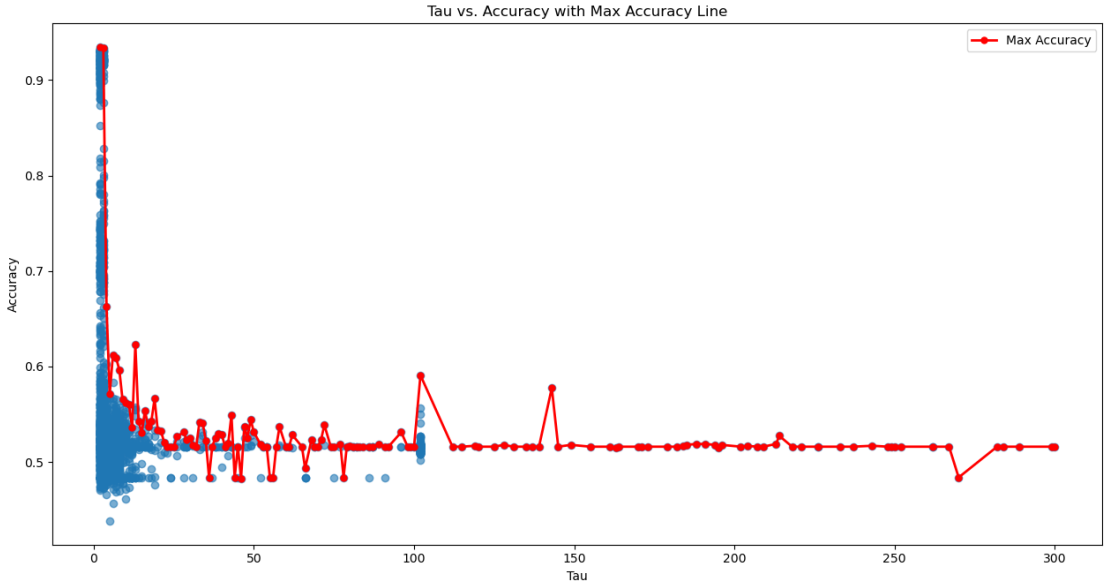


Figure 4.9: Convergence plot for the FordA task. Plotted are the results for the vanilla RNN and the CTRNN.

From the figure, it is directly clear that the performance of the model drastically declines as the time constants become larger. In chapter 3 the low-pass filtering behavior for high-frequency inputs was established. We can investigate the frequency spectrum of the input signal and compare regions of interest to the optimal time constants. Since the actual sampling rate is unknown for the FordA task we assume $dt = 1s$ and compute the mean energy spectrum for the signals in each class as shown in figure 4.10.

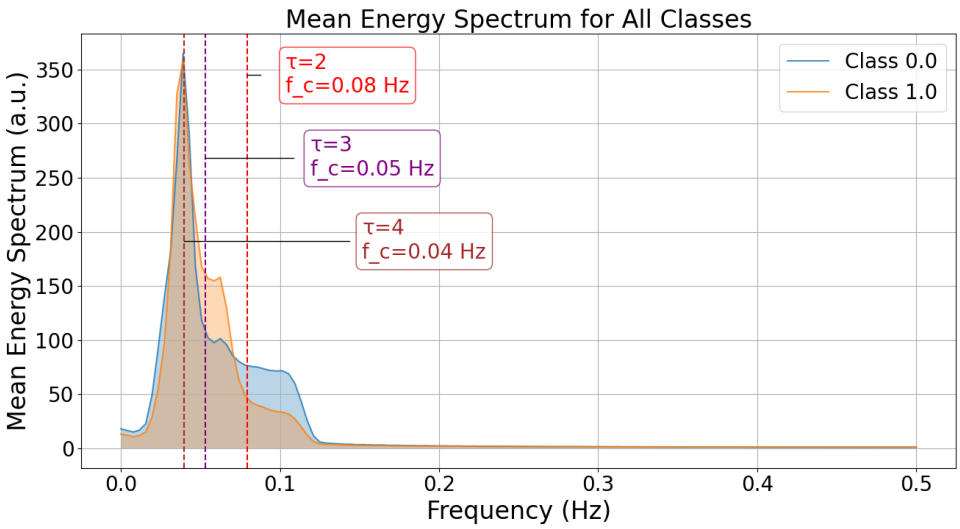


Figure 4.10: Convergence plot for the fordA task. Plotted are the results for the vanilla RNN and the CTRNN.

Understanding that the CTRNN behaves as a low-pass filter it becomes clear why larger time constants, i.e. slower neurons, have bad performance. It can be seen that for a time constant of $\tau = 2s$ the cutoff frequency of the low-pass filter is already near the part of the frequency spectrum containing

all the information. It is unfortunate that due to the finite sampling rate, we cannot investigate much faster time constants of the neuron relative to the input signal. This would have given a more complete picture and potentially a lower bound. We can however note that since the optimal time constant is close to the input frequency of interest ($\omega \approx \frac{1}{\tau}$) it is not operating in the region close to the quasi-steady state. This indicates it has most likely learned transient dynamics.

We can examine what the model has learned by using PCA analysis on the state of the neurons and following the trajectories.

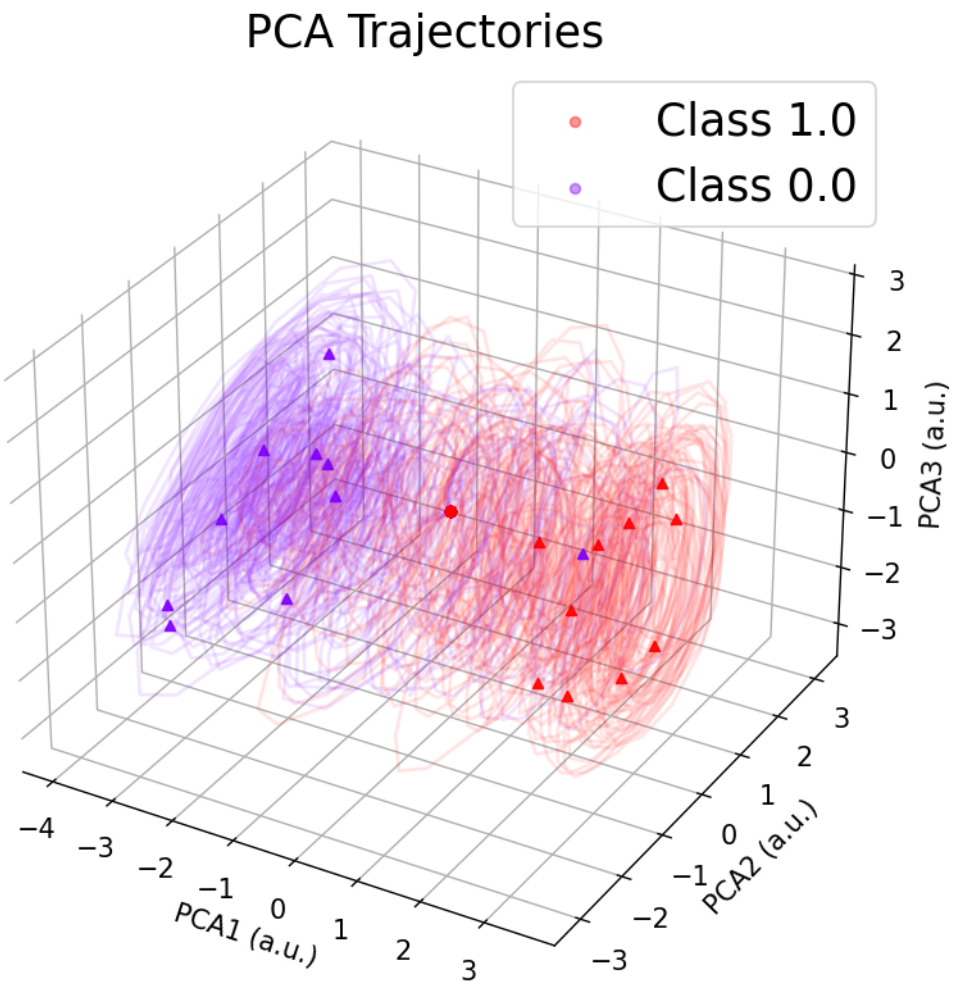


Figure 4.11: Examples of several trajectories of the state of the CTRNN during the FordA task tranformed into three dimensional PCA-space. Balls indicate starting points, triangles denote the final locations of the trajectories. We can see that the network can guide the different input classes to distinct regions in space well.

Figure 4.11 shows the state of the 75 neurons reduced to the three principal components. This accounts for 80 percent of the variability in the data and as such is a relatively accurate representation. It can be seen that the network can separate the classes well, and does so by creating two distinct regions in space.

We can examine if these regions correspond to fixed points by adding a zero padding at the end of input sequences and allowing the network to settle into steady state. Indeed, we then see the network has learned two distinct fixed points corresponding to the classes as shown in figure 4.12. Interestingly doing so reduces the accuracy to 84.5 percent, where 50 percent is the trivial solution. The model is still

able to separate the classes very well, even after any input is removed due to the learned fixed points. This accuracy is maintained for an arbitrarily long delay.

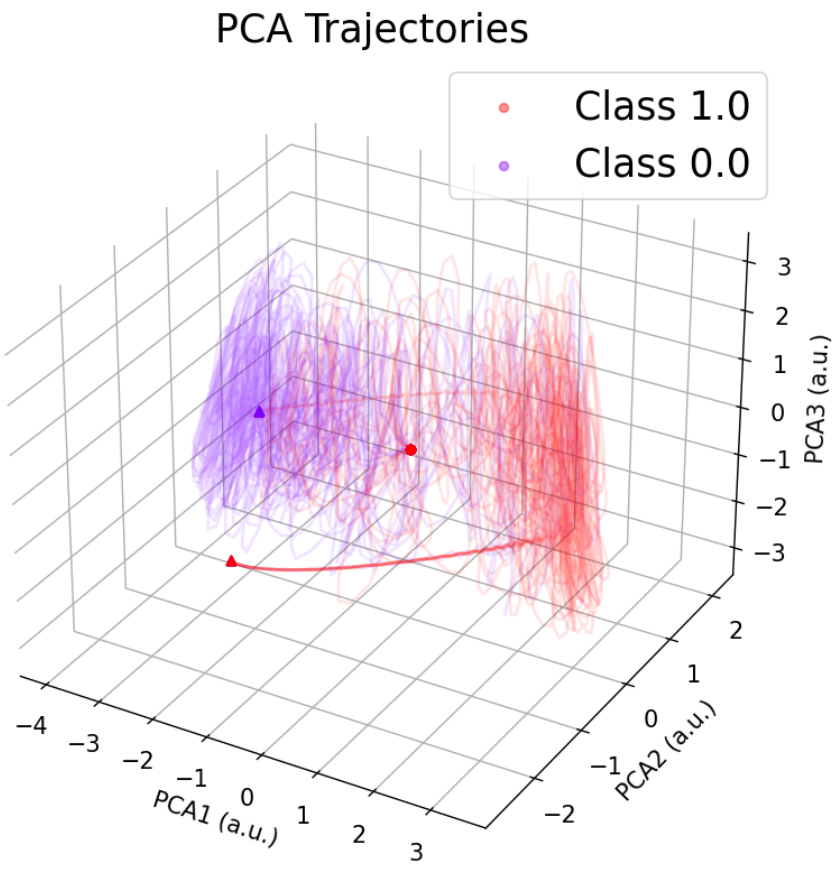


Figure 4.12: PCA trajectories of the state of the CTRNN during the FordA task with an added delay of T=500. Triangles denote the final locations of the trajectories. After the input is removed we can see that the state relaxes to two distinct fixed points. Balls indicate starting points, triangle endpoint. Notably, the accuracy of the model is still 84.5 percent after an arbitrarily long delay.

In a similar fashion to the analysis performed in chapter 3 we can plot the stationary fixed points for a certain input trajectory. Here we do not choose a simple sinusoidal input but follow one of the actual input trajectories directly. We select an example from class 0 and plot this in PCA space. For each point of the input, we also calculate what the fixed point would be if that input was a stationary input given to the network. The result is shown in 4.13

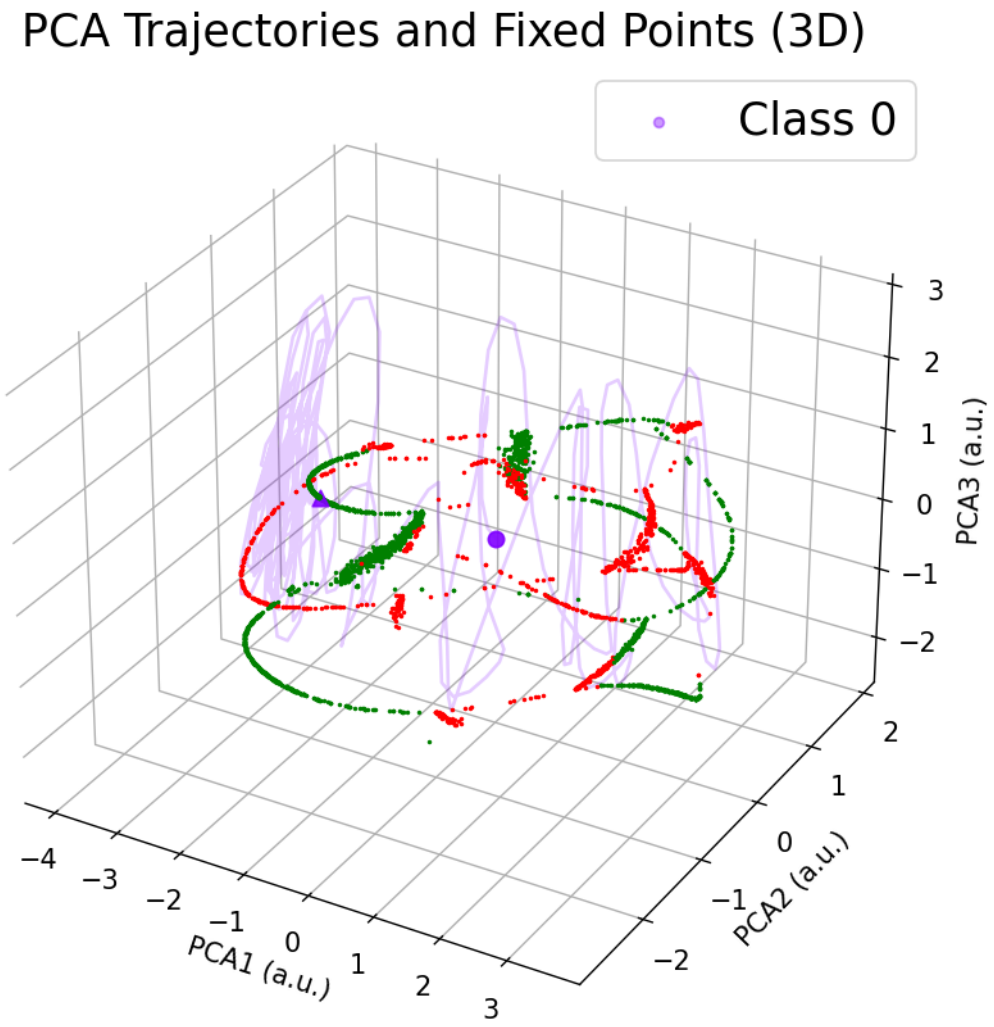


Figure 4.13: PCA analysis including stationary fixed points for a single example trajectory. Green indicates stable fixed points, red unstable fixed points. Fixed points are computed at every step in the trajectory. Purple ball indicates starting point, purple triangle endpoint.

The most important observation we can make is that the trajectory is not closely following the stationary fixed point trajectory. We also know that the input for this model is not filtered out. We are thus in a region where transient dynamics are important. It is therefore very likely that we cannot make the time constant arbitrarily small (i.e. network arbitrarily fast) for such a task, as the transient dynamics and timing are important for proper classification. This is important for photonic applications, as this implies that the temporal input dynamics must be properly matched to the network time constant. There exists a chance that reducing the time constant further to where the input signal becomes slow relative to the network dynamics still allows for proper classification. I think this however to be unlikely and even though a faster time constant than demonstrated in this report will most likely still show good results, they cannot be made arbitrarily fast. To make a definite claim on this, however, the experiments should be performed with much higher sampling rates and examine how this affects performance and what the network learns. The methods introduced in this report can be helpful when conducting such a follow-up study.

4.5. WHAT DOES THIS MEAN FOR PHOTONIC NEURAL NETWORKS

I would like to close this chapter with a short discussion of what the findings imply for a real-world photonic neural network implementation.

It is clear that low-pass filtering plays a major role in the capabilities of these networks. Knowing the input signal characteristic places a bound on how slow we can make the neurons in the network.

On the other side we have seen from the PCA analysis on fordA that while attractor dynamics play a major role, the dynamics are not purely attractor based but show strong signs of transient dynamics which are affected by the input.

These two points, together with the findings from chapter 3 allow us to make estimates regarding the coupling of the input frequency characteristic and the photonic neural network.

In chapter 2.1 it was stated that photonic neurons can have processing speeds as fast as the order of 100 picoseconds. We therefor take $\tau_{neuron} = 100ps$. Signals one order of magnitude above $\omega_{upper} = \frac{1}{\tau_{neuron}}$ will surely be filtered out. This gives an approximate upper bound. While the lower bound is more difficult to establish, we have seen in chapter 3 that in the cases we examined slow dynamics were achieved for $\omega_{lower} < 0.01\frac{1}{\tau_{neuron}}$. At this point the network operated in quasi-steady state. Converting this to a frequency amounts to frequencies $f_{slow} < \frac{\omega_{lower}}{2\pi} \approx 10$ GHz that are considered slow from the perspective of the network. This is still incredibly fast for most applications. If we slow to network down to for example 1 μs setting $\tau_{neuron} = 1\mu s$, we obtain $f_{slow} \lessapprox 100$ kHz. While still fast this is a much more practical range. It has been stated that lowering the speed decreases energy consumption, so theoretically there may be a beneficial tradeoff for some applications [3]. Concluding, we can thus propose that generally for continuous-time tasks CTRNNs perform best when $\frac{1}{100\tau_{neruon}} \lessapprox \omega_{task} \lessapprox \frac{10}{\tau_{neuron}}$.

On the other side purely attractor based dynamics can be leveraged where this extremely fast converge is desired, rather than problematic. Examples are Modern Hopfield Networks [60], Model Predictive Control [66], pathfinding algorithms [67], and potentially other tasks requiring rapid and frequent conversion such as Bayesian Neural Networks.

5

CONCLUSION & DISCUSSION

5.1. DISCUSSION

The report aimed to gain insights from understanding the dynamics of continuous-time recurrent neural networks and how these insights can be leveraged for designing and finding use cases for the photonic on-chip counterpart. It has been demonstrated that a large range of dynamics in general is possible, even for smaller networks. The complexity rapidly increases for larger networks. Understanding the smaller networks has, however, given insight into how these CTRNNs function. This has been coupled to larger networks in general, photonic implementations, and photonic applications. In this section of the discussion, I will list some of the key points and findings and make recommendations on how best to use the gained knowledge.

5.1.1. THE TIME CONSTANT

It has been shown that the time constant affects the general speed of the network. This means how fast the neurons react to inputs, and how quickly the network decays. It does not however affect the qualitative behavior of the network under no input conditions, but does so depending on how fast the input changes. In the general case of on-chip photonics typically we are operating at extremely high speeds, i.e. small time constants. This in turn implies that the network very quickly converges to its steady state. From chapter 3 we know that this only changes if the input is sufficiently fast compared to the time constant. I propose two approaches to leverage these insights:

First, we can slow the photonic neurons down. If one constructs a photonic neural network with an opto-electro-opto architecture the dominating time constant can be controlled, and made sufficiently slow, via a low-pass filter in the electronic domain. A demonstration of how this can be done can be found in appendix D. When doing so, however, we also reduce the inference speed of the network. While in principle there are other strong benefits to photonic neural networks, such as high parallelism and low loss/power, simply slowing the network down to match the task seems somewhat counterproductive and a more natural match such as with pure electronics or MEMS should be examined. However, it is possible to use different time constants for each neuron. I expect that this will in general improve the performance of CTRNNs. At the same time one can make more complex multiple time constant architectures which, together with the aforementioned other benefits of photonic neurons, could have promising results [35];

Secondly, the understanding that neurons can behave as filters, and that these are relatively easily controlled in the electrical domain, allows for interesting applications. This allows us to think about more complex filter neurons [55]. This is a novel suggestion for photonic neurons. One can imagine that having specific neurons only react to certain frequency bands, or ignore certain parts of the spectrum, can have useful applications. This behavior might be directly combined with sensing creating in- or near-sensing photonic computing, which still benefits from high parallelism and low energy consumption. In principle this can be combined with the first point.

5.1.2. ATTRACTOR DYNAMICS

We have seen that even for small networks a large variety of different dynamics in state space is possible. This complexity only increases with the growing size of the network. An interesting observation has been that for example for the FordA case, the network turns out the learn two distinct fixed points corresponding to the two distinct classes. This stands, to my knowledge, in direct contrast to for example reservoir computing, where the network (reservoir) dynamics are only changed upon initialization and thus cannot form these task-specific attractors. As most of the research for continuous-time neural networks and the link to the time constant has been performed for reservoir computing, this could be investigated further and more thoroughly. This can be done for a wider variety of tasks, different time constants for each neuron, and using the principle component analysis techniques to make a direct comparison to what these networks learn, and where and how exactly the time constant plays a role.

As seen in the FordA task, a network with learned fixed points, instead of purely transient dynamics, can theoretically retain its state indefinitely. This can in principle ease timing burdens for classification tasks even though the network dynamics are very fast. Another potential way to leverage this is by looking at the question from another vantage point. We have seen that the time constant does not affect the qualitative behavior of the network, purely the overall network speed. Where initially the question was posed how to match these dynamics to the input, we can also look at situations where the desired behavior is extremely fast convergence towards the steady state, for which photonics would be very suitable. It turns out there are quite a few interesting problems that require this behavior:

- (Modern) continuous Hopfield networks [60]. As briefly mentioned in chapter 3 Hopfield networks are designed to operate in steady state. The attractor dynamics of the network serve as a form of associative memory. Originally they were created with binary activation functions, but recently the attention has shifted towards continuous activation functions, and Hopfield Networks operating in continuous time. It has been shown that this greatly enhances their memory capabilities. Perhaps most interestingly, Modern Hopfield Networks have been proposed as alternative forms of attention mechanisms and alternatives to transformers (one of the most important techniques behind modern Large Language Models such as ChatGPT) [68]. To the best of my knowledge, this has not yet been proposed. The best starting points would be with a recent architecture proposed by Hopfield himself, which in contrast to other Modern Hopfield Networks, takes physical realizability into account [68]. I my opinion this is a very promising novel application leveraging many qualities of photonic neural networks.

- The extremely fast convergence of photonic neural networks can be leveraged for time-sensitive tasks. In such a case it is not the time constant of the physics, but quickly and repeatedly doing the computation that is beneficial. Certain optimization problems, and control problems, such as for example MPC or Bayesian Neural Networks could benefit from this [66], [67], [69]. It should be taken into consideration that this most likely would require fast switching of the weights, which could be the limiting factor. Regardless, it is certainly worthwhile to investigate these directions more thoroughly as they seem to match directly with the ultra-fast speeds associated with PNNs.

5.1.3. GENERAL CONSIDERATION

Lastly, I would like to point out general considerations which were not taken into account during this report, but in my opinion, do require attention.

Physical realization of larger networks, how these can be manufactured, and how errors and noise and other physical constraints affect the functioning of these networks. Smaller networks have been realized and demonstrated, and in some cases, some attention has been spent on noise and errors, but overall the networks have remained quite small. If we want to make photonic neural networks into a practical reality this is as far as I am concerned a top priority.

One of the guiding strengths behind the rise of machine learning has been backpropagation. Unfortunately, this is generally speaking difficult to on-chip as it requires full knowledge of the gradient in all the nodes [70]. As training can be very time-consuming, and energy-draining, good on-chip training solutions are required. Especially when the sequences become long, both computational time and

complexity increase and would benefit from in-chip training. This is an active research field, but new ideas and actual physical demonstrations are most welcome. To this end, I believe that understanding the dynamics of these networks theoretically, and physically, may aid us in finding new solutions.

5.2. CONCLUSION

This report has delved into the dynamics of continuous-time recurrent neural networks (CTRNNs) and the link to their photonic on-chip counterparts, providing insights into their behavior, applications, and performance across various tasks. The exploration included an examination of state-of-the-art photonic neural networks (PNNs), the fundamental dynamics of CTRNNs, and their performance on practical tasks such as sequential MNIST, FordA, and vowel classification.

In the analysis of CTRNNs, it was shown that the time constant plays an important role in determining the speed and filtering behavior of the network. Through a series of experiments with single neurons and small networks, it was demonstrated that slow-varying inputs allow the state to follow stationary fixed points, whereas fast inputs are filtered out. In between these points the most complex behavior is found with a mix of attractor and transient dynamics. This relationship between the input frequency and network dynamics was important for understanding how CTRNNs process information, and providing insight for proposing novel applications

When evaluating the CTRNNs against standard RNNs on sequence modeling tasks, it was found that CTRNNs often outperform RNNs. In the sequential MNIST task, CTRNNs demonstrated superior performance, likely due to their capacity to effectively handle sequential pulse-like data. Conversely, in the vowel classification task, standard RNNs performed better. This could be attributed to the simplicity of the data, but it might also be due to the loss being computed by integrating over all time steps, unlike the other tasks where the loss was based on the last time step. This suggests that the nature of the task, including how temporal information is processed and integrated over time, plays a significant role in determining which type of neural network performs best. It also indicates that not only the network architecture but also the methods used for temporal loss computation can significantly impact performance.

The in-depth analysis of the FordA task highlighted the practical implications of time constants in CTRNNs. It was observed that the optimal performance was achieved with smaller time constants, aligning with the frequency spectrum of the input signals. Due to the finite sampling rate of the data a fast dynamics could not be examined. Through PCA analysis of the trained network it was found that the FordA task has learned two distinct fixed points separating the classes. A further analysis of state-space trajectory showed that most likely both the transient and attractor dynamics play an important role. It was estimated that for continuous-time tasks perform best when $\frac{1}{100\tau_{neruon}} \lessapprox \omega_{task} \lessapprox \frac{10}{\tau_{neuron}}$.

In conclusion, this report has underscored the potential of CTRNNs and PNNs in various applications, providing a foundation for further research and development. The findings suggest that while photonic neural networks hold promise for ultra-fast, energy-efficient computing, their design must carefully consider the temporal dynamics of the target tasks. Future work should either focus on leveraging the filtering capabilities of the neurons and extending this with more complex filters, or different time constants; or focus on using the fast convergence to steady for tasks that require fast convergence or frequent updating.

ACKNOWLEDGEMENTS

I would like to start by thanking Dr. M.A. Bessa and Dr. R.A. Norte for the opportunity to work on this project, letting me explore the topics, and their guidance throughout. I have had many interesting discussions throughout the year and learned a lot.

I would like to thank my father for his help and support throughout the years. It hasn't always been easy, but we finally got there. Lastly, to my mother, you motivated me to keep going even though you weren't there. If only you could have seen.

A

Brief Background on Neural Networks

Artificial neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain -albeit a simplified model. They are composed of layers of interconnected processing nodes, or neurons, that receive inputs, process them, and produce outputs. Each neuron receives inputs from other neurons or directly from the input data and applies an activation function to produce an output. In this section, a very basic description of feedforward and recurrent neuronal networks is given as a background for further chapters. Most of this section is based on [29], [71]

A.0.1. Feedforward Neural Networks

A feedforward neural network is a type of neural network where the information flows in one direction, from the input layer through one or more hidden layers to the output layer. The output of one layer is used as the input to the next layer. The input layer contains neurons that receive the input data, while the output layer contains neurons that produce the final output. Figure A.1 shows a visualization of a feedforward neural network.

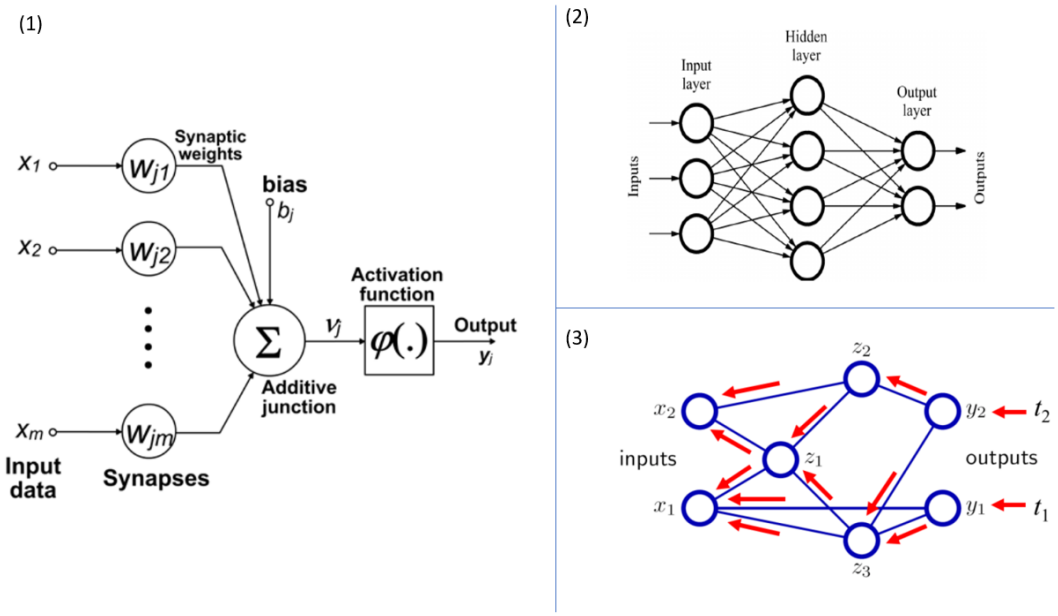


Figure A.1: Visualization of simple feedforward neural network. (1) Shows the model of a simplified neuron. (2) Shows a feedforward neural network with one input layer, one hidden layer, and one output layer. (3) Shows the general idea of backpropagation. [72]

The hidden layers contain neurons that perform intermediate computations. Neurons and layers can be connected in a variety of ways for specific computations. The output of each neuron in a feedforward neural network is calculated by taking a weighted sum of its inputs, adding a bias term, and applying an activation function. The weighted sum is calculated as follows:

$$y_j = \varphi(\sum_{i=1}^{m} w_{ij} x_i + b_j) \tag{A.1}$$

where y_j is the weighted sum of the inputs to neuron j after the activation function, w_{ij} is the weight of the connection between neuron i and neuron j, x_i is the output of neuron i, b_j is the bias term of neuron j, and $\varphi()$ represents some nonlinear function. Similarly, the equations can also be written in a vector-matrix multiplication form.

The activation function is a nonlinear function that introduces nonlinearity into the output of the neuron. Some commonly used activation functions are the sigmoid function, the ReLU (rectified linear unit) function, and the hyperbolic tangent function.

Without nonlinear activation functions, the neural network would be limited to performing only linear transformations of the input, which are too simple for many real-world problems. The nonlinear activation is a key feature that makes a neural network a universal function approximator [71]. Nonlinear activation functions introduce the ability to model more complex relationships between inputs and outputs, allowing for better accuracy in prediction or classification tasks.

A.0.2. HOW TO TRAIN YOUR NEURAL NETWORK?

The basic premise of a feedforward neural network is transforming the input data into a function that underlies the data. Here I will only discuss supervised learning, where the desired output is known for some input. Other categories exist such as unsupervised learning, where the network is not explicitly given the right answer; or reinforcement learning, where the network is allowed to interact with its environment and punished or rewarded according to rules. These are however not discussed.

In supervised learning the training data consists of labeled input and output pairs. When an untrained neural network is then presented with the input data it is very unlikely that the output of the

neural network represents the desired output. The first step is to determine a loss function that compares the output of the neural network to the labeled training data. A well-known loss function is the mean squared error, but this is by no means the only one and it can depend on the task what function is best suited to compare the predicted and desired output. It is then the job of the training algorithm to change parameters, namely the weights and biases, to minimize the loss function. While there are many different training algorithms, probably the best-known and most used training algorithm is backpropagation. This is a gradient descent algorithm that tries to minimize the error function by, in essence, the repeated application of the chain rule from calculus, a visualization can be found in Figure A.1. It should be mentioned that when using a gradient descent algorithm the error function should be differentiable.

While there are many subtleties and difficulties when training a neural network, I will only specifically mention overfitting here. A neural network is trained on a training dataset. The goal is then to train the neural network with the limited training dataset while still performing well on data it has not seen before. Before it was mentioned that the goal of training is to minimize the error function, however, minimizing it too much can also be problematic. Fitting too well with the training data may result in reduced performance on the test dataset, i.e. the model loses generalizability. The general idea is shown in Figure A.2

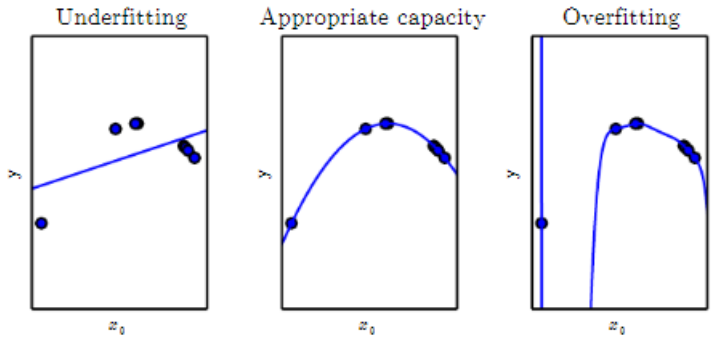


Figure A.2: General idea of under- and overfitting shown by a polynomial fit on some data points. The first image shows an underfitted model, the middle an appropriate fit, and the third image is an overfitted model. [29]

A.0.3. Recurrent Neural Networks

As mentioned, feedforward neural networks are universal approximators. However, the relation is static in the sense that there is no time component involved. The input is mapped into the output, but the neural network does not 'know' what came before. One can use a feedforward neural network for time series using some sampled time window as an input but the time window is fixed and must be large enough to capture the dynamics of interest properly.

A recurrent neural network (RNN) is a type of neural network that is designed to handle sequential data. It has loops that allow information to be passed from one step of the sequence to the next. The output of a neuron in an RNN depends not only on the current input but also on the previous inputs and the internal state of the network. The recurrent loop changes the neural network from a function approximator into a dynamical system. There are many interesting things to say about the nonlinear dynamics of RNNs, but these are outside the scope of this introduction. I refer to [71] for a nice introduction to the ideas. A recurrent neural network is commonly represented either as the recurrent loop or unfolded in time, as can be seen in figure A.3.

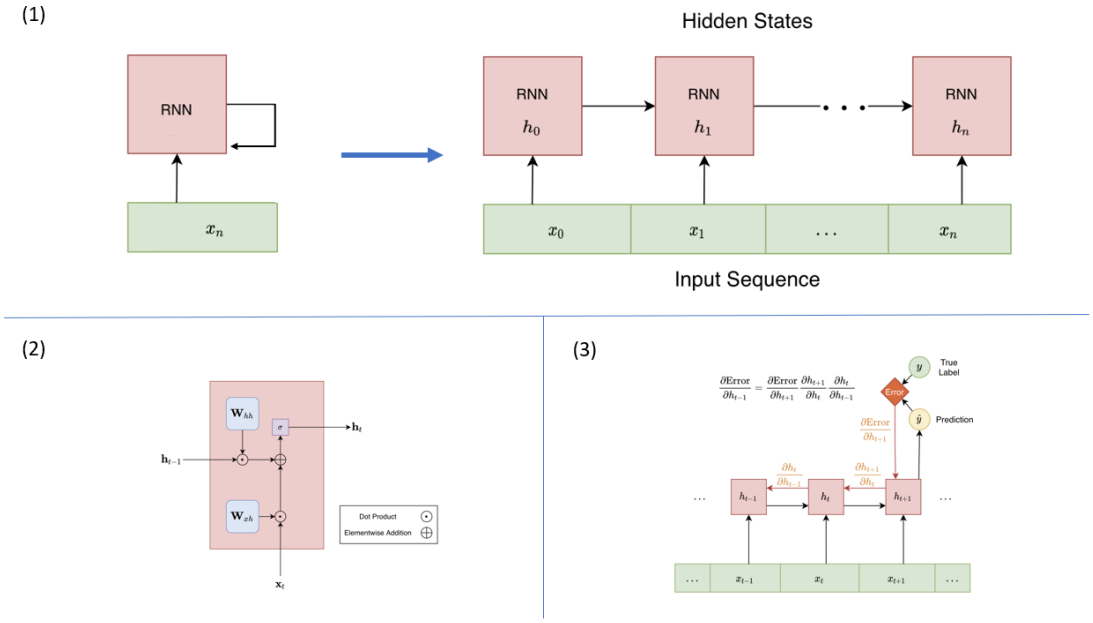


Figure A.3: (1) Visualization of both the folded and unfolded vanilla RNN. The unfolded representation presents the recurrent connection as, in essence, a very deep neural network. (2) A representation of a vanilla RNN cell. (3) A visualization of backpropagation through time. [73]

The equation of an RNN layer can be expressed as follows:

$$\boldsymbol{h_t} = \sigma(W_{hh}\boldsymbol{h_{t-1}} + W_{xh}\boldsymbol{x_t} + \boldsymbol{b_s}) \tag{A.2}$$

where $\boldsymbol{h_t}$ is the hidden state at time step t, x_t is the input at time step t, y_t is the output at time step t, W_{hh} is the weight matrix for the connections between hidden states at time steps $t-1$ and t, W_{xh} is the weight matrix for the connections between inputs and hidden states at time step t, $\boldsymbol{b_y}$ are the possible bias terms, and $\sigma()$ is a nonlinear activation.

The ability of RNNs to maintain a memory of past inputs and to use this memory to inform future predictions makes them well-suited for tasks such as natural language processing, speech recognition, and time series prediction. However, it is well known that RNNs can suffer from the problem of vanishing gradients during training, which can make it difficult for the network to learn long-term dependencies. This issue arises when training an RNN with backpropagation through time, which is essentially the same as backpropagation as discussed before on an unfolded RNN. The unfolding creates in essence a very deep feedforward neural network where the consecutive layers have the same weights. When the error is then backpropagated through this network the gradient can explode or vanish depending on the size of the weights. The gradient can reach 0 as fast as in 10 to 20 steps [29]. This has led to the development of variants of RNNs, such as long short-term memory (LSTM) and gated recurrent unit (GRU) networks, which are designed to address the vanishing gradient and improve the performance of RNNs on tasks involving long-term dependencies [73].

Another approach is to tackle the issue of efficiently training RNNs over long sequences via training method. To mitigate the computational burden and the problem of vanishing and exploding gradients encountered during the training of RNNs, a technique known as Truncated Backpropagation Through Time (TBPTT) is employed. Unlike standard backpropagation through time, which unrolls the entire sequence to compute gradients, TBPTT limits the number of steps the network is unrolled, and the gradients are propagated back. This approach not only addresses the computational inefficiency but also helps in managing the gradient issues to a certain extent. The idea can be seen in figure A.4.

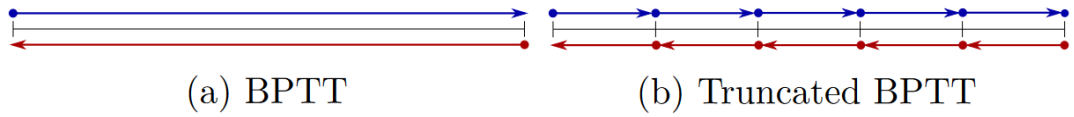


Figure A.4: A demonstration of the idea of truncated backpropagation through time, adapted from [74].

The TBPTT method involves partitioning the input sequence into smaller, manageable segments. For each segment, the network is unrolled for a fixed number of timesteps, and gradients are calculated and propagated back only through this truncated sequence. Mathematically, if we consider a segment of length k, the network is unrolled for k timesteps, and the update equations for the weights are applied based on the gradients computed within this window. The state at the end of each segment is then used as the initial state for the next segment, maintaining a form of continuity across the sequence despite the truncation.

By focusing on shorter sequences, TBPTT reduces the risk of vanishing and exploding gradients, as the depth of the computational graph over which gradients are computed is limited. Furthermore, it improves training efficiency by reducing the memory requirements and computational load, making it feasible to train RNNs on longer sequences. However, the choice of the k is critical; too small a value may not capture long-term dependencies effectively, while too large a value may reintroduce the original problems of computational complexity and gradient instability. [74]

A.0.4. Discrete-Time Versus Continuous Time

The difference between continuous time (CT) and discrete-time (DT) is important to mention. Most of the standard literature on machine learning is based on discrete time. However, in this report, the goal is to directly use the CT signal from the sensor as an input to a physical neural network. The physical neural networks are by nature CT. There are big differences in the dynamics of a DTRNN and a CTRNN. In section 2.1 it will be seen that photonic RNNs are typically CTRNNs. At this point, I will briefly mention that where in DTRNNs the recurrent input is always that of the previous step, in CTRNNs there is a notion of a time constant of the neuron. This time constant is related to how the neuron responds to new inputs, but also to how long information is retained. This allows the use of the time constant as a parameter which can strongly influence the performance of the NN. This will be discussed in more depth in section 2.2.

B

Derivation of CTRNN model for Different Low-Pass Locations

Here I will show the differences in two formulations used for CTRNNs. Neither is better than the other, they do however have different dynamics. I will demonstrate for a single input, a single recurrent connection, and no weights, but the general idea holds regardless.

The two options

A continuous time recurrent neuron can be modeled essentially the same as a discrete recurrent neuron which is filtered by a low pass filter. When modeling there is a choice as to the location of said low-pass filter. We can locate the filter before or after the nonlinear activation, as seen in Figure B.1. This alters the recurrence equation and hence the dynamics.

(a) Low pass before the nonlinear activation

(b) Low pass after the nonlinear activation

Figure B.1: Two different models of creating a continuous time recurrent neuron.

Difference in Recurrence Equation

It is well known that a low pass filter can be described by the following differential equation:

$$\tau \frac{dy(t)}{dt} = -y(t) = h(t) \tag{B.1}$$

where:

- $y(t)$ is the output signal,
- $h(t)$ is the input signal,
- τ is the time constant of the filter, which determines its cutoff frequency $f_c = \frac{1}{2\pi\tau}$.

As can be already observed from figure B.1 this will not result in equivalent equations in these two cases.

B.0.1. CASE 1: LOW PASS BEFORE ACTIVATION

In this case, $h(t)$ can be described as the sum of the input $u(t)$ and $y(t)$. Then using equation B.1 where we now take $x(t)$ as the output of the low pass filter following the notation in figure B.1a we get:

$$\begin{aligned} \tau \frac{dx(t)}{dt} &= -x(t) + h(t) \\ y(t) &= f(x(t)) \end{aligned} \tag{B.2}$$

where:

- $y(t)$ is the output signal of the neuron,
- $x(t)$ is the output of the low pass filter,
- $u(t)$ is the input to the neuron,
- $h(t) = u(t) + y(t)$ is the input signal,
- τ is the time constant of the filter.

B.0.2. CASE 2: LOW PASS BEFORE ACTIVATION

In the second case as seen in figure B.1b we now get that $h(t)$ is the nonlinear activation applied to the sum of $u(t)$ and $y(t)$. This results in, following the notation from the figure:

$$\begin{aligned} \tau \frac{dy(t)}{dt} &= -y(t) + h(t) \\ h(t) &= f(u(t) + y(t)) \end{aligned} \tag{B.3}$$

where:

- $y(t)$ is the output signal of the neuron and lowpass filter,
- $h(t) = f(u(t) + y(t))$ is the input to the low pass filter,
- $u(t)$ is the input signal to the neuron,
- τ is the time constant of the filter.

NUMERICAL DEMONSTRATION OF DIFFERENT DYNAMICS

Here I will demonstrate that these two formulations are not equivalent by implementing both a providing both models with the same ideal pulse:

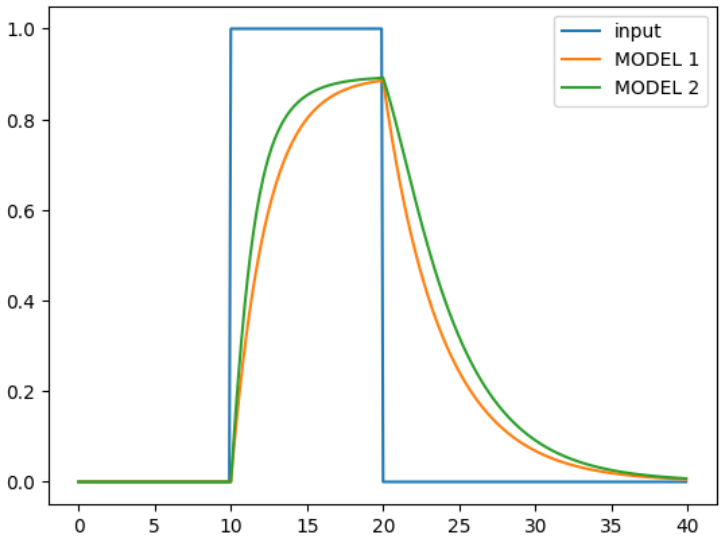


Figure B.2: Case 1 and Case 2 plotted for the same ideal pulse input.

While very simple it directly shows that these two are not equivalent. The full Colab Notebook showcases a bit more on the discretization and some different possibilities I examined.

As discussed in the main text for an opto-electronic-opto neuron the low pass filter should be located before the activation because the filtering can be done in the electrical domain. In reality, the summation and activation also have their own transfer functions, but since these are much faster than the electronics (at least in my report) these can be excluded.

C

Machine Learning and Hyperparameter Optimisation Details

Here I will go into more detail about how I have approached the training of the tasks, hyperparameter search, which hyperparameters I have optimized for, and some other details regarding machine learning.

Hyperparameter Optimization Using Optuna

I optimized the model's hyperparameters using Optuna. To do so first we define an objective function that trains each model with given hyperparameters and returns the validation metric. Since all the tasks are classification tasks I have chosen to optimize for validation accuracy. That is, during training, I use a training an a validation part of the dataset. The model is trained on the training set, and we use the intermediate models on the validation set to check if the model has not been over-fitted. The objective function returns the highest validation accuracy during training. To save some computational time I allowed for pruning to discard ineffective trials early. Depending on the specific task, I ran the optimization between 500-2000 trials, depending on the intermediate results. This process leads to sets of hyperparameters for each task. Alongside the best-performing models, Optuna also provides me with the ability to analyze which hyperparameters perform well. I have mainly used this to investigate the relation between the time constant and the accuracy. To a lesser degree, I have also looked at the relation between the hidden size, the weights, and the time constant.

General Hyperparameters

Here I will give the hyperparameters used during optimization and the rationale behind choosing these.

- **Batch Size**: Batch sizes were suggested categorically. The exact values are task-specific and were found somewhat empirically in an earlier stage.

- **Tau (τ) and Hidden Size**: The range for τ was suggested as an integer in the range of [2, 300]. Ideally, we would have liked for values lower than two, but this does not make sense for our discretization. For $\tau = 1$ the model becomes essentially a vanilla RNN. Lower time constants can only achieved in such a model by increasing the sampling rate. The hidden size was suggested categorically with [10, 25, 50, 75, 100]. This was done to allow the model to find the best possible model. In retrospect, it might have been more appropriate to select a given size since the size of the network and the importance of the time constant are related [47]

- **Activation Function** (`f_hidden`): The choice among several functions aimed to identify the most effective activation that promotes non-linearity. The function that where supplied are ["relu", "leaky relu", "sigmoid", "tanh"].
- **Optimizer**: Different optimizers were evaluated to determine the best fit for navigating the model's parameter space, focusing on achieving a robust and stable convergence. The following optimizers were suggested: ["Adam", "RMSprop", "SGD"].
- **Learning Rate and Adjustments** (`lr`, `lr_gamma`, `lr_step`): The dynamic tuning of learning rates and their adjustments is useful to adapt the training process to different stages of convergence, optimizing learning efficiency. The learning rate is suggested as a float on a logarithmic scale with lower and upper bounds [1e-8, 1e-1]; the learning rate gamma controls the decay of the learning rate during training. This was suggested as a float in the range of [0.1, 1.]; The learning rate step determines how frequently the decay of the learning rate is applied. This was suggested as an integer in a certain range [30, 100], which was determined somewhat empirically.
- **Regularization** (`L2_reg`) and **Weight Initialization Scale** (`W_scale`): These were finely adjusted within their bounds to prevent overfitting and ensure the model's adaptability, contributing to its robustness and generalization capacity. The *L2_reg* tries to ensure smaller weights by penalizing larger weights. This was suggested as a float in the range [1.e-12, 1.e-1] logarithmically. The extension to such small numbers was to ensure the optimization could essentially turn it off without adapting the code too much. The weights scale is determined as a float in the range of [0.01, 0.9]. This scaling was applied via a custom weight initialization where I create a random torch tensor, shift it to be centered around zero, and then scale the weights.

The complete specifics of all the suggestions and configuration and the rest of the code I can provide if desired.

D

A Physically Plausible On-Chip Opto-Electronic Neuron Model

The CTRNN can be built from on-chip photonic components which is discussed in this section. When building a model, several choices and assumptions must be made.

As discussed in chapter 2.1, photonic neural networks can be divided into two sub-classes: all-optical (AO), and opto-electronic-opto (OEO). Control over the time constant in the optical domain is difficult to the degree we require in this report. In the electrical domain, we have this control. Further, it has been shown experimentally that OEO architectures are cascadable and can be used for recurrent architectures [11]. Therefore an OEO architecture is chosen.

A photonic neuron (most neuron models really) can be divided into several stages: weighting, summing, and nonlinear activation. For each of these stages, I will be choosing a certain method, such that we can construct our photonic CTRNN model.

Weighting can be done in several ways which determine what range of weights the PNN can use. If one assumes to do the weighting via simple optical attenuators, such as in [3], the weight range will be restricted to [0, 1]. This is due to the attenuator only being able to attenuate the optical power to zero power. Another possibility is using microring resonators in combination with push-pull photodetectors (which is the summing stage) which then allows for negative weights giving a range of [-1, 1] [10], [27]. Such a weight filter bank is more complex, and also poses more complex control problems, but allows for a greater range of weights. Both options will be investigated.

OEO-PNNs all use essentially the same method, with some nuances, of summing the optical power via the linear response of a photodiode. For simplicity, in this report, it will be assumed that a single simple model of a photodiode is used. Alternatively, one would use a single photodiode for each waveguide.

The relatively weak photocurrent is amplified via a transimpedance amplifier (TIA) which converts the current into a voltage. The TIA is modeled as an operational amplifier (OpAmp) with an RC circuit in the feedback loop. Other methods of converting the current to a voltage are also possible [57]. This RC circuit will also provide the low pass filtering, and hence the time constant of the neuron.

The nonlinear activation is performed via a micro ring modulator which is modulated by the voltage from the TIA. This is the most common method for OEO-PNNs, but other methods are possible using for example current or thermal modulation [22], or other modulators such as Mach-Zehnder modulators [13].

This result in the following simplified model of an OEO-PNN:

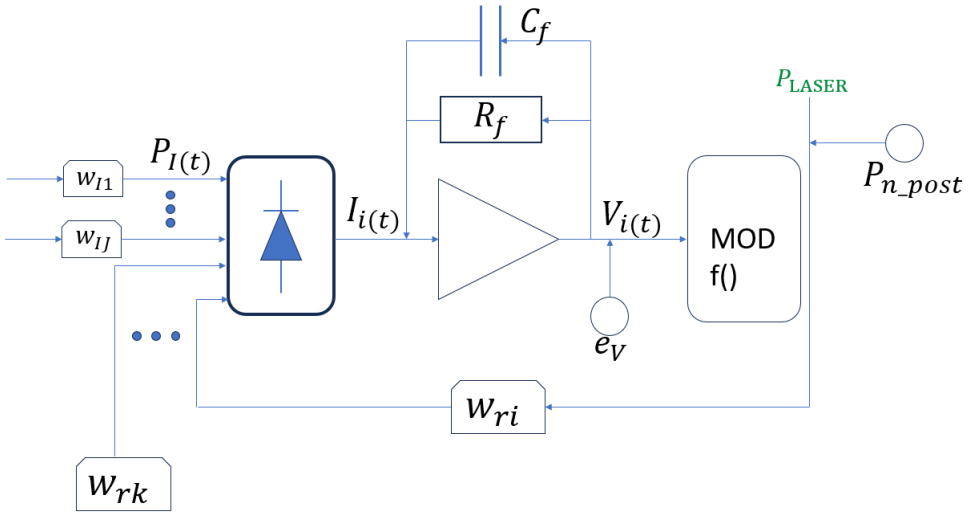


Figure D.1: Model of an optoelectronic photonic neuron. Input laser power is summed via a photodiode. The photocurrent is converted into a voltage and low pass filtered via a transimpedance amplifier. The voltage modulates a laser at the end of the neuron which is used as the output, and the recurrent input.

With, including various noise sources expressed as a voltage, results in the following equation :

$$R_f C_f \frac{dV_i(t)}{dt} = -V_i(t) + R_f R_d \sum_j w_{Ij} P_j(t) + R_f R d \sum_k w_{rk} P_{out,k}(t) + V_{bias} + e_V$$
$$P_{out,i}(t) = P_{laser} f(V_i(t)) + P_{npost} \tag{D.1}$$

Where $P_I(t)$ is the optical input power, R_f and C_f are the feedback resistance and capacitance which determine the time constant, $V(t)$ the voltage after amplification from the OpAmp which serves as the state of the neuron, e_v the noise from all sources written in terms of voltage, and P_{n-post} a noise contribution on the laser power.
When using the physical model we have additional constraints and assumptions we must consider before we can implement it numerically.

Firstly, we must make some assumptions on how the fan-out of each neuron is handled. Suppose we have a single neuron that must supply power to N other neurons (this can include recurrent and feedforward connections). In the non-physical model, we could supply each neuron directly with the output, but in the physical neuron, the waveguide must split for each neuron, losing power proportional to the fan-out [27], [57]. Therefore, we either must take care of this with a gain stage, in the supply laser, or with the gain of the OpAmp. To not overcomplicate the analysis, I will assume that there is some gain stage present such that $P_{out,i}$, and P_{Ij} have an equal range of power P_{laser}.

Secondly, as mentioned previously, the weights are in the range of [0, 1] due to the optical attenuation. This constraint can be alleviated by using the Broadcast and Weight scheme where the use of MRM as weights allows for a range of [-1, 1.], as discussed in 2.1.1.

Thirdly, the nonlinear function cannot be negative due to it being a modulation of optical power. This excludes functions such as the hyperbolic tangent or the leaky rectified linear unit. Other nonlinear functions are possible optically on-chip and I will allow these to be used.

Lastly, P_{out} cannot be negative, and the optical power also cannot exceed the input laser power P_{laser}. In this, we are assuming that each neuron has a supply laser and that this value is kept constant regardless of the fan-out.

It should be noted that while the optical power cannot be negative, or exceed unity, this is not the case for the state voltage $V(t)$. Further, a voltage bias V_b allows for shifting the center of the voltage operating point, and the gain resistor R_f can amplify the voltage up to the supply voltage of the OpAmp.

We can again discretize the equation to obtain the following equation:

$$dV_i(t+\Delta t) = (1-\frac{\Delta t}{R_f C_f})V_i(t) + \frac{\Delta t}{R_f C_f}(R_f R_d P_{laser}\sum_j w_{Ij}u_j(t) + R_f R_d P_{laser}\sum_k w_{rk}y_{out,k}(t) + V_{bias} + e_V)$$
$$y_{out,i}(t+\Delta t) = f(V_i(t+\Delta t)) + e_{npost} \tag{D.2}$$

Where now the input $u(t)$, output $y(t)$, and the weights w_r, w_i are in the range of [0, 1] (weight constraints can be relaxed to [-1, 1]).

D.0.1. Shortcomings and Recommendations

There are many more nuances to designing an on-chip PNN. We can think of how to route the waveguides exactly, how the summation is achieved exactly, how the electronic and photonic designs are integrated during fabrication, etc. This differs for different architectures and this was somewhat touched upon in the literature review. However, to limit the scope. I have not gone into extreme details and potential difficulties which come with making some choices. I do this at the expense of losing some realism, but I will leave that for future research or when an actual attempt would be made to realize this network. In my opinion a more realistic model should be made in a non-sequential hardware programming language such as for example Verilog or VHDL. This has been proposed before [75]

/

BIBLIOGRAPHY

[1] F. Zhou and Y. Chai, "Near-sensor and in-sensor computing," en, *Nature Electronics*, vol. 3, no. 11, pp. 664–671, Nov. 2020, Number: 11 Publisher: Nature Publishing Group, ISSN: 2520-1131. DOI: 10.1038/s41928-020-00501-9. [Online]. Available: https://www.nature.com/articles/s41928-020-00501-9 (visited on 10/09/2023).

[2] S. Yuan, C. Ma, E. Fetaya, *et al.*, "Geometric deep optical sensing," *Science*, vol. 379, no. 6637, eade1220, Mar. 2023, Publisher: American Association for the Advancement of Science. DOI: 10.1126/science.ade1220. [Online]. Available: https://www.science.org/doi/10.1126/science.ade1220 (visited on 03/26/2023).

[3] F. Ashtiani, A. J. Geers, and F. Aflatouni, "An on-chip photonic deep neural network for image classification," en, *Nature*, vol. 606, no. 7914, pp. 501–506, Jun. 2022, Number: 7914 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/s41586-022-04714-0. [Online]. Available: https://www.nature.com/articles/s41586-022-04714-0 (visited on 01/26/2023).

[4] G. Mourgias-Alexandris, A. Tsakyridis, N. Passalis, A. Tefas, K. Vyrsokinos, and N. Pleros, "An all-optical neuron with sigmoid activation function," en, *Optics Express*, vol. 27, no. 7, p. 9620, Apr. 2019, ISSN: 1094-4087. DOI: 10.1364/OE.27.009620. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=oe-27-7-9620 (visited on 03/09/2023).

[5] G. Mourgias-Alexandris, G. Dabos, N. Passalis, A. Totović, A. Tefas, and N. Pleros, "All-Optical WDM Recurrent Neural Networks With Gating," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 5, pp. 1–7, Sep. 2020, Conference Name: IEEE Journal of Selected Topics in Quantum Electronics, ISSN: 1558-4542. DOI: 10.1109/JSTQE.2020.2995830.

[6] G. Mourgias-Alexandris, N. Passalis, G. Dabos, A. Totovic, A. Tefas, and N. Pleros, "A Photonic Recurrent Neuron for Time-Series Classification," en, *Journal of Lightwave Technology*, vol. 39, no. 5, pp. 1340–1347, Mar. 2021, ISSN: 0733-8724, 1558-2213. DOI: 10.1109/JLT.2020.3038890. [Online]. Available: https://ieeexplore.ieee.org/document/9263298/ (visited on 01/26/2023).

[7] B. Shi, N. Calabretta, and R. Stabile, "Deep Neural Network Through an InP SOA-Based Photonic Integrated Cross-Connect," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 1, pp. 1–11, Jan. 2020, Conference Name: IEEE Journal of Selected Topics in Quantum Electronics, ISSN: 1558-4542. DOI: 10.1109/JSTQE.2019.2945548.

[8] B. Shi, N. Calabretta, and R. Stabile, "InP photonic integrated multi-layer neural networks: Architecture and performance analysis," en, *APL Photonics*, vol. 7, no. 1, p. 010 801, Jan. 2022, ISSN: 2378-0967. DOI: 10.1063/5.0066350. [Online]. Available: https://aip.scitation.org/doi/10.1063/5.0066350 (visited on 03/13/2023).

[9] B. Shi, N. Calabretta, and R. Stabile, "Parallel Photonic Convolutional Processing on-Chip With Cross-Connect Architecture and Cyclic AWGs," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 29, no. 2: Optical Computing, pp. 1–10, Mar. 2023, Conference Name: IEEE Journal of Selected Topics in Quantum Electronics, ISSN: 1558-4542. DOI: 10.1109/JSTQE.2022.3226138.

[10] A. N. Tait, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal, "Broadcast and Weight: An Integrated Network For Scalable Photonic Spike Processing," *Journal of Lightwave Technology*, vol. 32, no. 21, pp. 4029–4041, Nov. 2014, Conference Name: Journal of Lightwave Technology, ISSN: 1558-2213. DOI: 10.1109/JLT.2014.2345652.

[11] A. N. Tait, T. F. De Lima, E. Zhou, *et al.*, "Neuromorphic photonic networks using silicon photonic weight banks," en, *Scientific Reports*, vol. 7, no. 1, p. 7430, Aug. 2017, ISSN: 2045-2322. DOI: 10.1038/s41598-017-07754-z. [Online]. Available: https://www.nature.com/articles/s41598-017-07754-z (visited on 05/09/2023).

[12] J. Feldmann, N. Youngblood, M. Karpov, *et al.*, "Parallel convolutional processing using an integrated photonic tensor core," en, *Nature*, vol. 589, no. 7840, pp. 52–58, Jan. 2021, Number: 7840 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/s41586-020-03070-1. [Online]. Available: https://www.nature.com/articles/s41586-020-03070-1 (visited on 03/31/2023).

[13] T. F. de Lima, C. Huang, S. Bilodeau, *et al.*, "Real-Time Operation of Silicon Photonic Neurons," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, Mar. 2020, pp. 1–3.

[14] B. Shi, N. Calabretta, and R. Stabile, "Emulation and modelling of semiconductor optical amplifier-based all-optical photonic integrated deep neural network with arbitrary depth," *Neuromorphic Computing and Engineering*, vol. 2, no. 3, Sep. 2022, ISSN: 2634-4386. DOI: 10.1088/2634-4386/ac8827.

[15] Y. Bai, X. Xu, M. Tan, *et al.*, "Photonic multiplexing techniques for neuromorphic computing," en, *Nanophotonics*, Jan. 2023, Publisher: De Gruyter, ISSN: 2192-8614. DOI: 10.1515/nanoph-2022-0485. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/nanoph-2022-0485/html (visited on 02/14/2023).

[16] Y. Shen, N. C. Harris, S. Skirlo, *et al.*, "Deep learning with coherent nanophotonic circuits," en, *Nature Photonics*, vol. 11, no. 7, pp. 441–446, Jul. 2017, Number: 7 Publisher: Nature Publishing Group, ISSN: 1749-4893. DOI: 10.1038/nphoton.2017.93. [Online]. Available: https://www.nature.com/articles/nphoton.2017.93 (visited on 01/26/2023).

[17] J. Cheng, H. Zhou, and J. Dong, "Photonic Matrix Computing: From Fundamentals to Applications," en, *Nanomaterials*, vol. 11, no. 7, p. 1683, Jul. 2021, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-4991. DOI: 10.3390/nano11071683. [Online]. Available: https://www.mdpi.com/2079-4991/11/7/1683 (visited on 03/13/2023).

[18] H. Zhou, J. Dong, J. Cheng, *et al.*, "Photonic matrix multiplication lights up photonic accelerator and beyond," en, *Light: Science & Applications*, vol. 11, no. 1, p. 30, Feb. 2022, Number: 1 Publisher: Nature Publishing Group, ISSN: 2047-7538. DOI: 10.1038/s41377-022-00717-8. [Online]. Available: https://www.nature.com/articles/s41377-022-00717-8 (visited on 03/31/2023).

[19] J. Wu, X. Lin, Y. Guo, *et al.*, "Analog Optical Computing for Artificial Intelligence," en, *Engineering*, vol. 10, pp. 133–145, Mar. 2022, ISSN: 2095-8099. DOI: 10.1016/j.eng.2021.06.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809921003349 (visited on 02/13/2023).

[20] B. J. Shastri, C. Huang, A. N. Tait, T. F. d. Lima, and P. R. Prucnal, "Silicon photonic neural network applications and prospects," in *AI and Optical Data Sciences III*, vol. 12019, SPIE, Mar. 2022, pp. 135–144. DOI: 10.1117/12.2614865. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12019/120190K/Silicon-photonic-neural-network-applications-and-prospects/10.1117/12.2614865.full (visited on 02/13/2023).

[21] K. Liao, T. Dai, Q. Yan, X. Hu, and Q. Gong, "Integrated Photonic Neural Networks: Opportunities and Challenges," *ACS Photonics*, Feb. 2023, Publisher: American Chemical Society. DOI: 10.1021/acsphotonics.2c01516. [Online]. Available: https://doi.org/10.1021/acsphotonics.2c01516 (visited on 02/14/2023).

[22] A. N. Tait, T. Ferreira de Lima, M. A. Nahmias, *et al.*, "Silicon Photonic Modulator Neuron," *Physical Review Applied*, vol. 11, no. 6, p. 064 043, Jun. 2019, Publisher: American Physical Society. DOI: 10.1103/PhysRevApplied.11.064043. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.11.064043 (visited on 02/22/2023).

[23] T. Zhou, X. Lin, J. Wu, *et al.*, "Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit," en, *Nature Photonics*, vol. 15, no. 5, pp. 367–373, May 2021, Number: 5 Publisher: Nature Publishing Group, ISSN: 1749-4893. DOI: 10.1038/s41566-021-00796-w. [Online]. Available: https://www.nature.com/articles/s41566-021-00796-w (visited on 03/27/2023).

[24] R. D. Beer, "On the Dynamics of Small Continuous-Time Recurrent Neural Networks," en, *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, Mar. 1995, ISSN: 1059-7123, 1741-2633. DOI: 10.1177/105971239500300405. [Online]. Available: http://journals.sagepub.com/doi/10.1177/105971239500300405 (visited on 04/24/2023).

[25] K.-i. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," en, *Neural Networks*, vol. 6, no. 6, pp. 801–806, Jan. 1993, ISSN: 0893-6080. DOI: 10.1016/S0893-6080(05)80125-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089360800580125X (visited on 03/21/2023).

[26] H.-T. Peng, T. F. de Lima, E. C. Blow, *et al.*, "Time Series Prediction and Classification using Silicon Photonic Neuron with a Self-Connection," in *2022 Conference on Lasers and Electro-Optics (CLEO)*, ISSN: 2160-8989, May 2022, pp. 1–2.

[27] T. F. de Lima, H.-T. Peng, A. N. Tait, *et al.*, "Machine Learning With Neuromorphic Photonics," en, *Journal of Lightwave Technology*, vol. 37, no. 5, pp. 1515–1534, Mar. 2019, ISSN: 0733-8724, 1558-2213. DOI: 10.1109/JLT.2019.2903474. [Online]. Available: https://ieeexplore.ieee.org/document/8662590/ (visited on 02/13/2023).

[28] H.-T. Peng, J. Lederman, L. Xu, *et al.*, *A Photonic-Circuits-Inspired Compact Network: Toward Real-Time Wireless Signal Classification at the Edge*, arXiv:2106.13865 [cs, eess], Jun. 2021. [Online]. Available: http://arxiv.org/abs/2106.13865 (visited on 02/20/2023).

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[30] A. Madsen, "Visualizing memorization in RNNs," en, *Distill*, vol. 4, no. 3, e16, Mar. 2019, ISSN: 2476-0757. DOI: 10.23915/distill.00016. [Online]. Available: https://distill.pub/2019/memorization-in-rnns (visited on 03/28/2023).

[31] M. H Hasan, A. Abbasalipour, H. Nikfarjam, *et al.*, "Exploiting Pull-In/Pull-Out Hysteresis in Electrostatic MEMS Sensor Networks to Realize a Novel Sensing Continuous-Time Recurrent Neural Network," en, *Micromachines*, vol. 12, no. 3, p. 268, Mar. 2021, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2072-666X. DOI: 10.3390/mi12030268. [Online]. Available: https://www.mdpi.com/2072-666X/12/3/268 (visited on 04/24/2023).

[32] Z. Yu, D. S. Moirangthem, and M. Lee, "Continuous Timescale Long-Short Term Memory Neural Network for Human Intent Understanding," *Frontiers in Neurorobotics*, vol. 11, p. 42, Aug. 2017, ISSN: 1662-5218. DOI: 10.3389/fnbot.2017.00042. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5572368/ (visited on 04/24/2023).

[33] M. Emad-Ud-Din, M. H. Hasan, R. Jafari, S. Pourkamali, and F. Alsaleem, "Simulation for a Mems-Based CTRNN Ultra-Low Power Implementation of Human Activity Recognition," *Frontiers in Digital Health*, vol. 3, 2021, ISSN: 2673-253X. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fdgth.2021.731076 (visited on 04/24/2023).

[34] T. Matsuki and K. Shibata, "Learning Time Constant of Continuous-Time Neurons with Gradient Descent," en, in *RITA 2018*, A. P. P. Abdul Majeed, J. A. Mat-Jizat, M. H. A. Hassan, Z. Taha, H. L. Choi, and J. Kim, Eds., ser. Lecture Notes in Mechanical Engineering, Singapore: Springer, 2020, pp. 149–159, ISBN: 9789811383236. DOI: 10.1007/978-981-13-8323-6_13.

[35] Y. Yamashita and J. Tani, "Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment," en, *PLOS Computational Biology*, vol. 4, no. 11, e1000220, Nov. 2008, Publisher: Public Library of Science, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000220. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000220 (visited on 05/09/2023).

[36] T. Kurikawa and K. Kaneko, "Multiple-Timescale Neural Networks: Generation of History-Dependent Sequences and Inference Through Autonomous Bifurcations," *Frontiers in Computational Neuroscience*, vol. 15, 2021, ISSN: 1662-5188. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2021.743537 (visited on 05/01/2023).

D

[37] C. Angelo, "Multiple Time Scales Recurrent Neural Network for Complex Action Acquisition," en, *Frontiers in Computational Neuroscience*, vol. 5, 2011, ISSN: 1662-5188. DOI: 10.3389/conf.fncom.2011.52.00009. [Online]. Available: http://www.frontiersin.org/10.3389/conf.fncom.2011.52.00009/event_abstract (visited on 05/01/2023).

[38] S. Heinrich, S. Magg, and S. Wermter, "Analysing the Multiple Timescale Recurrent Neural Network for Embodied Language Understanding," en, in *Artificial Neural Networks*, P. Koprinkova-Hristova, V. Mladenov, and N. K. Kasabov, Eds., ser. Springer Series in Bio-/Neuroinformatics, Cham: Springer International Publishing, 2015, pp. 149–174, ISBN: 978-3-319-09903-3. DOI: 10.1007/978-3-319-09903-3_8.

[39] A. N. Tait, E. Zhou, A. X. Wu, *et al.*, "Demonstration of a silicon photonic neural network," en, in *2016 IEEE Photonics Society Summer Topical Meeting Series (SUM)*, Newport Beach, CA, USA: IEEE, Jul. 2016, pp. 72–73, ISBN: 978-1-5090-1900-7. DOI: 10.1109/PHOSST.2016.7548726. [Online]. Available: http://ieeexplore.ieee.org/document/7548726/ (visited on 03/21/2023).

[40] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, *et al.*, "Experimental demonstration of reservoir computing on a silicon photonics chip," en, *Nature Communications*, vol. 5, no. 1, p. 3541, Mar. 2014, Number: 1 Publisher: Nature Publishing Group, ISSN: 2041-1723. DOI: 10.1038/ncomms4541. [Online]. Available: https://www.nature.com/articles/ncomms4541 (visited on 01/31/2023).

[41] H. Jaeger, D. Doorakkers, C. Lawrence, and G. Indiveri, *Dimensions of Timescales in Neuromorphic Computing Systems*, arXiv:2102.10648 [cs], Feb. 2021. [Online]. Available: http://arxiv.org/abs/2102.10648 (visited on 05/01/2023).

[42] H. Jaeger and F. Catthoor, *Timescales: The choreography of classical and unconventional computing*, arXiv:2301.00893 [cs], Jan. 2023. [Online]. Available: http://arxiv.org/abs/2301.00893 (visited on 05/01/2023).

[43] E. R. Howard, B. A. Marquez, and B. J. Shastri, "Photonic Long-Short Term Memory Neural Networks with Analog Memory," in *2020 IEEE Photonics Conference (IPC)*, ISSN: 2575-274X, Sep. 2020, pp. 1–2. DOI: 10.1109/IPC47351.2020.9252216.

[44] F. Duport, A. Smerieri, A. Akrout, M. Haelterman, and S. Massar, "Fully analogue photonic reservoir computer," en, *Scientific Reports*, vol. 6, no. 1, p. 22 381, Mar. 2016, Number: 1 Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/srep22381. [Online]. Available: https://www.nature.com/articles/srep22381 (visited on 02/02/2023).

[45] T. Hülser, F. Köster, L. Jaurigue, and K. Lüdge, "Role of delay-times in delay-based photonic reservoir computing [Invited]," en, *Optical Materials Express*, vol. 12, no. 3, p. 1214, Mar. 2022, ISSN: 2159-3930. DOI: 10.1364/OME.451016. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=ome-12-3-1214 (visited on 02/01/2023).

[46] H. Jaeger, "Long Short-Term Memory in Echo State Networks: Details of a Sim- ulation Study," en,

[47] D. Verstraeten, "Reservoir Computing: Computation with dynamical systems," eng, ISBN: 9789085783091, dissertation, Ghent University, 2009. [Online]. Available: http://hdl.handle.net/1854/LU-779431 (visited on 01/17/2024).

[48] D. Sussillo and O. Barak, "Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks," en, *Neural Computation*, vol. 25, no. 3, pp. 626–649, Mar. 2013, ISSN: 0899-7667, 1530-888X. DOI: 10.1162/NECO_a_00409. [Online]. Available: https://direct.mit.edu/neco/article/25/3/626-649/7854 (visited on 09/02/2023).

[49] O. Barak, "Recurrent neural networks as versatile tools of neuroscience research," en, *Current Opinion in Neurobiology*, vol. 46, pp. 1–6, Oct. 2017, ISSN: 09594388. DOI: 10.1016/j.conb.2017.06.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0959438817300429 (visited on 09/02/2023).

[50] G. Bailador, D. Roggen, G. Tröster, and G. Triviño, "Real time gesture recognition using continuous time recurrent neural networks," en, in *Proceedings of the Second International Conference on Body Area Networks BodyNets*, Florence, Italy: ICST, 2007, ISBN: 978-963-06-2193-9. DOI: 10.4108/bodynets.2007.149. [Online]. Available: http://eudl.eu/doi/10.4108/bodynets.2007.149 (visited on 05/25/2023).

[51] S. Heinrich, T. Alpay, and S. Wermter, "Adaptive and Variational Continuous Time Recurrent Neural Networks," en, in *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, Tokyo, Japan: IEEE, Sep. 2018, pp. 13–18, ISBN: 978-1-5386-6110-9. DOI: 10.1109/DEVLRN.2018.8761019. [Online]. Available: https://ieeexplore.ieee.org/document/8761019/ (visited on 04/24/2023).

[52] H. Jaeger, "A tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach," en,

[53] H. Jaeger, *Lecture Notes Neural Network*, Apr. 2023. [Online]. Available: https://www.ai.rug.nl/minds/uploads/LN_NN_RUG.pdf.

[54] L. Manneschi, M. O. A. Ellis, G. Gigante, A. C. Lin, P. Del Giudice, and E. Vasilaki, "Exploiting Multiple Timescales in Hierarchical Echo State Networks," *Frontiers in Applied Mathematics and Statistics*, vol. 6, 2021, ISSN: 2297-4687. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fams.2020.616658 (visited on 12/22/2023).

[55] F. Wyffels, B. Schrauwen, D. Verstraeten, and D. Stroobandt, "Band-pass Reservoir Computing," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, ISSN: 2161-4407, Jun. 2008, pp. 3204–3209. DOI: 10.1109/IJCNN.2008.4634252. [Online]. Available: https://ieeexplore.ieee.org/document/4634252 (visited on 03/05/2024).

[56] K. Vandoorne, M. Fiers, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, "Photonic reservoir computing: A new approach to optical information processing," in *2010 12th International Conference on Transparent Optical Networks*, ISSN: 2162-7339, Jun. 2010, pp. 1–4. DOI: 10.1109/ICTON.2010.5548990. [Online]. Available: https://ieeexplore.ieee.org/document/5548990 (visited on 01/17/2024).

[57] T. F. De Lima, A. N. Tait, H. Saeidi, *et al.*, "Noise Analysis of Photonic Modulator Neurons," en, *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 1, pp. 1–9, Jan. 2020, ISSN: 1077-260X, 1558-4542. DOI: 10.1109/JSTQE.2019.2931252. [Online]. Available: https://ieeexplore.ieee.org/document/8782580/ (visited on 11/10/2023).

[58] G. Mourgias-Alexandris, M. Moralis-Pegios, A. Tsakyridis, *et al.*, "Noise-resilient and high-speed deep learning with coherent silicon photonics," en, *Nature Communications*, vol. 13, no. 1, p. 5572, Sep. 2022, Number: 1 Publisher: Nature Publishing Group, ISSN: 2041-1723. DOI: 10.1038/s41467-022-33259-z. [Online]. Available: https://www.nature.com/articles/s41467-022-33259-z (visited on 03/24/2023).

[59] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982, Publisher: Proceedings of the National Academy of Sciences. DOI: 10.1073/pnas.79.8.2554. [Online]. Available: https://www-pnas-org.tudelft.idm.oclc.org/doi/10.1073/pnas.79.8.2554 (visited on 05/13/2024).

[60] D. Krotov and J. Hopfield, *Large Associative Memory Problem in Neurobiology and Machine Learning*, en, arXiv:2008.06996 [cond-mat, q-bio, stat], Apr. 2021. [Online]. Available: http://arxiv.org/abs/2008.06996 (visited on 05/13/2024).

[61] Suvanjanprasai, *English: The image contains handwritten numbers from 0 to 9. The size of all the small images is 28*28 pixels. The image can be used to make a number detection program using machine learning or neural network.* May 2023. [Online]. Available: https://commons.wikimedia.org/wiki/File:MnistExamplesModified.png (visited on 02/25/2024).

D

[62] S. Bai, J. Z. Kolter, and V. Koltun, *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*, arXiv:1803.01271 [cs], Apr. 2018. [Online]. Available: http://arxiv.org/abs/1803.01271 (visited on 08/16/2023).

[63] T. W. Hughes, I. A. D. Williamson, M. Minkov, and S. Fan, "Wave physics as an analog recurrent neural network," *Science Advances*, vol. 5, no. 12, eaay6946, Dec. 2019, Publisher: American Association for the Advancement of Science. DOI: 10.1126/sciadv.aay6946. [Online]. Available: https://www-science-org.tudelft.idm.oclc.org/doi/10.1126/sciadv.aay6946 (visited on 11/14/2023).

[64] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*, arXiv:1907.10902 [cs, stat], Jul. 2019. DOI: 10.48550/arXiv.1907.10902. [Online]. Available: http://arxiv.org/abs/1907.10902 (visited on 02/25/2024).

[65] D. H. P. C. C. (DHPC), *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[66] C. Gonzalez, H. Asadi, L. Kooijman, and C. P. Lim, *Neural Networks for Fast Optimisation in Model Predictive Control: A Review*, en, arXiv:2309.02668 [cs, eess, math], Dec. 2023. [Online]. Available: http://arxiv.org/abs/2309.02668 (visited on 05/03/2024).

[67] S.-G. Hong, S.-W. Kim, and J.-J. Lee, "The minimum cost path finding algorithm using a Hopfield type neural network," in *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.*, vol. 4, Mar. 1995, 1719–1726 vol.4. DOI: 10.1109/FUZZY.1995.409914. [Online]. Available: https://ieeexplore.ieee.org/document/409914 (visited on 05/03/2024).

[68] H. Ramsauer, B. Schäfl, J. Lehner, *et al.*, *Hopfield Networks is All You Need*, arXiv:2008.02217 [cs, stat], Apr. 2021. DOI: 10.48550/arXiv.2008.02217. [Online]. Available: http://arxiv.org/abs/2008.02217 (visited on 04/30/2024).

[69] Y. Liu and W. Xu, "Application Of Improved Hopfield Neural Network In Path Planning," en, *Journal of Physics: Conference Series*, vol. 1544, no. 1, p. 012 154, May 2020, ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1544/1/012154. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1544/1/012154 (visited on 05/06/2024).

[70] G. Hinton, *The Forward-Forward Algorithm: Some Preliminary Investigations*, arXiv:2212.13345 [cs], Dec. 2022. DOI: 10.48550/arXiv.2212.13345. [Online]. Available: http://arxiv.org/abs/2212.13345 (visited on 04/15/2024).

[71] Jaeger, *LN_nn_rug.pdf*, Apr. 2023. [Online]. Available: https://www.ai.rug.nl/minds/uploads/LN_NN_RUG.pdf (visited on 05/01/2023).

[72] F. Martins, E. Pereira, and R. Guarnieri, "Solar Radiation Forecast Using Artificial Neural Networks," *International Journal of Nuclear Energy Science and Technology*, vol. 2, Dec. 2012.

[73] A. Tsantekidis, N. Passalis, and A. Tefas, "Chapter 5 - Recurrent neural networks," en, in *Deep Learning for Robot Perception and Cognition*, A. Iosifidis and A. Tefas, Eds., Academic Press, Jan. 2022, pp. 101–115, ISBN: 978-0-323-85787-1. DOI: 10.1016/B978-0-32-385787-1.00010-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323857871000105 (visited on 04/07/2023).

[74] C. Tallec and Y. Ollivier, *Unbiasing Truncated Backpropagation Through Time*, en, arXiv:1705.08209 [cs], May 2017. [Online]. Available: http://arxiv.org/abs/1705.08209 (visited on 03/11/2024).

[75] J. Singh, H. Morison, Z. Guo, *et al.*, "Neuromorphic photonic circuit modeling in Verilog-A," *APL Photonics*, vol. 7, p. 046 103, Apr. 2022. DOI: 10.1063/5.0079984.