
DEGREE PROJECT IN TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2023*

# Template

## KTH Thesis Report

Max Schaufelberger

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

## Author

Max Schaufelberger <maxscha@kth.se>
School of Engineering Sciences
KTH Royal Institute of Technology

## Place for Project

Stockholm, Sweden
Ottignies-Louvain-la-Neuve, Belgium

## Examiner

Prof. Olof Runborg
Department of Numerical Analysis
KTH Royal Institute of Technology
Stockholm, Sweden

## Supervisor

Prof. Elias Jarlebring
Department of Numerical Analysis
KTH Royal Institute of Technology
Stockholm, Sweden

## Supervisor

Arvind Kumar
Division of Computational Science and Technology
KTH Royal Institute of Technology
Stockholm, Sweden

## Supervisor

Frédéric Crevecoeur
Institute of Information and Communication Technologies, Electronics and Applied Mathematics
UCLouvain Catholic University of Louvain
Louvain-la-Neuve, Belgium

# Abstract

The report needs to be compiled using XeLaTeX as different fonts are needed for the project to look like the original report. You might have to change this manually in overleaf.

This template was created by Hannes Rabo <hannes.rabo@gmail.com or hrabo@kth.se> from the template provided by KTH. You can send me an email if you need help in making it work for you.

Write an abstract. Introduce the subject area for the project and describe the problems that are solved and described in the thesis. Present how the problems have been solved, methods used and present results for the project. Use probably one sentence for each chapter in the final report.

The presentation of the results should be the main part of the abstract. Use about ½ A4-page. English abstract

Abstract:

In the end, this approach depended on hand-tuning hyper-parameters to set the controller to give usable results which was in opposition to our goals.

## Keywords

Template, Thesis, Keywords ...

# Abstract

Svenskt abstract Svensk version av abstract – samma titel på svenska som på engelska.

Skriv samma abstract på svenska. Introducera ämnet för projektet och beskriv problemen som löses i materialet. Presentera

## Nyckelord

Kandidat examensarbete, ...

# Acknowledgements

Write a short acknowledgements. Don't forget to give some credit to the examiner and supervisor.

# Acronyms

| | |
|---|---|
| **AD** | Automatic Differentiation |
| **ML** | Machine Learning |
| **RL** | Reinforcement Learning |
| **BP** | Back-Propagation |
| **LTI** | Linear Time Invariant |
| **RNN** | Recurrent Neural Network |
| **NN** | Neural Network |
| **GD** | Gradient Descent |
| **EWC** | Elastic Weight Consolidation |
| **ANN** | Artificial Neural Network |
| **SNN** | Spiking neural network |
| **GPU** | Graphics Processing Unit |
| **BPTT** | Backpropagation Through Time |
| **RTRL** | Real Time Recurrent Learning |
| **SRDP** | Spike Rate Dependent Plasticity |
| **STDP** | Spike Time Dependent Plasticity |
| **LSM** | Liquid State Machine |
| **NEF** | Neural Engineering Framework |
| **SOP** | Synaptic Operation |
| **IF** | Integrate and Fire |
| **LIF** | Leaky-integrate-and-fire |
| **TTFS** | Time to First Spike |
| **ODE** | ordinary differential equation |
| **LHS** | Left Hand Side |
| **RHS** | Right Hand Side |
| **HH** | Hodgkin–Huxley |

**NLP** Natural Language Processing

# Contents

# Chapter 1

# Introduction

The human brain is a brilliant computing unit comprised of around 86 billion[7] neurons. Each of these neurons can have thousands of connections to other neurons. Between these connections, information travels trough the network as electrical impulses that interact with the neurons own electrical potential. With this network, the human brain is capable of performing vastly different and complex tasks. Machines and robots beat humans in raw computing power by several orders of magnitude, yet many tasks are next to impossible to solve by machines and classical algorithms alone. Moreover, many machine implementations lack the speed, precision or flexibility of the human counterpart.

Researchers tried to combat this by mimicking the brain's internal network structure to solve problems deemed unsuitable for classic algorithms.

Artificial Neural Networks (ANNs) have shown a great success in previously hard to solve problems.

However the classical ANNs still struggle in context of control. But where the highly abstract ANNs reach there limits a more biologically plausible network can overcome this obstacle. SNNs compute and simulate spiking rates or even extend to calculating individual spikes to mimic natural neural networks. With this leap in complexity also comes the advantage of enhanced computing power. Furthermore, with newer more biologically inspired networks we are able to solve a broader range of problems. Using SNNs, we set out to design such a network in order to control a linear system.

## 1.1 Background

The most common neural network architecture for ANNs are the feed-forward networks. In these networks, information travels only in one direction and is not propagated by spikes but gradients of activation usually set in $[0, 1]$ or $[-1, 1]$. These ANNs have made impressive progress in the fields of image recognition, autonomic driving, medical diagnosis[63] or Natural Language Processing (NLP) (using Transformers[79]).
This abstract representation brings advantages e.g in modelling and implementation but also gives away some key features of the human brain. Due to the information travelling only towards the output, feed-forward networks cannot build a memory or easily process temporal data. Recurrent models exist which allow for memory [31] and sequential data input but loose some of the advantages compared to the Feed-Forward due to its increased complexity.
A third generation[54] of network architectures has risen, which aims to be even more biologically plausible. Inspired from nature, they implement spiking behaviour and recurrence found in the human brain. This newer form of SNN is as powerful as the classic feed-forward but suited for temporal data encountered in control.
While state of the art feed-forward networks are still outperforming SNNs[1], in some cases modern SNNs are on par[49] or more performant with previous feed-forward implementations consuming less energy.

## 1.2 Problem

Conventional Feed-Forward neural networks do are not designed to work with temporal data. They are static input output machines. This makes sense in the context of many tasks but at the same time limits the power of these networks. There are workarounds to fit temporal data, for example by sampling the previous values back into the network used for example in time series forecasting [74][85][78] or by quantizing the whole input if the complete time horizon is available i.e. with recorded audio data.
Instead of these workarounds, recurrent neural networks are often proposed for these kinds of tasks. However recurrent neural networks experience problems when training

[1]Most benchmarks are based on static information e.g. images which are adapted to SNNs and therefore do not allow a perfectly fair comparison.

with back-propagation[9]. For Recurrent Neural Networks (RNNs) and deep Feed-forward Neural Networks the gradients used in the back-propagation algorithm can explode or vanish. Different methods have been proposed to combat this problem, e.g. batch normalization[40], using alternative activation functions(ReLU)[59] or gradient clipping[62] to name a few. For recurrent models in particular different architectures have been suggested, most prominently among them the LSTM cell [33] with proven success [51, 57, 66].
Yet, these recurrent designs are not a plausible representation of biological networks. RNNs still operate on a continuous range of values instead of discrete spikes. Utilizing the continuity, they are trained with the biologically implausible global learning rules such as the Back-Propagation algorithm.
Furthermore, SNNs come with the added benefit of consuming less power. Usually deep ANNs are run on Graphics Processing Units (GPUs), especially for training, in which the energy consumption can exceed 400W for modern chips[2]. The brain however is estimated to only consume about 20W [21] for its immense computing capacity. Accompanying the SNN with neuromorphic hardware can yield another boost in efficiency with processors energy consumption in the pJ per Synaptic Operation (SOP)[39] offering huge potential power savings.

## 1.3 Goal

The goal of this project is to create a SNN that can control any given Linear Time Invariant (LTI) system. Furthermore should the Neural Network (NN) be robust against failing neurons or connections.
As an additional constraint, the goal should be reached by using biologically tractable methods as much as possible.
To mimic the brain's learning, we want to use local training rules that are biologically plausible.
Ideally, optimality should be reached, even though it is often not clear if this is possible. Already in highly researched ANNs this is usually a unattainably strong condition, as conventional NNs using Gradient Descent (GD) only guarantee a local minima.
Furthermore in nature the brain does not offer separate between training and trial periods. The brain self-modulates its learning online without. This means that the

[2] e.g. a NVidia RTX 4090

network is expected to improve on itself as it working the task at hand.
Lastly, adjusting neural networks to a specific task requires time to tune hyper-parameters by hand to achieve optimal results. Goal here is to automate as much of the process as possible such that the network does not need to get adjusted manually. The network should find acceptable solutions given the task at hand, independent of the given control command or size of the system without problem specific adjustments of hyper-parameters.
For the SNN itself we desire to find a balance between the biologic plausibility and performance. This means we seek key features of biologic networks such as irregular firing patterns, robustness to noise and locality. Additionally, we seek performance when the network controls a given system.
If successful, we would obtain a general purpose controller that would allow us to control any given linear system just by plugging in the given system and the desired reference trajectory.
Furthermore we have a simple and robust controller that would not require expensive computation necessary for e.g. LQG control.

## 1.4 Methodology

To achieve our set goal we first investigate what kind of Spiking network architecture to use. There are many different ways to design a SNN each with its advantages and shortcomings. We seek an architecture that lies in between the most accurate biologic spiking model and yet does allow some abstraction to apply it to our control problem. Part of the goal is that the user does not have to touch the neural network working underneath the control problem. Firstly, we set out to achieve our goal by implementing a SNN that allows to simulate, not control, any given linear system with external inputs. With this network in place another SNN that acts as the controller will be build and combined with the first. The controller would generate a control signal that would be used as external input to the first returning the new system state to the controller in the process.
Afterwards, we equip the network dedicated to simulation with the ability to learn the given system at hand.
Lastly, we combine the two networks into one in order to enable a spiking learning rule at the cost of having to disregard the input matrix.

Is the last paragraph in the right place here?

## 1.5 Outline

make the outline in the end!

First, we discuss why a biologic neural network is chosen compared to the more widely used ANNs in this task and what the goal is. In the

# Chapter 2

# Theoretical Background

In this chapter, a detailed description about background of the degree project is presented together with related work. In the background we cover the autoencoder as is describes the fundamental behaviour of the later chosen network. Additionally, a general overview of SNN architectures and design choices is presented.

## 2.1 Related Work

Spiking neural networks for control have been used in many different contexts, e.g for robotic movement, digit recognition[49] or object detection[68, 91].

In [13] spiking neurons are used to control target reaching movements of a 4-DoF robotic arm using a plausible neuron model and learning rule. In their approach they lean into the DIRECT model of [19], which learns the a priori unknown robot kinematics by randomly repeating movement commands and learning the resulting translation of the end effector.

Another approach for robotic arm control comes from [24] using the Neural Engineering Framework (NEF)[27], in which different regions of the brain have been simulated to generate the trajectory as well as the control signals. On the other hand, the authors of [52] used Reinforcement Learning (RL) in combination with a SNN and a variation of the Spike Time Dependent Plasticity (STDP) rule to control the cart-pole problem.

The authors of [10] summarized several approaches for robotic control of flying or driving robots and used layered spiking neurons with a local learning rule to train the

network to reach the target while avoiding obstacles.
While each of these models incorporate some biologic plausibility, they only take one or a few aspects of biological plausibility into their model, be it the neuron model, the learning rule, encoding or network structure. Furthermore are these approaches often targeted towards the application of robotic arm and not general control of dynamic systems.
Spiking neurons have also been used for PID controllers. In [82] it is shown that a PID controller is possible using three neurons, one for P,I and D respectively. However no example using the network as an example is shown. Furthermore the learning is based on individual parameter tuning for each neuron. Lastly the robustness of 3 neurons is not biologically plausible.
In [69] a PID controller is designed using spiking neurons on neuromorphic hardware in order to control a 1-DoF UAE.
However it is unclear how the network was trained on the neuromorphic hardware or how this approach can be adapted to bigger systems.
Furthermore due to the prevalent background noise found in the brain we stride for a noise robust solution which is not given in any of the above mentioned approaches.

to general?Keep in?

Next to the individually crafted models by authors, various public libraries and frameworks have been developed in order to facilitate SNN simulation. It is important to note that different libraries are aimed towards different purposes. While some exist to act as a hybrid for SNN with Machine Learning (ML) goals (e.g. snnTorch[28], nengo [8]), others are geared for fast and accurate neurlogic simulations (e.g. Brian[70]). Each framework implements a different feature set such as GPU Support, Torch-like structure, convolution, learning rules or support for neuromorphic hardware. See [84] for a summary.

## 2.2 Autoencoder

An autoencoder is a type of neural network for learning a representation of an input signal. It consists of an encoder and decoder function $z = f(x)$ & $\hat{x} = g(z)$. The encoder function maps from the input space, e.g $\mathbb{R}^n$, to a encoded space $\mathcal{C}$. The decoder is then

decoding the data from $\mathcal{Z}$ back to $\mathbb{R}^n$. The goal is that the decoded representation is as accurate as possible. The trivial case is if $\mathcal{Z}$ is equal or larger than the input space. Then every possible input can be encoded by its own value as

$$\begin{aligned} z &= f(x) = x \\ \hat{x} &= g(z) = z = x \end{aligned} \tag{2.1}$$

and the autoencoder simply memorizes each input. In most cases however $\mathcal{Z}$ is constrained such that the autoencoder has to find the characteristic properties of the input. This is often done by reducing the dimension if $\mathcal{Z}$ or by regularization. Regularization is added to the Loss

$$L(x, \hat{x}) + C(z) \tag{2.2}$$

where L can be e.g. MSE and the regularization term $C$ can enforce sparsity or other properties[30].
The functions $f$ and $g$ can be selected freely but play a crucial role in the performance. Frequently, they are combined in a neural network that is trained to adjust the network's parameters as well as function parameters. Apart from $f$ and $g$, the internal mechanism can also be freely chosen. For this reason, an autoencoder can also have a SNN at its core, with $f$ and $g$ serving as the encoder and decoder of spikes, respectively. This step is a pivotal aspect of the selected network architecture, functioning essentially as an autoencoder with spiking neurons. At each time step, a signal is encoded into a group of neurons, from which the spiking dynamics are decoded to reconstruct the desired signal.

Insert graphical illustration. Do it in a later part.When explaining the network itself.

## 2.3 Neural Networks

The topic of NNs is impossible to cover fully and has been examined in countless papers before. Therefore we focus the first section on design choices in neural networks and dedicated spiking models from the literature.
In the design process of a spiking neural network one has to select the different parts

the comprise the network. In the following sections the most important aspects are highlighted and the most common choices explained. As SNNs are still under active research many hybrids and combined methods exist which are not covered here.

### 2.3.1 Spiking Neural Network Choices

Copying nature to solve engineering problems is not a novel practise and also for neural networks this is not new. Many different network architectures with different levels of biologic plausibility have been investigated and published and there is little consent in design decisions. As a result different choices are made by different researches leading to various approaches that are similar and yet different.
Biological realism can be incorporated at various points within the network. Therefore, inspecting each nuance explicitly becomes impractical. Apart from highly specific implementations illustrated below, many SNNs can be categorized into different segments, showcasing certain designs found in the literature.

#### Neuron model

As outlined in more in detail below neuron models can have different complexities and accuracy. By far the most used model in the literature are Leaky-integrate-and-fire (LIF) and Integrate and Fire (IF) model.

**Biological Neuron model** The first biologically accurate model of neuron spiking behaviour is the Hodgkin–Huxley (HH) model from 1952[35]. Since then the HH model has been extended in multiple ways to cover more e.g. different ion channels. The HH-model considers the neuron with its ion channels. The membrane acts as a capacitance and the travelling ions in each ion channel contribute a current to the overall membrane potential. These ion gates are voltage dependent and are defined positive in direction out of the cell.
A particular ion channel for ion $X$ can be modelled as

$$I_X = g_X \cdot (V - V_X) \tag{2.3}$$

These currents are summed for the different ion channels in question, most frequently for Sodium, Potassium and a leak current. In reality there are a plethora of different channels and channel properties[1]. The $V_X$ are the equilibrium potentials for each of the channels and can be computed using the Nernst equation [46].

$$C\frac{dV}{dt} = g_{Na} \cdot (V - V_{Na}) + g_K \cdot (V - V_K) + g_l \cdot (V - V_l) \tag{2.4}$$

Do model the voltage dependency of the ion channels, the conductances are described with gating variables, usually called $n$, $h$ and $g$ for Na-Activation, Na-Inactivation and K-activation respectively. One gating variable is set between $[0, 1]$ and models the permeability of said gate. Multiple gates are used to fit to each ion channel in order to match experimental data and the model behaviour.
Gates have first order dynamics of the form

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n \tag{2.5}$$

for e.g the n gate. The other gates' dynamics are analogous. The functions $\alpha$ and $\beta$ are voltage but not time dependent. The discussion of initial values as well as functions for $\alpha_p,\ \beta_p\ \ p = (n, h, m)$ can be found in [34] or [46]. The gates' conductance for each ion channel have been derived from natural approximations as follows:

$$\begin{aligned} g_{Na} &= \bar{g}_{Na} n^4 \\ g_K &= \bar{g}_K m^3 h \end{aligned} \tag{2.6}$$

and give form to the final model

$$\begin{aligned} C\frac{dV}{dt} &= I(t) - \bar{g}_{Na} n^4 (V - V_{Na}) - \bar{g}_K m^3 h (V - V_K) - g_L (V - V_L) \\ \frac{dn}{dt} &= (1 - n)\alpha_n(V) - \beta_n n(V) \\ \frac{dm}{dt} &= (1 - m)\alpha_m(V) - \beta_m m(V) \\ \frac{dh}{dt} &= (1 - h)\alpha_h(V) - \beta_h h(V). \end{aligned} \tag{2.7}$$

We did not define a gate for the leak term as it is assumed constant.

[1] See `channelpedia.epfl.ch` for an extensive list

**IF and LIF** In contrast of the HH model in eq. (2.7), the simplest models of neurons are the IF and LIF models.

**IF Neurons** IF Neurons, as the name implies, integrate the incoming current over time.

$$\frac{dV(t)}{dt} = \frac{1}{C} I(t) \tag{2.8}$$

The membrane voltage is governed by the incoming current spikes of connected neurons and the membrane capacitance. The neuron potential does not change without a change of input current and thus presents as a perfect integrator of the input.

**LIF Neurons** In contrast to that the LIF neuron contains a leak term on the RHS which brings the voltage back to its resting potential over time. The model can be expressed as

$$\tau \frac{dV(t)}{dt} = -(V(t) - E_r) + RI(t), \tag{2.9}$$

where $\tau = RC$ is the time constant the composed of the membrane resistance $R$ and the membrane capacitance $C$ and the resting potential $E_r$. In the absence of input $I(t)$ the voltage settles on the membrane potential $E_r$.
The input $I(t)$ encapsulates external inputs as well as a sum of Dirac functions indicating a firing neuron.

$$I(t) = \sum_k w_i \delta(t - t_i^k) \tag{2.10}$$

Every received spike $k$ is multiplied by its respective synaptic strength from neuron $i$ and added towards the input current. When the membrane voltage exceeds the threshold potential $\bar{V}$, a spike is sent out by the neuron and the voltage sets back to its reset voltage $V_{res}$.

**Izhikevich Neuron** While the above models deliver a useful and cheap model, they lack in accuracy. The Izhikevich model [41] of the neuron tries be the of both worlds in terms of efficiency and accuracy. It is comprised of 2D ODEs with the membrane

potential $v$ defined as

$$\begin{aligned} \frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I(t) \\ \frac{du}{dt} &= a(bv - u). \end{aligned} \tag{2.11}$$

With the chosen factors, the neuron experiences a spike when $u \geq 30$mV, in which case the neuron resets to

$$\begin{aligned} u &\leftarrow u + d \\ v &\leftarrow c. \end{aligned} \tag{2.12}$$

The parameters describe $a$ scale of recovery, $b$ sensitivity, $c$ the reset potential of $v$ and $d$ the reset of variable $u$. Depending on these parameters one can achieve different behaviours of the neuron e.g. regular spiking, fast spiking and low threshold spiking to name a few.

### Encoding

The coding of information plays an important role in a SNN. Since the network is working on discrete spikes, a methodology needs to be implemented to convert e.g continuous values in spikes. To let the network be susceptible to the outer world, a subset of neurons of the network are exposed to these external inputs.

**Time to First Spike (TTFS)** In this coding scheme the information of an input is solely encoded in the time between the external input and the time neurons fire a spike. A simple visualization could be by implementing the stronger the input the sooner the input neuron spikes after the input onset.
An extension of this idea is rank order coding, where the ordering of spikes encodes information[77].

**Phase coding** Phase coding is a slight variation of the TTFS code. In certain areas of the brain show oscillations similar to a clock[42]. The premise of phase coding is that the oscillation of this clock can be used to convey information. Thus, a spike relative to the phase of a clock cycle entails information on the input signal.

**Rate coding** Rate coding transforms the intensity of a signal to a spiking pattern with a corresponding frequency. The intensity is often normalized to realistic spiking frequencies. Since the brain is noisy, the spike is not fired with the respective rat but rather with a Poisson process given suitable rate $\lambda$. Using a Poisson point process comes with the drawback of comparatively high spiking to represent a value accurately due to noise[23].

**Population coding** Population coding extends the neuron encoding to multiple neurons. Instead of one neuron firing, a group of neurons is combined to encode information. This allows for redundancy and robustness. Inside this population the information can be embedded in different dynamics of this population. One way could be to consider the firing rate of the neurons as a group. Alternatively pattern analysis can be performed to read out information in the distribution of spikes in the group.

### Connectivity/Topology

The structure of neural networks is mainly distinguished between hierarchical and recurrent topologies. In hierarchical topologies there are segments of neurons that follow a (often unidirectional) structure. Feed-forward NNs are a simple example of this architecture.
Recurrent networks allow for loops in the connectivity of neurons and therefore enable feedback and temporal patterns.
There can be hybrid implementation where there are different groups of neurons are connected in sequence.
Using RNNs offers a broader range of applications but has to deal with (as of this day) inferior or more complex learning paradigms.

### Plasticity

The choice of plasticity determines how the network adapts its parameters during learning. There are mostly two different routes used in the literature. The dominant of these is the adjustment of synaptic weights similar to ANNs. It is noteworthy that most learning algorithms are based on this approach.
The alternative is to work on adjusting thresholds instead[5, 20]. One can adjust the

frequency or likeliness of a neuron firing by lowering or increasing its threshold making it harder or easier for a neuron to spike. Increasing the threshold after a spike was fired allows for the modelling of refractory periods seen in biological neurons.
Threshold adaptation can also be in conjunction to weight adaption policy. In [71] adaptive synapses are trained alongside connection weights.
Another form of plasticity concerns the retention of knowledge. In most ML applications the learning is solely a part of the deployment of a NN. After the training period the network parameters are fixed and further learning is frozen. This is in opposition to the biological neural networks, where continuous learning is found not only due to the fact of neurons dying and getting replaced continuously and learning of new tasks.
ANNs struggle with adjusting to new tasks while retaining the same performance for previous tasks. Since the learning of the new task is allowed to adjust every weight it completely ignores relevant weights for the previous task.
To combat this different ideas have been suggested. These include fixing parts of the weights, using Drop-Outs or using Elastic Weight Consolidation (EWC)[48]. In EWC, crucial weights are retained by augmenting the cost function with a penalization term on the parameters deviation from the previous task. The proportional factors are computed using the Fisher information matrix after the learning of one task is completed and focus crucial neurons with higher proportional factors.
A similar approach has been proposed with synaptic intelligence[88]. Here a similar penalty term is introduced that computes the neurons individual importance. The difference is that this importance can be computed online. Each synapse tracks a scalar value describing the relative importance of decreasing the loss. This value is the base for the importance calculation that again penalizes deviations of parameter weights from the previous values.
However, as of now this has only been implemented on ANNs with Feed-forward architecture.

### Learning Algorithms

Key to give any NN the ability to solve a task is to learn/train the it. The adaption of weights and biases is necessary to accomplish any functionality based on the underlying data[90]. There are a plethora of different learning techniques available, see [1, 72] for a review. The most fundamental distinction can be made between

supervised, unsupervised and reinforcement learning rules. Another distinction can be made by the biologic plausibility of a given learning rule. For example for ANN the gradient based Backpropagation algorithm is biologically implausible in various ways but most importantly its non-locality, yet immensely successful.
In this context, a loss function $L$ is employed to calculate the derivative $\frac{\partial L}{\partial \theta_{ij}}$, determining the contribution of each neuron's parameters $\theta_{ij}$ to the error. Subsequently, these parameters are adjusted iteratively until a minimum is reached. The computation of gradients is done efficiently using the backpropagation algorithm (reverse accumulation from Automatic Differentiation (AD)), in which the gradients are propagated from the output backwards towards the input by making use of the chain rule and the fact that the NN has a layered structure. An in-debt explanation is given in e.g [30] or [61].
If the chosen network is recurrent, using Back-Propagation (BP) to find gradients is more complex. To remedy this, adaptations of the classic BP algorithm have been proposed.
If the network at hand is recurrent, often the Backpropagation Through Time (BPTT) can be used. Its concept is based on using the conventional Backpropagation algorithm by ”unrolling” the recurrent into a feed-forward network in time. In addition to the original network inputs additional inputs are designated in the feed-forward that feed the internal network state to the next layer, representing the next time step.
With this structure in place the gradients can now be computed either by propagating the error backwards or by forward propagation of the activity gradient. The former is commonly known as the normal BPTT the latter is called Real Time Recurrent Learning (RTRL)[83]. While they allow the computation of the exact gradients, they can both suffer from issues like vanishing or exploding gradients [9, 62] resulting in getting trapped in local minima easily or divergence. This is due to the unrolling in time which causes the network to become deep quickly. For each step of the input sequence fed into the network an extra layer in the unrolled network is added.
Gradient based methods require differentiability and therefore continuity, thus are only applicable for ANNs. This means that they cannot be used for SNNs as spiking introduces discontinuities. Since spikes are discrete events the derivative is zero during no spike periods and undefined during a spike.
Methods have been proposed to use backpropagation in spiking networks[49] yet they still lack a biologically plausible way to learn. The problem is that in biology,

neurons do not have access to the global error from the loss function but only receive information from pre-synaptic neurons.
If the network is spiking and recurrent, the BPTT algorithm can be used in combination to spiking networks. This leads to the SpikeProp[12] algorithm. In SpikeProp the optimization is done on the firing times and the discontinuity is replaced by a linear function. Since, variations have emerged with improved convergence[58], e.g using different replacement functions [60, 76] or adaptive learning weights [67]. Alternatively, instead of optimizing the firing times and replacing the discontinuity on the spike trains, SuperSpike[87] replaces the discontinuities in the membrane potential with an approximate continuous function.

**(Anti-) Hebbian Learning and STDP** Tending towards more biologically plausible methods, STDP is the most prominent method.
It increases synaptic weight of a synaptic connection if a presynaptic spike is followed by a postsynaptic spike and reduces the weight if the spikes occur independently i.e the postsynaptic neuron fires first.
The Hebbian learning rule is one of the oldest learning principles for spiking networks with large experimental evidence in biology (see for example [29] for a summary). Its key idea can be summarized in the famous mnemonic ”Neurons that fire together, wire together”. If the postsynaptic neuron fires shortly after the presynaptic neuron, the connection strength is increased. Oppositely, if the postsynaptic neuron fires before the presynaptic, their connection strength is decreased.
The longer the delay between firing activation, the smaller the increase. This behaviour of potentiation and depression is pictured in fig. 2.3.1 and build the bases for the STDP rule. This unsupervised local learning rule is based on Hebb’s principle observed in nature. Since then a plethora of variations have been proposed and demonstrated successful[75, 81, 84]. In addition to the Hebbian rule, there is also the anti-Hebbian rule that is reverses the aforementioned behaviour. This means that regular firing of pre- and postsynaptic neurons is discouraged and more irregular and distributed spiking is favoured. This can be understood as mirroring fig. 2.3.1 along the x-axis.

These rules are based in biology and satisfy the restrictions found in nature. Many learning rules adapted from ANNs lack these properties e.g. locality.

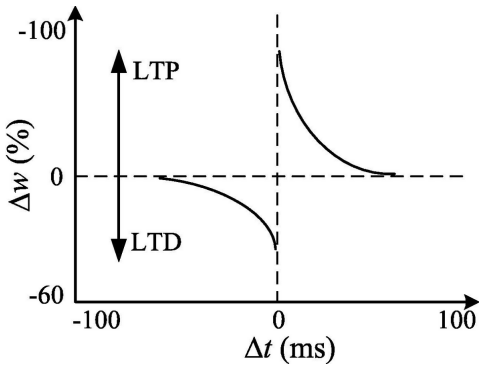



Figure 2.3.1: Graphical representation of STDP learning rule. Weight change depending on the time between pre- and postsynaptic activation. Negative time is the time between the postsynaptic neuron firing before the presynaptic. Graphic taken from [86].

Locality demands that the basis of learning is restricted to having only the information of the direct pre- and postsynaptic neuron available. This already rules out most backpropagation algorithms as the come derive the derivative with respect to a global cost function.

Of course there many variations and extensions have been proposed, e.g. Spike Rate Dependent Plasticity (SRDP)[47] or reward modulated STDP[50].

### 2.3.2 Spiking Neural Networks

Apart from the above mentioned segments of SNNs, different SNN archetypes have been studied in the literature. In the following sections, rate networks as well as Liquid state machines will be presented. Note that the following sections do not represent a comprehensive review but rather a select choice of options. Next to the SNN types mentioned below, other approaches exist, e.g. the NEF[27].

#### Rate Networks

Poisson or Rate networks are built around the idea that information is encoded in the firing rate of a neuron. The precise timing of a spike is essentially meaningless[18]. This is in contrast to the approach chosen here. While states are decoded using neuron firing rates, each fired spike is times exactly in order to minimize a cost function.
The encoding of a value, e.g. four, is set by endowing the input neurons with a Poisson point process with a suitable encoding rate $r_i$[23].

One typically uses probabilistic stimuli because observations in the spiking of the human brain do are different in a trial by trial basis.
Input spikes are travelling through the recurrent network with weighted connections. The decoding is done by counting the spikes of output neurons over a certain time window. The time window plays a crucial rule in the decoding. If it is set smaller, spatio temporal patterns can be captured which can convey information about the input. Equally the sensitivity to noise becomes higher. If the time window is set to large, the firing patterns are lost due to exceedingly large averaging though the impact of random spikes is reduced.
Using this method is comparatively simple way of encoding as firing rates are used in favour of the individual spikes which are modelled by random processes. Additionally this approach is biologically not completely unsound. In nature it has been shown that the firing rate does convey information about the stimuli's magnitude. [2].
The connection weights are subject to change over the learning/training period[4] and can be trained with different training algorithms e.g. (anti-)Hebbian, STDP, gradient based or more involved training methods[22]. See [86] for an overview.

The issue with a Poisson process to put out spikes for its respective rate is that the Poisson process needs many spikes to transmit a signal accurately. For N neurons representing a given value the error or variance scales with $\frac{1}{\sqrt{N}}$[11]. This means to get accurate results a huge amount of spikes need to be fired. If the time between spikes follows an exponentially distributed pattern, and having a greater number of samples enhances the accuracy of estimating the rate parameter.
Theoretically using a large number of spikes is unproblematic. Even with large spike counts neuromorphic hardware is still highly energy efficient compared to deep neural networks when deployed[39].

Though there are more issues to this method. Firstly, this approach has the problem that responses are limited by the time window in which spikes are counted[6]. This implies that the rate decoding can be insufficiently swift to capture rapidly propagating information [32], necessitating the exploration of faster means to transmit information. The second problem is that due to the Poisson process one needs more spikes to represent an average firing rate.
Despite the aforementioned challenges, rate-encoded SNNs have garnered interest in

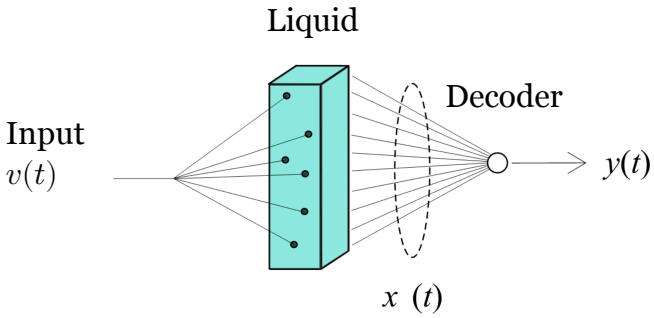



Figure 2.3.2: Abstract idea of Reservoir Computing. Adapted from [53]

research. A big hurdle of deploying SNNs is the lack of performant learning algorithms. For this there efforts have been made to train recurrent or convolutional ANNs using backpropagation and convert the trained network afterwards to a SNN[64] using rate encoding[26][25].

### Liquid state machines

One alternative method has been the use of Liquid State Machine (LSM) or more general Reservoir computing.
The term Reservoir computing was introduced by Benjamin Schrauwen and describes a general group of recurrent network approaches[80].
The ”reservoir” is a non-linear map from input to outputs that combines the input in various, even random ways. These contain but are not limited to sums, differences, multiplications, division and exponentiation. In general the output $\mathbf{x}(t)$ is higher dimensional that the input $\mathbf{v}(t)$, in order to allow for sufficient variety in the mapping. The output of the reservoir, which is usually treated as a black box, is fed in a linear decoder in order to retrieve the desired output signal.
The liquid can be made of any system that fulfils two properties.

- Non-linear nodes of computation
- Fading memory

To these points it is usually set for the system to be time invariant[53]. A reservoir can be a mathematical abstract formulation or physical object, e.g. a literal bucket of water [73].
After the choice of ”liquid” in the reservoir is fixed, its dynamics are not altered.

Only the linear decoder is trained to return the desired decoded output[43]. This is a considerable time saving since the training of recurrent networks is expensive. On the contrary the linear decoder can be learned relatively cheaply.
A reservoir computer is called a LSM if one chooses a spiking neural network as the reservoir. The requirements mentioned above are fulfilled by the recurrent structure to retain information of the neurons and its non-linear spiking behaviour.
LSMs are capable of computing any dynamical system of any order of the form of

$$z^{(n)} = G(z, z^{(1)}, z^{(2)}, \dots, z^{(n-1)}) + u \tag{2.13}$$

given a sufficiently large liquid and a suitable feedback and decoder[56] and have been used for speech recognition[44][89] or object detection[68]. The systematic structure can be in seen in fig. 2.3.3. The feedback $K(x, u)$ is a function of the dynamical system input $u(t)$ and the output $x(t)$. The result of $K(x, u)$ is fed back replaces the previous input $v(t)$ into the Liquid. The decoder $h(x)$ is not linear but can be simplified to be in a cost-performance trade-off when using a sufficiently large Liquid.

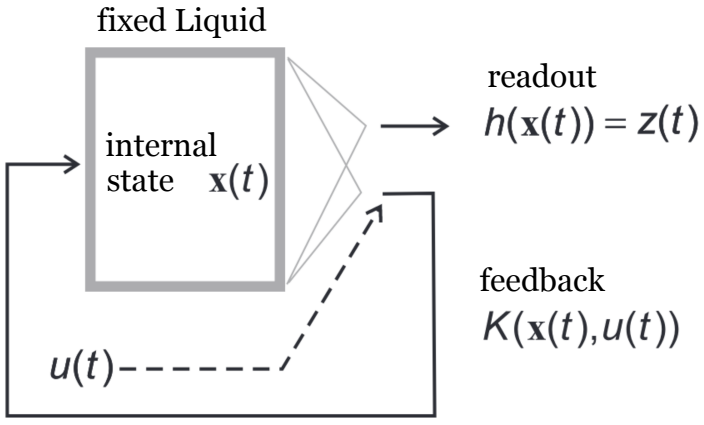



Figure 2.3.3: Adding suitable feedback allows LSMs to be universal approximator. Adapted from [55]

### Balanced Efficient Coding Networks

The idea of tightly balanced spiking networks was first proposed by Boerlin et al.[11]. It uses predictive coding in combination with spikes to simulate arbitrary linear or

non-linear systems[3]. The technical derivation will be described in section 3.3.1. The approach defines a cost function measuring the networks' accuracy in addition with regularization terms that moderates the spiking behaviour. Using a greedy optimization algorithm this cost function determines the voltage threshold and therefore the neurons' spiking behaviour.
For each neuron voltage can be understood as a projection of the global system error to a local error. One neuron is only tracking the system error under this projection. When the error under this projection reaches a threshold, a spike is fired.
The firing of a neuron resets its voltage as well as correcting the system to reset the perceived projected error.

Balanced networks differ from the previous rate encoding in that excitation and inhibition is closely tracked. In rate encoded networks both inhibitory and excitatory spikes are received by a single neuron. An change of the variable is then governed by which type dominates. Here a rate coding is used, however in the matter of control, a combination with instantaneous decoding is [45] utilized. The difference to typical rate networks is given by the fact that each spike distinct information.
Each neuron's projects some information about the state and the network. By projecting the system error in different directions, we assign each neuron to track one projection error. The projection is called voltage. If a neurons voltage reaches the threshold, the projected error exceeds the threshold, a spike is fired and counteracting the projected error.
The projection of the system error therefore has immense influence on an individual neuron's spiking behaviour but also the whole network.
On the other hand in rate networks, a population of neurons is given a value to represent and the population spiking rate encodes this value. An individual spike in this population does not have a direct relevance to the system and a discrete impact on its behaviour.

is this sufficiently explained?

# Chapter 3

# Methodology

Research question: Develop a biologically sensible SNN to control any linear dynamical system.

## 3.1 Choice of Network architecture

The field of SNNs is under ongoing research. Therefore many different network models and learning approaches have been proposed e.g. LSMs, REACH[24]. To follow the idea that every spike contains information, purely rate based spiking networks are ignored as there is evidence that the precise spike timing is relevant in nature[18, 65].

On the other side the use of HH's model is prohibitively complex. Additionally there are no training/learning rules available to solve such a high level problem.
The choice of Efficient coding networks allows us to retain all most of our biologic plausibility requirements while allowing the use of high level learning rules suited to our problem. Since the aim is to learn system dynamics, the SNN should mirror this by learning the dynamics instead of a decoder in LSMs. Moreover the choice of Balanced networks enables us to potentially learn the decoder in addition to the dynamics as well.

## 3.2 Approach

The construction of our SNN can be separated into several blocks. Firstly, a SNN is created for the simulation of a dynamic system with given external inputs.
As a second stage, we derive a second SNN that acts as the controller, delivering the relevant control signals to the first network.
With the controller in place, we work on learning the dynamics of the first SNN from the ground up. It was not possible to learn the dynamics of the controlling SNN. However it is possible to avoid using a dedicated SNN to act as a controller. By copying the key controlling behaviour of the control network we are able to control the trained network directly.
While this approach comes with the drawback of fixing the control matrix $\mathbf{B}$ to the identity matrix, it allows the whole approach to work without information extra that was not trained before. Moreover the control scheme itself requires some on $\mathbf{B}$ itself, making it a acceptable trade-off.

Does this last paragraph fit here better than a few pages ago?

## 3.3 Simulation of Dynamic systems using SNNs

In the following sections the simulation of dynamic systems using SNNs is derived and explained. This serves as the a basic building block for the attempted method on how to solve our target set out in section 1.3. We begin with the formal derivation of the network dynamics.

### 3.3.1 Balanced network simulation

This section follows the derivation found in [11] and [36]. The goal is to describe a dynamical system of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}(t) \tag{3.1}$$

with $J$ state variables. The estimation is done by leaky integration of spike trains $\mathbf{o}(t)$ in

$$\dot{\hat{\mathbf{x}}} = -\lambda_d \hat{\mathbf{x}} + \mathbf{\Gamma}\mathbf{o}(t). \tag{3.2}$$

$\mathbf{\Gamma}$ is a given matrix of size $\mathbb{R}^{J\times N}$, $N$ being the number of neurons. This matrix is given as initial and can be optimized by training later on[17].

From eq. (3.2) it becomes clear that each spike carries an important information for the state vector as each spike in $\mathbf{o}(t)$ changes the approximation $\hat{\mathbf{x}}$ by some weights of $\mathbf{\Gamma}$. In addition to the estimate $\hat{\mathbf{x}}$ we define a spiking rate variable $\mathbf{r}$ following the dynamics of

$$\dot{\mathbf{r}} = -\lambda_d \mathbf{r} + \mathbf{o}(t). \tag{3.3}$$

The rate variable is connected to the state vector in the decoding with

$$\hat{\mathbf{x}} = \mathbf{\Gamma} \mathbf{r}. \tag{3.4}$$

The spiking dynamics arise from the minimization of a cost function. A spike is fired if it minimizes the cost function that tracks the error between the true and estimated value over time

$$E(t) = \int_0^t \|\mathbf{x}(u) - \hat{\mathbf{x}}(u)\|_2^2 \, du. \tag{3.5}$$

### 3.3.2 Greedy optimization of the cost

The cost function eq. (3.5) is minimized using a greedy optimization i.e. a spike is fired if it reduces the cost. For the derivation we use the cost function eq. (3.23) which is identical to setting $\mu = 0, \nu = 0$.
We express this as

$$E(t|i \text{ spike}) < E(t, i \, \overline{\text{spike}}) \tag{3.6}$$

If there is no spike fired, the rate and estimated state variable in eq. (3.2) and eq. (3.3) respectively behave as

$$\begin{aligned} \dot{\hat{\mathbf{x}}} &= -\lambda_d \hat{\mathbf{x}} \\ \dot{\mathbf{r}} &= -\lambda_d \mathbf{r} \end{aligned} \tag{3.7}$$

and therefore decay exponentially with $e^{-\lambda_d t}$.
If a spike is fired at time $t^k$, the inhomogeneous solution is found by variation of

constants in eq. (3.3) to

$$\begin{aligned} r_i^h &= c_i(t)e^{-\lambda_d t} \\ c_i'(t)e^{-\lambda_d t} - c_i(t)\lambda_d e^{-\lambda_d t} &= -\lambda_d c_i(t)e^{-\lambda_d t} + \delta(t-t^k) \\ c_i'(t) &= \delta(t-t^k)e^{\lambda_d t} \\ c_i(t) &= e^{\lambda_d t^k}\boldsymbol{H}(t-t^k) \\ r_i &= e^{-\lambda_d t} + e^{-\lambda_d(t-t^k)}\boldsymbol{H}(t-t^k). \end{aligned} \tag{3.8}$$

The last equation is the identical the solution of eq. (3.7) with the addition of a decaying exponential added at time $t_i^k$. $\boldsymbol{H}(t)$ denotes the Heaviside step function. Analogously the estimate is updated at time $t^k$ to

$$\mathbf{x} = \mathbf{x} + \boldsymbol{\Gamma}_i e^{-\lambda_d(t-t^k)}\boldsymbol{H}(t-t^k). \tag{3.9}$$

We look at the error a $\epsilon$ time in the future of $t^k$ and check eq. (3.6)

$$\begin{aligned} &\int_0^{t^k+\epsilon} \Bigg( \underbrace{\left\|\mathbf{x}(u) - \hat{\mathbf{x}}(u) - \boldsymbol{\Gamma}_i h(u-t^k)\right\|_2^2}_{\text{I}} + \underbrace{\nu \left\|\mathbf{r}(u) + \lambda_d \mathbf{e}_i h(u-t^k)\right\|_1}_{\text{II}} \\ &+ \underbrace{\mu \left\|\mathbf{r}(u) + \lambda_d \mathbf{e}_i h_d(u-t)\right\|_2^2}_{\text{III}} \Bigg) du \\ &< \int_0^{t^k+\epsilon} \left(\|\mathbf{x}(u) - \hat{\mathbf{x}}(u)\|_2^2 + \nu\|\mathbf{r}(u)\|_1 + \mu\|\mathbf{r}(u)\|_2^2\right) du \end{aligned} \tag{3.10}$$

where we abbreviated $h(u) = e^{-\lambda_d(u)}\boldsymbol{H}(u)$. To treat each term individually we start with I. Simplifying the norm we obtain

$$\text{I} = \left\|\mathbf{x}(u) - \hat{\mathbf{x}}(u)\right\|_2^2 - 2h(u-t^k)\boldsymbol{\Gamma}_i^T\left(\mathbf{x}(u) - \hat{\mathbf{x}}(u)\right) + h^2(u-t^k)\boldsymbol{\Gamma}_i^T\boldsymbol{\Gamma}_i. \tag{3.11}$$

For II the 1-norm and the rate holds that the $r_i(u) > 0 \quad \forall i$. Thus we can simplify $\|r\|_1 = \sum_k r_k$ resulting in

$$\text{II} = \nu\left(\|r\|_1 + h(u-t^k)\right). \tag{3.12}$$

Similarly to I, III can be simplified by $\|\mathbf{r}\|_2^2 = \mathbf{r}^T\mathbf{r}$, giving

$$\text{III} = \mu\|\mathbf{r}\|_2^2 + \mu h^2(u-t^k) + 2\mathbf{r}\cdot\mathbf{e}_i h(u-t^k). \tag{3.13}$$

After cancellation the remaining terms are grouped grouped by time dependency to yield

$$\begin{aligned}&\int_0^{t^k+\epsilon} h(u-t^k)\mathbf{\Gamma}_i^T\left(\mathbf{x}(u)-\hat{\mathbf{x}}(u)\right)-\mu r_i(u)du\\&>\frac{1}{2}\int_0^{t^k+\epsilon} h^2(u-t^k)\mathbf{\Gamma}_i^T\mathbf{\Gamma}_i+\nu h(u-t^k)+\mu h^2(u-t^k)du\end{aligned}\tag{3.14}$$

Using the fact that the Heaviside function in eq. (3.8) and subsequently in $h(u)$ allow us to change the borders of integration to $\int_{t^k}^{t^k+\epsilon}$. Lastly we simplify $h(t)=1$ if $t\approx\epsilon$ and have

$$\mathbf{\Gamma}_i^T\left(\mathbf{x}-\hat{\mathbf{x}}\right)-\mu r_i>\frac{\|\mathbf{\Gamma}\|^2+\nu+\mu}{2}\tag{3.15}$$

We notate the Left Hand Side (LHS) as the voltage

$$V_i(t)=\mathbf{\Gamma}^T(\mathbf{x}(t)-\hat{\mathbf{x}}(t))-\mu r_i(t)\quad i=1\dots N.\tag{3.16}$$

and the constant Right Hand Side (RHS) as the voltage threshold $T_i$

$$V_i>T_i=\frac{\|\mathbf{\Gamma}_i\|^2+\nu+\mu}{2}.\tag{3.17}$$

The dynamic variable $\mathbf{x}$ is tracked by firing spikes in when the defined "voltage" of a neuron surpasses its threshold.
For negligible quadratic cost $\mu$ the voltage can be understood as measure of the error projected on $\mathbf{\Gamma}_i$.

### 3.3.3 Neuron Voltage

As mentioned above, a neuron spikes if it meets the condition eq. (3.17). But so far it is unclear how neuron voltage evolves over time. Denote $\mathbf{L}$ the the left pseudo-inverse of $\mathbf{\Gamma}$

$$\mathbf{L}=\left(\mathbf{\Gamma}\mathbf{\Gamma}^T\right)^{-1}\mathbf{\Gamma}\tag{3.18}$$

such that $\mathbf{L}\mathbf{\Gamma}^T=\mathbf{I}$.
Next, taking the derivative of eq. (3.16) yielding

$$\dot{\mathbf{V}}(t)=\mathbf{\Gamma}^T\left(\dot{\mathbf{x}}(t)-\dot{\hat{\mathbf{x}}}(t)\right)-\mu\dot{\mathbf{r}}(t).\tag{3.19}$$

Now using the pseudo-inverse to rewrite the voltage equation eq. (3.16) as

$$\begin{aligned} \mathbf{V}(t) &= \mathbf{\Gamma}^T(\mathbf{x}(t) - \hat{\mathbf{x}}(t)) - \mu\mathbf{r}(t) \\ \mathbf{LV}(t) &= (\mathbf{x}(t) - \hat{\mathbf{x}}(t)) - \mu\mathbf{Lr}(t) \\ \mathbf{x}(t) &= \mathbf{LV}(t) + \hat{\mathbf{x}}(t) + \mu\mathbf{Lr}(t) \end{aligned} \tag{3.20}$$

Now the derivative terms in eq. (3.19) are replaced with with their respective equations eq. (3.1), eq. (3.2) and eq. (3.3). Lastly we substitute eq. (3.20) in eq. (3.19) and obtain

$$\begin{aligned} \dot{\mathbf{V}} &= \mathbf{\Gamma}^T\mathbf{ALV} \\ &+ \left(\mathbf{\Gamma}^T\mathbf{A\Gamma} + \mu\mathbf{\Gamma}^T\mathbf{AL} + \lambda_d\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\lambda_d\right)\mathbf{r} \\ &+ \left(\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\mathbf{I}\right)\mathbf{o} + \mathbf{\Gamma}^T\mathbf{c}. \end{aligned} \tag{3.21}$$

The last argument is to consider the network behaviour for larger networks. We increase the number of neurons $N \longrightarrow \infty$ and require that the network output as well as the firing rates remains constant.
When looking at the decoding at eq. (3.4) we therefore need to scale $\mathbf{\Gamma}$ by $\frac{1}{N}$. To make sure that the threshold in eq. (3.17) will not get dominated by cost terms $\mu$ & $\nu$, they should also scale with $\frac{1}{N^2}$. As the threshold decreases with $\frac{1}{N^2}$ so does the Voltage itself. With this in mind, all terms that scale with $\frac{1}{N^2}$ are neglected. As a substitute for the neglected voltage term, a generic leak term is added making these LIFs neurons. The dynamics are therefore

$$\begin{aligned} \dot{\mathbf{V}} &= -\lambda_V\mathbf{V} + \mathbf{W^s r} + \mathbf{W^f o} + \mathbf{\Gamma}^T\mathbf{c} \\ \mathbf{W^s} &= \mathbf{\Gamma}^T(\mathbf{A} + \boldsymbol{\lambda}_d\mathbf{I})\mathbf{\Gamma} \\ \mathbf{W^f} &= -\left(\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\mathbf{I}\right) \end{aligned} \tag{3.22}$$

### 3.3.4 Regularization

Two regularization terms were added to influence spiking behaviour.

$$E(t) = \int_0^t \left(\|\mathbf{x}(u) - \hat{\mathbf{x}}(u)\|_2^2 + \nu\|\mathbf{r}(u)\|_1 + \mu\|\mathbf{r}(u)\|_2^2\right) du \tag{3.23}$$

The parameter $\nu$ controls the amount of spiking by penalizing the total number of

spikes as

$$||r(t)||_1 = \sum_i |r_i(t)| = \sum_i r_i(t). \tag{3.24}$$

The firing rate is directly related to the number of spiking and therefore the cost is reduced by fewer spikes.

The second term solves different issues at the same time. One problem concerns networks that have decoding kernels with the same direction but opposite sign. To show this we imagine a network of only two neurons. A network of two neurons is sufficient to simulate a scalar ordinary differential equation (ODE) i.e $\mathbf{A} \in \mathbb{R}$. We further assume that the feed-forward weights $\mathbf{\Gamma}$ has the form

$$\Gamma = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{3.25}$$

Ignoring the cost terms in eq. (3.17) the threshold is set at

$$V_i > \frac{\|\mathbf{\Gamma}_i\|^2}{2} \tag{3.26}$$

after which that a spike is fired and the voltage of neuron $i$ resets to

$$V_i = V_i + \mathbf{W}^{\mathbf{s}}_{ii} = V_i + \|\mathbf{\Gamma}_i\|_2^2 \quad \text{with } \mathbf{W}^{\mathbf{f}} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.27}$$

ideally setting the Voltage to $-T_i$. This can be seen when looking at the threshold as

$$T_i = \frac{\|\mathbf{\Gamma}_i\|^2}{2} = \frac{-\text{diag}(\mathbf{W}^{\mathbf{f}})}{2}. \tag{3.28}$$

The repolarization of the spiking neuron acts as a depolarization or pushing the voltage towards its threshold for neurons with opposing sign. The problem now is that for neurons with the same kernel magnitude the depolarization is larger enough to push this neuron over the threshold. The subsequent spike re-polarizes the neuron but in turn excites the first neuron over its threshold. This back and forth pattern of ”ping-ponging” repeats indefinitely.

maybe a picture?

For the given example above, the threshold is given by 0.5 for both. The neurons' voltages of are identical up to the sign, since they are tracking the error for the same variable. At the time one neurons reaches the threshold of 0.5 the second neuron's voltage is close to -0.5 considering noise (in a perfect system minus the value of the spiking neuron). After the spike is fired, the first neuron is reset to -0.5 stemming from

$$\begin{aligned} \mathbf{V} &= \mathbf{V} + \mathbf{W^f o} = \mathbf{V} + \mathbf{W^f} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathbf{V} + \mathbf{W}^{\mathbf{f}}_{:0} \\ \mathbf{V} &= \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \end{aligned} \tag{3.29}$$

$\mathbf{W}^{\mathbf{f}}_{00}$ whereas the second neuron gets pushed up to 0.5 , causing a spike. This in turn reverts the changes of eq. (3.29) resulting in a loop. This problem is caused by the greedy optimization, looking only at the immediate future to decrease the cost.
To fix this the threshold is slightly increased, disallowing a spike to reach the opposing neuron's threshold. As seen above in eq. (3.17), this can be done by either raising the linear or quadratic cost.
The second issue fixed by adding quadratic cost is when there are neurons with similar kernel direction but non normalized. In a perfect noise free scenario the neuron with the smaller threshold will always fire first. The neurons reset after the spike will reset the neurons with similar direction, inhibiting the second neuron from ever firing. The linear cost does not make a difference since it is penalizing the global number of spikes but does not discern where the neurons are firing. By penalizing the rate in the 2-norm it forces the network to spread the firing among the whole network.

## 3.4 Control of Dynamic systems using SNNs

### 3.4.1 Balanced networks as a controller

We now make the step to use the balanced network approach from above as a controller mechanism.
The idea was taken from [38] and is illustrated in fig. 3.4.1. With the given reference signal, the network receives the feedback error of the system. The networks spikes are decoded into a control signal which is further fed into the dynamical system.
The system itself is simulated using a common numerical method i.e. explicit Euler.

Add figure

Figure 3.4.1: Schematic to illustrate the use of balanced networks as controllers.

Yet the goal is to capture the entire problem using SNNs. The control signal $\mathbf{u}$ is generated using the an independent SNN which is in turn the command $c$ for a separate SNN simulating the states with feedback to the controlling SNN.

### 3.4.2 Dynamics

The derivation of this method is similar to the one in section 3.3. Names and variables are reused if not stated here.
The system in question has the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}. \tag{3.30}$$

The basic definitions of the SNN remain the same with rate $\mathbf{r}$ as well as decoding weights $\mathbf{\Gamma}$. Additionally, [38] defines instantaneous decoding weights $\mathbf{\Omega}$ with the same shape as $\mathbf{\Gamma} \in \mathbb{R}^{J \times N}$. It is important to note that $J$ does not represent the number of state variables but the number of inputs. The decoding is the same as in eq. (3.4) with the added $\mathbf{\Omega}$ giving.

$$\mathbf{u}(t) = \mathbf{\Gamma}\mathbf{r} + \mathbf{\Omega}\mathbf{o}. \tag{3.31}$$

The derivation of the network dynamics in [36] is similar to [11] and the derivation presented above. Differences arise in the computation of the cost function as the spike changes the system to

$$\begin{aligned}
\mathbf{u} &= \mathbf{u} + h(t - t^k)\mathbf{\Gamma}_k + \mathbf{\Omega}_k \\
\mathbf{r} &= \mathbf{r} + h(t - t^k)\mathbf{e}_k \\
\hat{\mathbf{x}} &= \hat{\mathbf{x}} + h(t - t^k) \int_0^{t-t^k} e^{(\mathbf{A}+\lambda_d \mathbf{I})\zeta} d\zeta \mathbf{B}\mathbf{\Gamma}_k + e^{A(t-t^k)}\mathbf{B}\mathbf{\Omega}_k
\end{aligned} \tag{3.32}$$

where $\mathbf{\Gamma}_k$ and $\mathbf{\Omega}_k$ correspond to the $k$-th column of $\mathbf{\Gamma}$, $\mathbf{\Omega}$ and $h$ the same as defined above. Results are similar for the rate and control signal whereas the state update is obtained by formally integrating the system. The rest of the derivations are analogous

and completely derived in [38]. The results summarize to eqs. (3.33) to (3.38).

$$\mathbf{V} = \mathbf{\Omega}^T \mathbf{B}^T \left(\mathbf{x} - \hat{\mathbf{x}}\right) - \mu \mathbf{r} \tag{3.33}$$

$$\dot{\mathbf{V}} = -\lambda_V \mathbf{V} + \mathbf{\Omega}^T \mathbf{B}^T \mathbf{c}(t) + \mathbf{W^f o} + \mathbf{W^s r} \tag{3.34}$$

$$\mathbf{c} = \dot{\mathbf{x}} - \mathbf{A x} \tag{3.35}$$

$$\mathbf{W^f} = -\left(\mathbf{\Omega}^T \mathbf{B}^T \mathbf{B \Omega} + \mu \mathbf{I}\right) \tag{3.36}$$

$$\mathbf{W^s} = -\mathbf{\Omega}^T \mathbf{B}^T \mathbf{B \Gamma} \tag{3.37}$$

$$T_i = \frac{\mathbf{\Omega}_i^T \mathbf{B}^T \mathbf{B \Omega}_i + \nu + \mu}{2} \tag{3.38}$$

Note that the notation differs in the original paper and the reference signal is denoted by $\hat{\mathbf{x}}$ instead of $\mathbf{x}$ here and $\mathbf{\Omega}_i$ again refers to the $i$-th column of $\mathbf{\Omega}$.

probably make it like the paper. Regardless it should probably be the state reference value in the definition of c and not the network's state

### 3.4.3 The instantaneous decoding weights

The necessity of instantaneous decoding is necessary otherwise no spiking can occur. In eq. (3.32) the control signal is integrated with the matrix exponential. The problem is that the integral

$$\lim_{t \longrightarrow 0} \int_0^t e^{(\mathbf{A} + \lambda_d \mathbf{I})\zeta} d\zeta = 0 \tag{3.39}$$

for our small $\epsilon$ time horizon.
This is true for any matrix exponential $e^{\mathbf{\Lambda}\zeta}$ seen by Taylor expansion

$$\begin{aligned} \lim_{t \longrightarrow 0} \int_0^t e^{\mathbf{\Lambda}\zeta} d\zeta &= \lim_{t \longrightarrow 0} \int_0^t \sum_{k=0}^{\infty} \frac{(\mathbf{\Lambda}\zeta)^k}{k!} d\zeta \\ &= \lim_{t \longrightarrow 0} \sum_{k=1}^{\infty} t \frac{(\mathbf{\Lambda} t)^{k-1}}{k!} = 0. \end{aligned} \tag{3.40}$$

This means that the rate decoding vanishes in the derivation of eqs. (3.33) to (3.38). Therefore the firing threshold condition becomes

$$-\mu \mathbf{r}_i > \frac{\nu + \mu}{2} \tag{3.41}$$

if $\mathbf{\Omega}$ is ignored which is an insatiable condition since $\mathbf{r}$ is always non-negative.

### 3.4.4 Extension with direct Error feedback

The same group of [38] later published a new but very similar approach in [37] which is based on the same idea, however the approach in [37] makes the error a direct part of the voltage dynamics. The difference arises from the an new derivation avoiding the pseudo-inverse $\mathbf{L}$.
Instead, during the analogous step of eq. (3.19) they set $\hat{\mathbf{x}}$ and $\mathbf{x}$ to follow the same dynamics, namely

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}(t) \tag{3.42}$$

for the reference signal and

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} \tag{3.43}$$

for the system.
In total this adjustment changes the dynamics of eq. (3.34) to

$$\dot{\mathbf{V}} = -\lambda_V \mathbf{V} + \mathbf{\Omega}^T \mathbf{B}^T \mathbf{A}\mathbf{e}(t) + \mathbf{\Omega}^T \mathbf{B}^T \mathbf{c}(t) + \mathbf{W}^{\mathbf{f}}\mathbf{o} + \mathbf{W}^{\mathbf{s}}\mathbf{r}. \tag{3.44}$$

Important to note is that due to the investigation of the network's limit behaviour in [38], similarly done in eq. (3.21) and the subsequent neglect of certain terms, changes the definition of $\mathbf{W}^{\mathbf{s}}$. In the derivation in [37] this is not done and therefore terms remain changing $\mathbf{W}^{\mathbf{s}}$'s definition to

$$\mathbf{W}^{\mathbf{s}} = -\mathbf{\Omega}^T \mathbf{B}^T \mathbf{B}\mathbf{\Gamma} + \mu\mathbf{I}. \tag{3.45}$$

add that there is always noise somewhere

## 3.5 Learning of network parameters

All the methods described above work on the optimally ideal weights for the dynamics. However in nature often new skills or dynamics are learned and not optimally tuned for the problem at hand. In the following chapter the optimally of derived is explained and local learning rules for the weights are introduced. Specifically, the learning rules presented in [14, 15, 17].

### 3.5.1 Learning of fast connection weights $\mathbf{W}^f$

Before deriving the learning rule, we are first going to establish the idea that the minimization of the loss is equal to minimizing the fluctuations of the membrane potential.
This assumption is based on the fact that the Voltage is the projected error of the tracked signal to the reference. Therefore a good tracking should cancel out

$$\mathbf{V} = \mathbf{\Gamma}(\mathbf{x} - \hat{\mathbf{x}}) \approx \mathbf{0} \tag{3.46}$$

neglecting the regularization terms.
Therefore, the average membrane voltage

$$L = \int_0^T \|\frac{1}{2}\mathbf{V}(t)\|^2 dt \tag{3.47}$$

over time serves as a good and simple measure of tracking performance.
Since the Voltage is bound by the threshold, the computation of the average can be reduced to averaging of the spikes.

**Learning with $\mathbf{L_1}$ costs** As such, the minimal variation is given if the voltage threshold and its respective reset are centred around zero. Voltage resets, given by $\mathbf{W}_i^f$ for spiking neuron $i$, need to be strengthened if its reset undershoots the negative threshold value $T_i$ and vice versa. A spiking reset that hyperpolarized the neuron over the negative threshold will be weakened.
Mathematically, this averaging over spikes condition is formulated as the loss at time $t$, using the fact we can express the Voltage post-spike $\mathbf{V}^{\text{post}}$ as

$$\mathbf{V}^{\text{post}} = \mathbf{V}^{\text{pre}} + \mathbf{W}^f \mathbf{e}_i \tag{3.48}$$

for neuron $i$ firing a spike.
Taking the derivative of the loss function in eq. (3.52) with respect to $\mathbf{\Omega}$ yields

$$\frac{\partial L_t}{\partial \mathbf{W}^f} \propto \left(2\mathbf{V}^{\text{pre}}(t) + \mathbf{W}^f \mathbf{e}_i\right) \mathbf{e}_i^T \tag{3.49}$$

Rewriting the equation above and adding a tuning parameter gives the learning rule

$$\Delta \mathbf{W}_{ij}^{f} = -\frac{\partial L_t}{\partial \mathbf{W}_{ij}^{f}} \propto -\left(\mathbf{V}_i^{\text{pre}}(t) + \beta \mathbf{W}_{ij}^{f}\right). \tag{3.50}$$

It is important to note that this learning rule only becomes active during the firing of a spike. This means that if no spike is fired no learning takes place. To introduce spiking and learning, smoothed Gaussian white noise if often used for training and inducing firing of all neurons.

The role of $\beta$ is to regulate the amount of spiking by artificially increasing the magnitude of the recurrent weights. Therefore, with increased magnitude neurons are overly hyperpolarized after a spike making it harder to spike again soon, similar how the linear costs above reduce spiking activity.

For $\beta = \frac{1}{2}$ no regulation is present since in the optimal case $\mathbf{W}^f$ will be twice the threshold Voltage.

Keep this in?

However values $\beta < \frac{1}{2}$ lead to unstable ping pong patterns described previously.

It is important to note that this learning rule is entirely local and can be categorized as a Hebbian learning rule, as the presynaptic voltage is multiplied by the postsynaptic reset, given by the column in $\mathbf{W}^f$.

**Learning with $\mathbf{L_2}$ costs** The previous derivation did only incorporate $L1$ costs using the $\beta$ parameter, but does not implement any $L2$ regularization as in the derivation in section 3.3. With L2 costs, the optimal matrix is given by the addition of the quadratic cost term to the diagonal as in eq. (3.22). Since the cost term is added to the matrix $\mathbf{W}^f$ and the voltage definition in eq. (3.16) the voltage no longer explicitly tracks the projected error. To reuse the argument about minimizing Voltage fluctuations as a performance measure we therefore add the terms again to cancel out the terms from the previous definition. Specifically, we want to work the true projection error $\mathbf{V}_{\text{ex}}$ and therefore compute the loss in eq. (3.52) using

$$\begin{aligned} \mathbf{V}_{\text{ex}} &= \mathbf{V} + \mu \mathbf{r} \\ \mathbf{W}_{\text{ex}}^{f} &= \mathbf{W}^{f} + \mu \mathbf{I}. \end{aligned} \tag{3.51}$$

We then formulate the same argument on the loss

$$\begin{aligned} L_t &= \left\| \frac{1}{2} (\mathbf{V}_{\text{ex}}^{\text{pre}}(t) + \mathbf{V}_{\text{ex}}^{\text{post}}(t)) \right\|^2 \\ &= \left\| \mathbf{V}^{\text{pre}}(t) + \mu \mathbf{r}(t) + \frac{1}{2} (\mathbf{W}^f + \mu \mathbf{I}) \mathbf{e}_i \right\|^2 \end{aligned} \tag{3.52}$$

and take the derivative with respect to $\mathbf{W}^f$, resulting in the learning rule of the form

$$\Delta \mathbf{W}_{ij}^f = -\frac{\partial L_t}{\partial \mathbf{W}_{ij}^f} \propto -\beta (\mathbf{V}_i^{\text{pre}}(t) + \mu \mathbf{r}_i(t)) - \mathbf{W}_{ij}^f - \mu \delta_{ij} \tag{3.53}$$

when a spike is fired.

### 3.5.2 Learning of slow connection weights $\mathbf{W}^s$

The recurrent weights $\mathbf{W}^s$ are called slow because they are in conjunction with the filtered spike train $\mathbf{r}(t)$ in eq. (3.21) in contrast to $\mathbf{W}^f$ which are reset the neuron voltage after a spike.
For learning $\mathbf{W}^s$ an adaptive learning approach from [16] is used.
For this we consider a "learner" dynamic system of the form

$$\dot{\hat{\mathbf{x}}} = \mathbf{M}\hat{\mathbf{x}} + \mathbf{c}(t) \tag{3.54}$$

where the matrix $\mathbf{M}$ changes to allow $\hat{\mathbf{x}}$ to follow the same dynamics of $\mathbf{x}$ given by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}(t). \tag{3.55}$$

Over time, $\mathbf{M}$ should converge to $\mathbf{A}$. To facilitate this, the error $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ is feed back into the learner system $\dot{\hat{\mathbf{x}}} = \mathbf{M}\hat{\mathbf{x}} + \mathbf{c}(t) + K\mathbf{e}$ to direct $\mathbf{M}$. The adjustment of $\mathbf{M}$ in the modified dynamics

$$\dot{\hat{\mathbf{x}}} = (\mathbf{M} + K\mathbf{I})\hat{\mathbf{x}} + \mathbf{c}(t) + K\mathbf{x} \tag{3.56}$$

is then calculated by using minimizing the loss

$$L = \frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{3.57}$$

with respect to the matrix parameters $M_{ij}$ and changing their value according to

$$\dot{M}_{ij} = -\frac{\partial L}{\partial M_{ij}} = \left(\frac{\partial \hat{\mathbf{x}}}{\partial M_{ij}}\right)^T \mathbf{e}. \tag{3.58}$$

Explain the derivative, or its approximation.

It is important to note that this derivation completely ignores any sort of control matrix $\mathbf{B}$ given in a control problem. While this can be a disqualifying property for many systems it is also important to ask how such a matrix could be implemented. Since the complete state of the system is encoded partly in each neuron it is impossible to selectively control certain states. Therefore from now on it is assumed that the control works on each neuron or that the control matrix $\mathbf{B} = \mathbf{I}$ is the identity matrix.

# Chapter 4

# Results

#### Acceptable error

Due to the networks discrete nature of spikes, the error cannot surpass a certain minimal threshold. This threshold is fixed by the fact that a spike of neuron $i$ produces a constant change in the signal given by the decoding $\mathbf{\Gamma}_i$. It is therefore impossible to reach accuracy levels commonly seen with classic control schemes and numerical methods. While it is theoretically possible to achieve arbitrarily small errors by scaling $\mathbf{\Gamma}_i$, the error is bound by the spiking threshold $\frac{\|\mathbf{\Gamma}_i\|+\mu}{2}$ and the uniformity of $\mathbf{\Gamma}$.

add something here maybe? Maybe about relative error or what kind of accuracy we are striving for and how we measure that.

## 4.1 Results on the Simulation from ??

### 4.1.1 Toy Example

The results from simulating the network with given input $\mathbf{c}(t)$ can be seen in fig. 4.1.1. In this scalar example the ODE

$$\dot{x} = -10x + c(t) \tag{4.1}$$

is simulated with 2 neurons. One neuron corrects the network simulation of the system up and down respectively. As shown before this correction happens immediately after

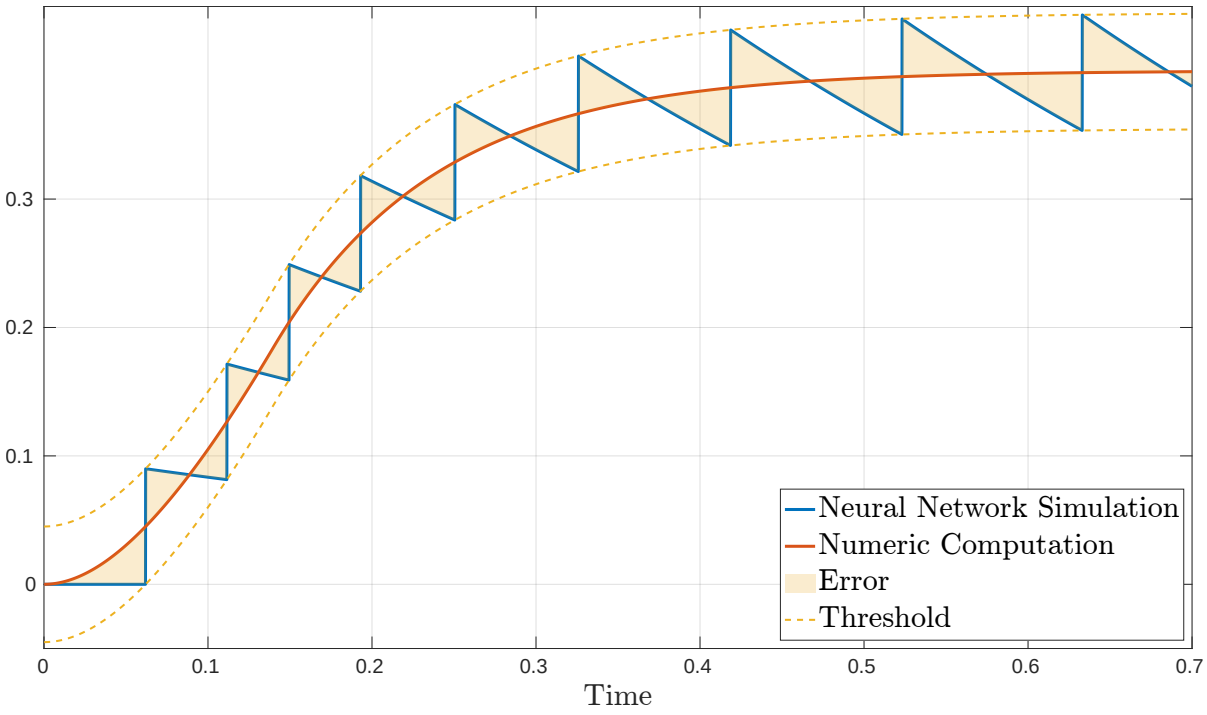
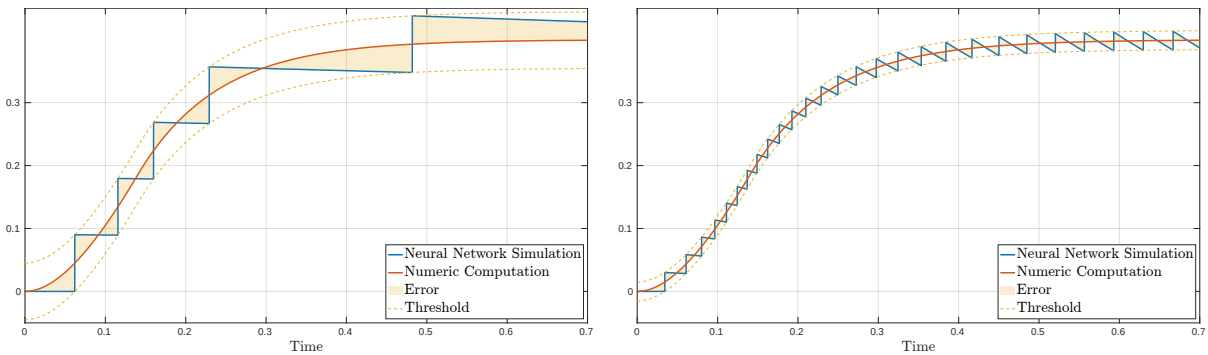



Figure 4.1.1: Baseline example

each spike by adding weights of the relevant decoding vector to the trajectory. Since this example is following a scalar variable with 2 neurons the decoding matrix $\mathbf{\Gamma} \in \mathbb{R}^{1\times 2}$ and set to $[0.09, -0.09]$ for this example. This can be seen in fig. 4.1.1, the neural network simulation jumps up after each spike by 0.09 is added. The external input follows a linear increase until 0.15s after which it remains constant.

For reference a conventional numeric solution is given which lies directly between the two neurons threshold, visualized with dotted lines.

Already for this toy example different parameters can be tuned to get different results.




(a) Reduced Readout Decay rate

(b) Reduced Threshold

Figure 4.1.2: Variation of Readout Decay and Decoder for simple 1D system.

In fig. 4.1.2a the readout decay is reduced compared to fig. 4.1.1 which reduces how fast the output tends to zero. This also elevates the importance of a single spike as it

has longer lasting effects on the output, seen by the system showing fewer spikes than before.
Alternatively, if the decoding weights can be scaled to let each spike make a smaller change in the output seen in fig. 4.1.2b. Since the threshold is closely tied to the Decoding weights this also reduces the spike threshold and therefore yields more accurate results.

### 4.1.2 Toy example in 2D

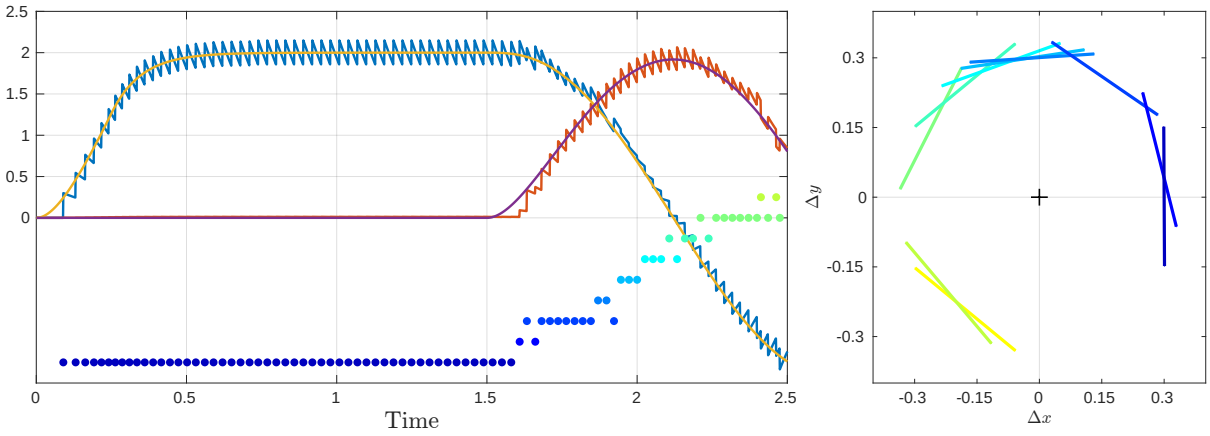



Figure 4.1.3: Simple 2D example with numerical solution and spike response. Curves for $x$ in yellow/blue. Curves for $y$ in purple/red. The networks output closely tracks the perfect numerical solution. For the network each decoding vector was chosen from a normal distribution and normalized to $\|\mathbf{\Gamma}_i\|_2 = 0.3$. Beneath a raster plot for each spiking neuron
On the left the threshold for each neuron's projected error.

### 4.1.3 Geometry in 2D

In two dimensions the network with its allows for a geometric interpretation. For this we let the network simulate a simple leaky integration of inputs as in eq. (4.1) but in two dimensions. We have the corresponding results in fig. 4.1.4 on the right. On the left we a phase plot in the $xy$-axis.

### 4.1.4 Importance of Feedforward/Decoding weights

So far we have not given much attention to decoding/feedforward weights. Yet they play a crucial rule in the performance of our network as seen in fig. 4.1.5. Here we simulate again a simple leaky integration as in eq. (4.1) however this time in 2D. With

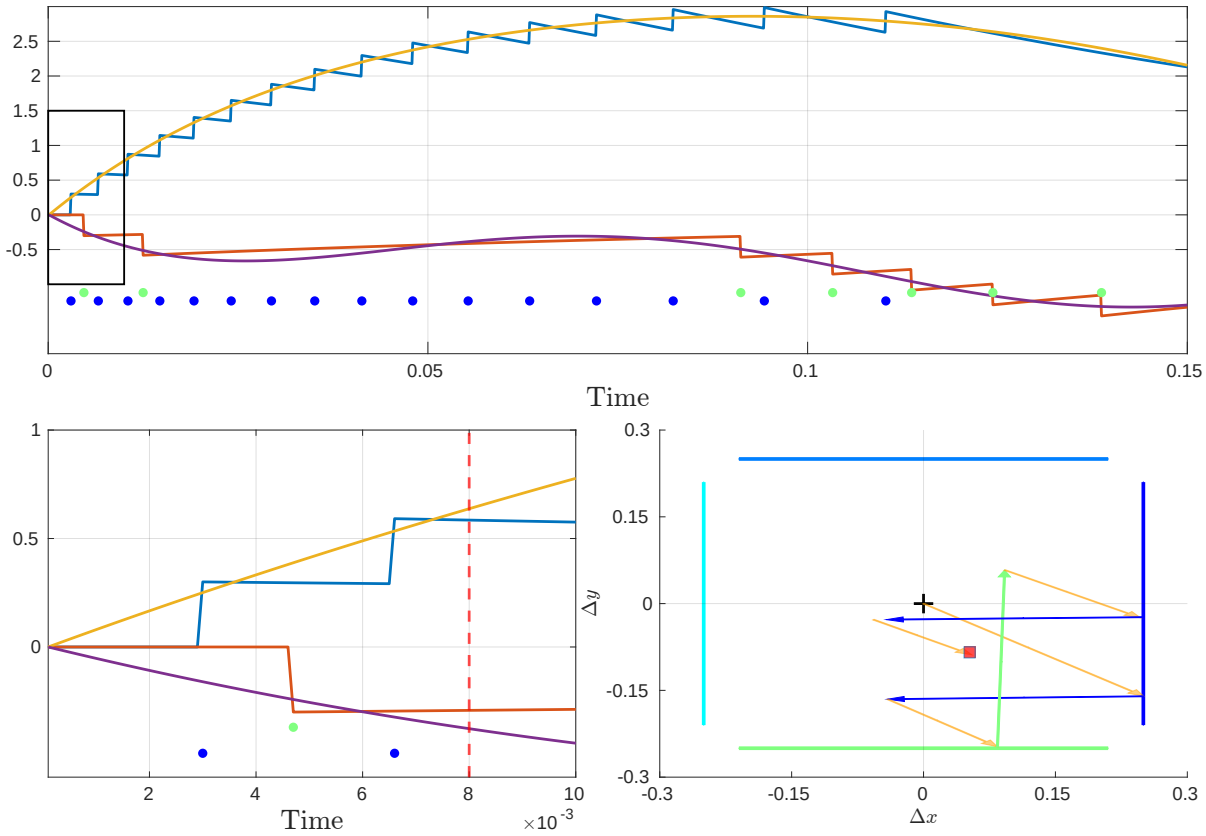



Figure 4.1.4: Simple 2D example with numerical solution and spike response. Curves for $x$ in yellow/blue. Curves for $y$ in purple/red. The networks output closely tracks the perfect numerical solution. For the network each decoding vector was chosen from a normal distribution and normalized to $\|\mathbf{\Gamma}_i\|_2 = 0.3$. Beneath a raster plot for each spiking neuron
On the left the threshold for each neuron's projected error.

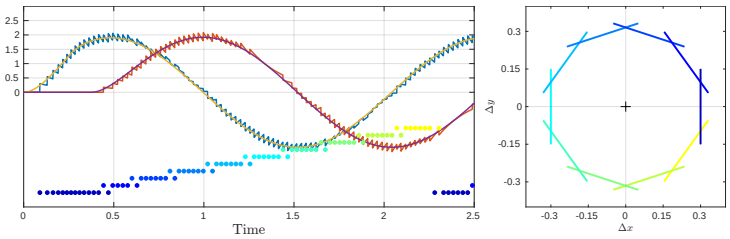
the output, in each test we also plot a bounding box for the relative error.

The bounding box of the error is the normal to each of the decoding weights. In eq. (3.2) the network output gets the $i$th column $\mathbf{\Gamma}$ added when the $i$th neuron spikes. From the minimization of the cost we know that the spike of neuron $i$ reduces the error in projected by $\mathbf{\Gamma}_i$. If we know imagine the origin of the fig. 4.1.5 l to always be on top of the true trajectory of $[x,\ y]^T$, the network's error is just deviation from the origin.

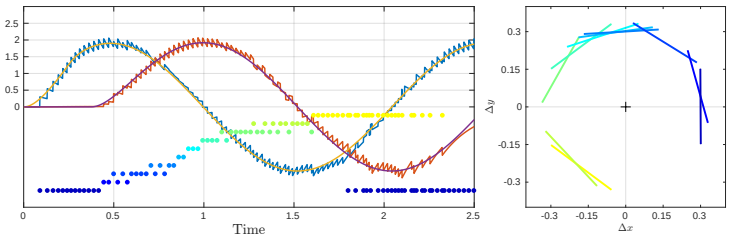
Explain phase plot explain the normal to the decoding weights Explain the 3 plots Dont forget to explain the Trichter in the bad plots

### 4.1.5 Bigger Systems

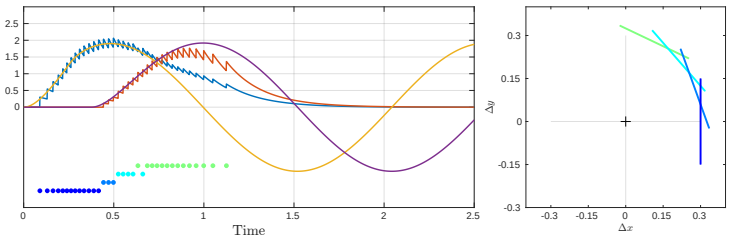
So far we have only looked simple systems in 1D or 2D. However our network can also simulate more complex higher dimensional systems. In the example in fig. 4.1.6 we




(a) Network




(b) Reduced Threshold




(c) Reduced Threshold

Figure 4.1.5: Network output after leaky integration of oscillating input with different decoding vectors.

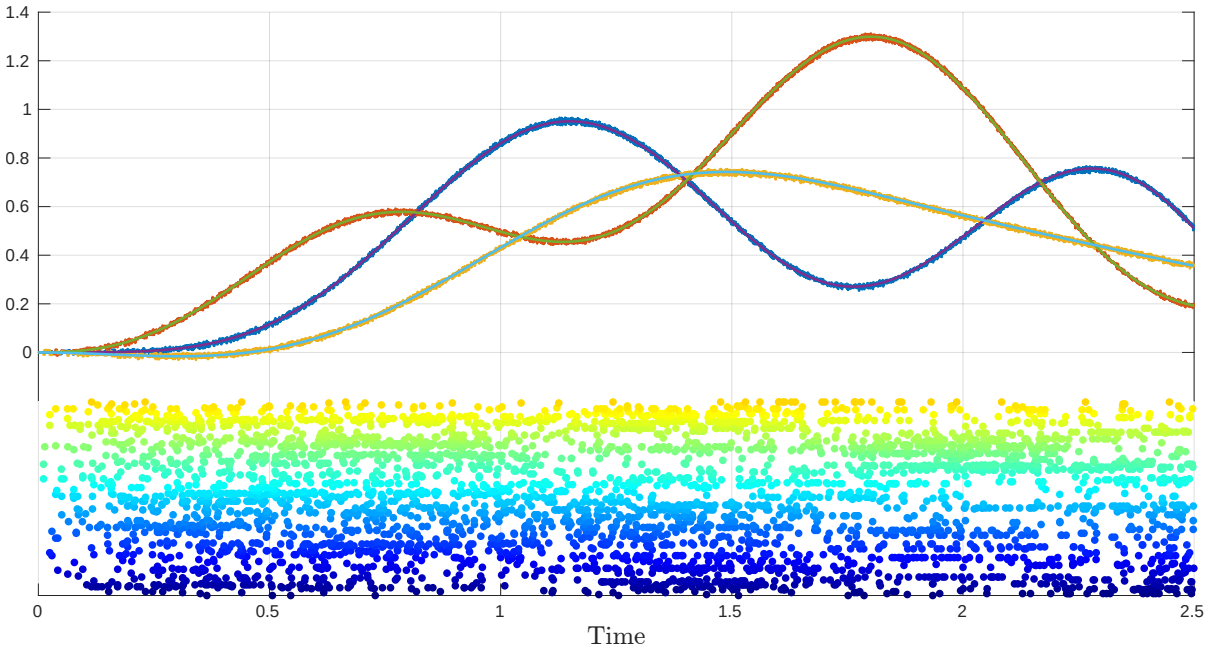


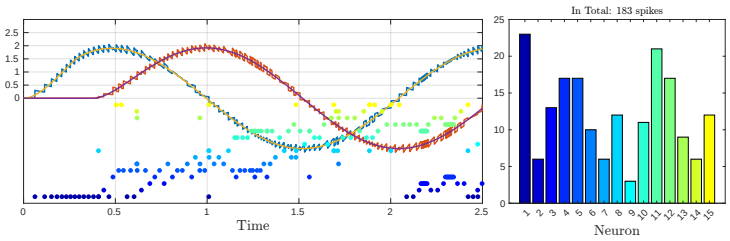

Figure 4.1.6: Trajectory of dynamic mass spring system with 100 masses and 2 thousand neurons. Only the first 3 trajectories are plotted and the spikes for the first 200 neurons.

simulated a linear n-mass spring system with the dynamics

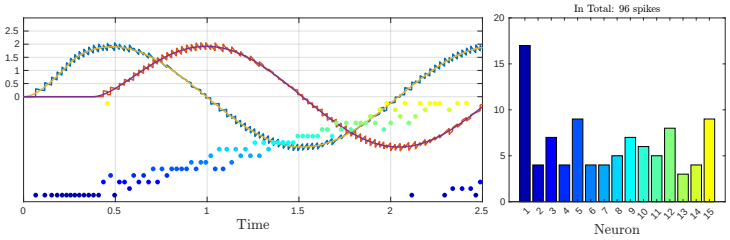
$$m_i\ddot{x}_i = K \cdot \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{bmatrix} + c_i(t). \tag{4.2}$$

The external forces were each offset sinusoidal waves with varying frequency and amplitude for the first 3 masses and random forces for the rest. As can be seen from the figure, the network perfectly overlays the numerical solution.

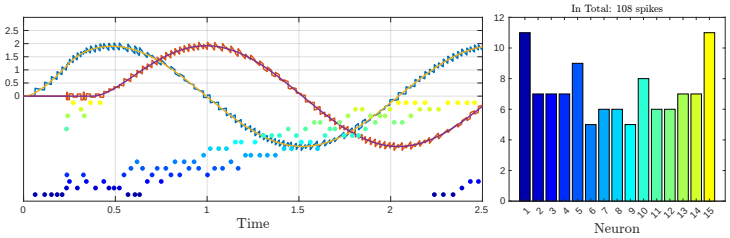
### 4.1.6 Varying cost parameters $\mu, \nu$




(a) Network




(b) Reduced Threshold




(c) Reduced Threshold

Figure 4.1.7: Network output after leaky integration of oscillating input with different decoding vectors.

## 4.2 Results on the control

For the evaluation of the results for the control problem derived in section 3.4 we first note a few important implementational details in order to get useful results.

Afterwards we compare the performance of the two different derivations from sections 3.4.2 and 3.4.4 for specific examples as well as their limits. Lastly we take a closer look at the main reason for this system to work and how we can adopt it to our learning problem in section 3.5.

For exampl dirac doesn work,Actua anything wi discontinui doesnt work???,lon simulation times. All f the one wit error.

Mention th the rate stu kind of use unless you to reduce th number of spikes

### 4.2.1 Implementation details

Before any simulation can be run it is important that certain detail in the implementation are set in the correct order for the results to make sense.

#### Multiple spikes per timestep

Although this also applies for the simulation step in **??**, its affects are much more pronounced in the control setting.

**Single Spike** As the dynamics outlined suggest the simulation step includes a check if a threshold has been reached.
A simple implementation in MatLab pseudo code is seen in Listing 4.1.

```
1
2 for t_step = 1:N_step
3   V = update_Voltages(V,dt,Ws,Wf,c,...);
4   [value, index] = max(V-Threshold);
5   spikes(index,t_step) = 1;
6   V = V + Wf(:,k);
7   ...
8 end
```

Listing 4.1: Single spike implementation

While this works for a variety of problems it does not give results for any given system and reference trajectory.
Especially problems with an abrupt change underperform since the network is only allowed to correct the error by one spike. However by jumps and rapid changes this is not sufficient to compensate errors and then networks struggles for many iterations to recover, ruining the overall accuracy in the process.

**Parallel Spikes** Now the direct way to fix this to find all neurons that have reached their thresholds. Since networks have many neurons, letting every neuron spike increases the networks ability to correct errors. A potential implementation is seen in Listing 4.2.

```

for t_step = 1:N_step
  V = update_Voltages(V,dt,Ws,Wf,c,...);
  spiking_neurons = V>Threshold; % procudes a list
  spikes(spiking_neurons,t_step) = 1;
  V = V + Wf(:,spiking_neurons);
  ...
end
```

Listing 4.2: Letting every neuron spike in parallel

The problem with this implementation is that the error gets tracked by multiple neurons simultaneously and also in both in positive and negative direction. To illustrate this we consider a simple example where we have a single control variable $u$ and $2N$ neurons, where $N$ neurons track positive and negative error respectively. Disregarding noise, the threshold is reached by $N$ neurons synchronously. As long as the total error is larger than the compensation of $N$ neurons in the first place, this works just fine. However this is not permanent and at some point all $N$ neurons firing overly depolarize the opposite side neurons to the extend that they all fire in the next iteration regardless of the real system error.
The root cause is that a single spike influences the whole network. In this extreme case 1 spike resets all other $N-1$ neurons that were spiking as well and therefore would not give any performance boost compared to the previous approach.
The network with this implementation behaves normally while a rapid change is present but will evolve into $N$ neurons spiking in alternating cadence respectively.
To remedy this it is necessary that one dimension of the error is only projected onto two neurons. While this can be achieved by carefully selecting the decoder it is not a generic approach. Moreover this reverses the idea of multiple neurons tracking the error in order to increase the networks performance and would result in zero change. Therefore we need to let a single neuron spike multiple times.

**Multiple Spikes** The correct way to handle this problem is to allow more than one spike per time step but compute each spike's change in the network separately. This

way we can achieve the necessary performance without the problems of the previous approach. A implementation of such a regime is seen in Listing 4.3

```
1
2  for t_step = 1:N_step
3    V = update_Voltages(V,dt,Ws,Wf,c,...);
4    [value, index] = max(V-Threshold);
5    while value > 0
6      spikes(index,t_step) = 1;
7      V = V + Wf(:,k);
8      ...
9      [value, index] = max(V-Threshold);
10   end
11 end
```

Listing 4.3: Letting each neurons spike as many times a necessary while computing each spike's influence sequentially.

In order to retain biologic plausibility, it is necessary to limit the number of spikes fired in one time step. While this is not a problem in most cases when hyper-parameters are properly tuned, it can be an issue if there are sudden jumps or rapid transitions.

### Signed Error and Slow Connectivity

**Neglected Term** First to note is that the derivation of $\mathbf{W}^s$ in eq. (3.37) is different from the definition it [38] in two ways. Firstly in the original derivation the rate is scaled by $\lambda_d$ i.e.

$$\hat{r}(t) = \lambda_d r(t). \tag{4.3}$$

Substituting this in for the equations in section 3.4.2 gives the exact same dynamics. Secondly and more importantly in their derivation consider network in the limit for $N \to \infty$ neurons. Specifically the derivation arrives at

$$\begin{aligned}
\dot{\mathbf{V}}(t) =& \boldsymbol{\Omega}^T\mathbf{B}^T\left(\dot{\hat{\mathbf{x}}}(t) - \dot{\mathbf{x}}(t)\right) - \mu\lambda_d\dot{\mathbf{r}}(t) \\
=& \boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{A}\mathbf{L}\mathbf{V}(t) \\
&+ \left(\mu\lambda_d\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{A}\mathbf{L} + \mu\lambda_d^2\mathbf{I} - \frac{1}{\lambda_d}\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{B}\boldsymbol{\Gamma}\right)\mathbf{r}(t) \\
&- \left(\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{B}\boldsymbol{\Omega} + \mu\lambda_d^2\mathbf{I}\right)\mathbf{o}(t) \\
&+ \boldsymbol{\Omega}^T\mathbf{B}^T(-\mathbf{A}\hat{\mathbf{x}}(t) + \dot{\hat{\mathbf{x}}}(t))
\end{aligned} \tag{4.4}$$

where

$$\mathbf{L} = \left(\mathbf{B\Omega\Omega}^T\mathbf{B}^T\right)^{-1}\mathbf{B\Omega}$$

is the pseudo-inverse of $\left(\mathbf{B\Omega}\right)^T$. Our focus is now set on the terms in front of the rate. As [38] and the original derivation in [11] argue the term $\mu\lambda_d^2$ vanishes in the limit and can therefore be neglected, yielding the above result for $\mathbf{W}^s$ above in eq. (3.37).
However in the testing it was noticed that this derivation only works acceptable for a small range of values for $\lambda_d$, explicitly $\lambda_d \leq 20$. For values above the performance deteriorates rapidly if not a large number of neurons is considered.
After testing the relative importance of terms it was noted that the term $\mu\lambda_d^2$ does improve the performance with a smaller number of neurons significantly compared to the other terms and was therefore added in the results following below.
Furthermore the authors give later examples that show their use of the extra term. This can be seen in all examples but especially in example figure 2. Conveniently the authors provided a picture of the their $\mathbf{W}^s$ matrix which clearly shows a nonzero entry on the main diagonal. The example showcases the case $\mathbf{\Gamma} = \mathbf{0}$ which, given the definition of $\mathbf{W}^s$ in eq. (3.37), yields $\mathbf{W}^s = \mathbf{0}$. Examining the colormap reveals that the values on the diagonal equal to 1 which is the same $\mu\lambda_d^2$ in the given example. The same can be validated for the other examples.
Running the same example results in a subpar performance shown in fig. 4.2.1. To match the behaviour shown in the paper we needed to either involve the omitted term or increase the number of neurons to $N = 500$. Even though in fig. 4.2.2 it might look like the behaviour is similar it is important to point out that over time the network's performance deteriorates over time similar to fig. 4.2.1 and certainly cannot match the behaviour shown in [38].

here maybe a figure showing the comparison with and without the addition might be better?

Table Mit extra term ohne extra term Fehler in einem beispiel mit extra anzahl neuron die man hinzufuegen muss um ausgleich zu haben

### 4.2.2 Performance Comparison

To test both methods, compare both schemes with the same control trajectories.
Both methods require to receive the desired state trajectory as input. To test their

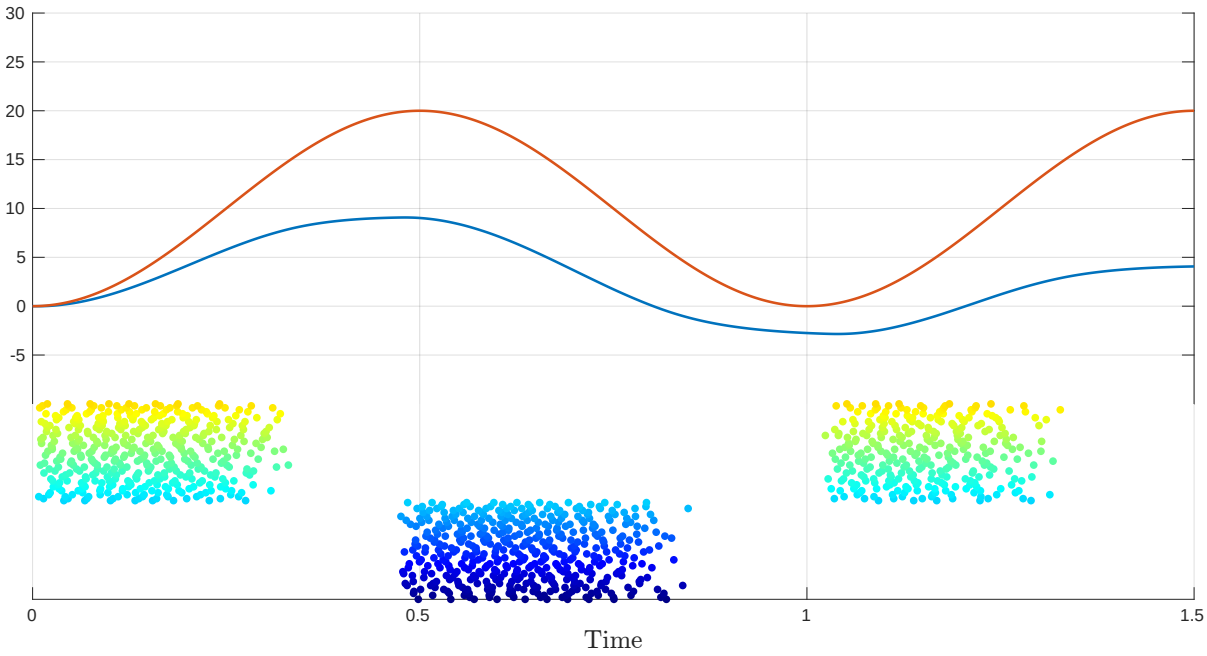



Figure 4.2.1: Control performance of the same the example in figure 2 from [38] without the extra $\lambda_d \mu^2 \mathbf{I}$ term. Simulation with $N = 100$ neurons.

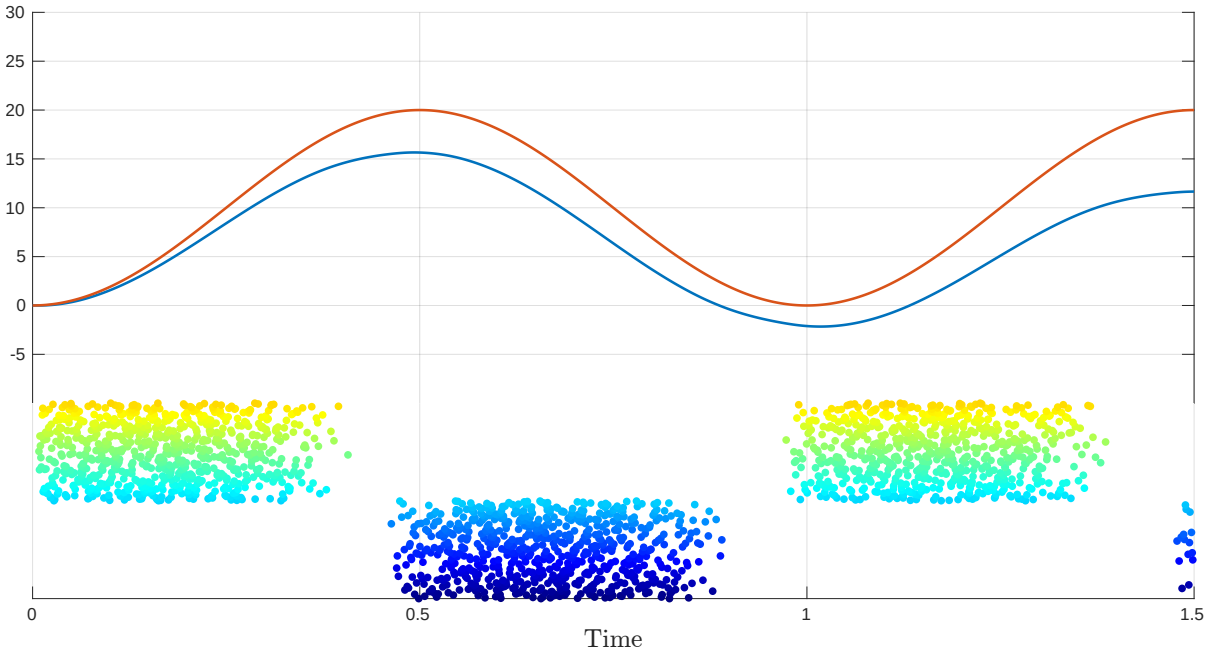



Figure 4.2.2: Control performance of the same the example in figure 2 from [38] without the extra $\mu \lambda_d^2 \mathbf{I}$ but $N = 500$ neurons to compensate. Neurons are split 50:50 as before.

usability, different size problems as well as configurations of $\mathbf{B}$ and $\mathbf{C}$ will be tested.

For testing purposes, we set the test system $\mathbf{A}$ to

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -1 & 10 \end{bmatrix}. \tag{4.5}$$

Figure 4.2.3: Step response for simple 2D system and. Both runs are identical except of $\mathbf{\Gamma} = 0$(top) and $\mathbf{\Gamma} = 100\mathbf{\Omega}$. Both approaches performed identically.

Additionally, for the $N = 50$ neurons we choose their decoding weights randomly and normalize them to 0.01.

#### Ideal B and C

Both methods performed identically well in most test cases with $\mathbf{C}$ and $\mathbf{B}$ chosen to be the identity matrix. Error levels are limited by the scaling of $\mathbf{\Gamma}$ with single spike deviations from the target.
To influence the control behaviour, different parameters can be tuned. Increasing costs $\mu$ and $\nu$ will lead to a increased settling time. Increasing the number of neurons or the number of allowed spikes per neuron will improve the error, especially the immediate increase seen in the top panel of fig. 4.2.3. Lastly, the scale of the rate encoding $\mathbf{\Gamma}$ can be tuned to lower the rise time. However this can lead to overshoot, seen in the bottom panel of fig. 4.2.3.

#### Suboptimal B and C

However things change when one of the two is chosen differently.
In order for the control schemes to work the necessary condition $\mathbf{B}^T\mathbf{C}^T \neq 0$[36] has to hold. However the condition has to be understood as $\operatorname{rank}(\mathbf{B}^T\mathbf{C}^T) = \operatorname{rank} C$. Otherwise certain state variables cannot be controlled. These "invisible states are neglected by the network. This limitation is shared between both methods. Reason behind that is that the matrix $\mathbf{B}^T\mathbf{C}^T$ blinds the Voltage on projecting the error properly in eq. (3.33) and eq. (3.44).
If only $\mathbf{B}$ is chosen without full rank while $\mathbf{C}$ is, some state of $\mathbf{x}$ is moving freely, influenced only by the other states and the system $\mathbf{A}$. If on the other hand a specific $\mathbf{C}$ is chosen it is important to check $\mathbf{B}$ such that $\mathbf{B}^T\mathbf{C}^T$ is not more rank deficient than each on their own. If however $\mathbf{B}$ is full rank but $\mathbf{C}$ is not, output $\mathbf{y} = \mathbf{C}\mathbf{x}$ will not reveal the full state of $\mathbf{x}$. So while $\mathbf{y}$ appears to follow its respective trajectory, $\mathbf{x}$ will not follow the reference trajectory, even though the network received the full state as reference in the first place.
Yet since we are restricted to use $\mathbf{B} = I$ due to our learning rule limitations, it is only a

secondary problem.

Prob rewrite this a bit that we are fine with this constraint. Also the exception with higher order systems (they seem to work just fine)

also mention that this was observed with the error free approach making it a more interesting option

the error version was a bit better performing when adding noise. Natural since the direct error term helps out in that regard

### Equivalence of $\hat{\mathbf{x}}$ and $\mathbf{x}$

However the direct error method suffers from another problem. The definition of the $\mathbf{c}$ is based on the fact that the reference trajectory $\hat{\mathbf{x}}$ follows the same dynamics as the controlled system

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{c}. \tag{4.6}$$

This represents an open loop controller without any feedback from the system itself. Now as both $\hat{\mathbf{x}}$ and $\mathbf{x}$ follow the same dynamics, it appears natural that they can be interchanged to allow for feedback. with the actual state present in the network. However in practice plugging this in eq. (3.44) can pose problems to the network, even if $\mathbf{C}$ and $\mathbf{B}$ are chosen ideally.
While they both follow the same dynamics, their values may be different. Yet even with tiny differences, $\mathbf{c}$ can ruin the control completely.

This can be avoided by strictly using $\mathbf{c} = \dot{\hat{\mathbf{x}}} - \mathbf{A}\hat{\mathbf{x}}$, though this would render the network into a open-loop controller with all its associated problems.
Moreover, noise in the reference and its derivative can negatively impact performance

Verify this. Give an example or a proof.

After careful investigation, the problem from the aforementioned supposed equivalence for $\mathbf{c}$.
To illustrate the problem we consider a 1D example.

$\hat{\mathbf{x}} \neq \mathbf{x}$

The second important implementational detail is that concerns the error term in eq. (3.44) in [37]. After investigating the problem it was found to be an issue with the error term. To illustrate this it is useful to consider their basic example of a scalar system

$$\dot{x} = -10x + u \tag{4.7}$$

but only two neurons with weights

$$\begin{aligned} \mathbf{\Omega} &= k \cdot [-1, 1] \quad k > 0 \\ \mathbf{\Gamma} &= [0, 0] \end{aligned} \tag{4.8}$$

and no additional cost terms.
This configuration allows for a straightforward allocation of functions to individual neurons. In the provided illustration, Neuron 2's voltage tracks the error whenever the network output $\hat{x}$ lags behind the reference value $x$. It becomes active if the deviation surpasses

$$V_2 = 1c\,(x - \hat{x}) < \frac{k^2}{2} \tag{4.9}$$

prompting a corrective response to increase the network output. We now consider the error term with $e(t) = x - \hat{x}$

$$\begin{aligned} \dot{\mathbf{V}} &= \mathbf{\Omega}\mathbf{B}^T\mathbf{A}e(t) + ... \\ \dot{\mathbf{V}} &= -10k \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} e(t). \end{aligned} \tag{4.10}$$

Now, let's examine a scenario where the network is falling short of the reference value, leading us to deduce that $e(t) > 0$. Consequently, we can conclude that the error term is conflicting with the intended definition of our voltage by inadvertently increasing the voltage of Neuron 1 in the wrong direction while it is Neuron 2 that should increase its voltage.
Due to the minus sign obtained from $\mathbf{A} = -10$, the respective projections of the error by $\mathbf{\Omega}$ gets flipped leading to an exponential increase in voltage. If $\mathbf{c}$ is calculated from the reference trajectory and its derivative, this error is compensated for. However by the erroneous substitution this is no longer the case and the error increases while $\mathbf{c}$ does not compensate.

The problem is that the reference $\mathbf{x}$ changes on scales that the network cannot resolve. These small changes are amplified by the multiplication of $\mathbf{A}$ in eq. (3.36) allow to balance the error term. The network state $\hat{\mathbf{x}}$ is not accurate enough to resolve this as it's network activity is zero when the simulation is starting. One way to resolve this issue is to artificially set the error to zero if $\|\mathbf{A}\| << \|x\|$. This is possible since in most problems the error term only contributes a negligible amount to the total voltage change. The biggest contribution is from $\mathbf{c}$ after which the error is small enough to be ignored directly.
In the error free formulation, the above mentioned issues are no problem. Since there is no direct error in the network, the network state $\hat{\mathbf{x}}$ can be substituted instead of the reference without loss of performance.

### 4.2.3 Direct error/Feed-Forward

The most important insight of the approach in eqs. (3.36) and (3.42) is that the signal $\mathbf{c}(t)$ is the principal source of control in the system. Moreover it is explicitly calculated from the system $\mathbf{A}$ as well as the information of the reference signal and its derivative. If one considers the output feedback once more and where one does not have access to the target state $\mathbf{x}$ but only its output $\mathbf{y} = \mathbf{C}\mathbf{x}$, both derivations break down. While the direct error approach of section 3.4.4 requires the knowledge of the complete state vector, it makes the incorporation of output feedback superfluous. In the error-free derivation, $\mathbf{c}$ is directly calculated from the desired state $\mathbf{x}$ without considering $\mathbf{C}$.
Instead the feed-forward input $\mathbf{c}$ would need to be defined in terms of $\mathbf{y}$ and $\mathbf{C}\mathbf{A}\hat{\mathbf{x}}$. This would eliminate the error term outright as it would now be implicitly part of $\mathbf{c}$.

The derivation would be similar

$$
\begin{aligned}
\mathbf{V} &= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\mathbf{y}-\hat{\mathbf{y}}\right)-\lambda_d\mu\mathbf{r}\\
\dot{\mathbf{V}} &= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\dot{\mathbf{y}}-\dot{\hat{\mathbf{y}}}\right)-\lambda_d\mu\dot{\mathbf{r}}\\
&= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\dot{\mathbf{y}}-\mathbf{C}\dot{\hat{\mathbf{x}}}\right)-\lambda_d\mu\dot{\mathbf{r}}\\
&= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\dot{\mathbf{y}}-\mathbf{C}(\mathbf{A}\hat{\mathbf{x}}+\mathbf{B}\mathbf{u})\right)-\lambda_d\mu\dot{\mathbf{r}}\\
&= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\dot{\mathbf{y}}-\mathbf{C}(\mathbf{A}\hat{\mathbf{x}}+\mathbf{B}\mathbf{u})\right)-\mu\lambda_d\left(-\lambda_d\mathbf{r}+\lambda_d\mathbf{o}\right)\\
&= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\left(\dot{\mathbf{y}}-\mathbf{C}(\mathbf{A}\hat{\mathbf{x}}+\mathbf{B}\left(\frac{1}{\lambda_d}\mathbf{\Gamma}\mathbf{r}+\mathbf{\Omega}\mathbf{o}\right)\right)-\lambda_d^2\mu\left(-\mathbf{r}+\mathbf{o}\right)\\
&= \mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\underbrace{\left(\dot{\mathbf{y}}-\mathbf{C}\mathbf{A}\hat{\mathbf{x}}\right)}_{\mathbf{c}}+\left(\frac{1}{\lambda_d}\mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\mathbf{C}\mathbf{B}\mathbf{\Gamma}+\mu\lambda_d^2\mathbf{I}\right)\mathbf{r}\\
&\qquad -\left(\mathbf{\Omega}^T\mathbf{B}^T\mathbf{C}^T\mathbf{C}\mathbf{B}\mathbf{\Omega}-\mu\lambda_d^2\mathbf{I}\right)\mathbf{o}
\end{aligned}
$$

with the exception that is now the difference of the reference output and the network output instead of the the state. Otherwise the equation is identical to the original derivation.

ld I keep
n? If so
e should I
t?

this is messy...and incoherent

All this supports the idea that the feed-forward inputs are the governing force in the network's performance. Furthermore by the removal of the trajectory and only the use of its derivative show more the characteristics of an open loop controller.

at really

The benefit of this approach is that given that using this approach we do not need to know the state in order to track any trajectories but just the output or more specifically the derivative of the state output $y$.

### 4.2.4 Limitations

prob noise in the inputs? noise in the network B'C' = 0 bereits klar. Ist aber staerker als gedacht. "Directly actuated". if there is a limit of how many spikes per time interval the magnitude is bound size?? how to find a large controllable system? inexact derivatives?

## 4.3 Results on the learning

To give a complete examination of results, the rigorous testing required to demonstrate them is not possible. Therefore we restrict our testing to select model problems and only demonstrate a selection of interesting results. Meanwhile testing has been conducted with more and different settings than presented here which will be mentioned occasionally for the purpose of completeness.
For our result evaluations we give a simple test scenario, with which we then evaluate the learning performance for slow and fast learning and combined learning separately. Slow learning alone is skipped as [15] expect to use it in conjunction to the fast learning. Furthermore, while testing it has been observed that the reliability and performance lack behind when only training is trained.
For the combined training we set out to find the determining source of error in overall performance. To access we look into different metrics to determine the progress of learning and highlight key parameters for the success of learning the given system, e.g hyper-parameters or learning input sequences.
Lastly we describe the limits of this methods found in different configurations where the learning breaks down.

### 4.3.1 Testing method

#### Model problem

As the first problem we consider a simple 2D toy example given by a damped oscillator of the form

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & -10 \end{bmatrix} \mathbf{x} + \mathbf{c}(t). \tag{4.11}$$

Since we are only concerned with the learning and neglect the control at this point we give a predefined input $\mathbf{c}(t)$ set as a single square impulse. With this trajectory we can observe the systems response to a sudden jump as well as the evolution of the system in the absence of external input. This allows us to see the system's performance in both scenarios of the control problem later on.

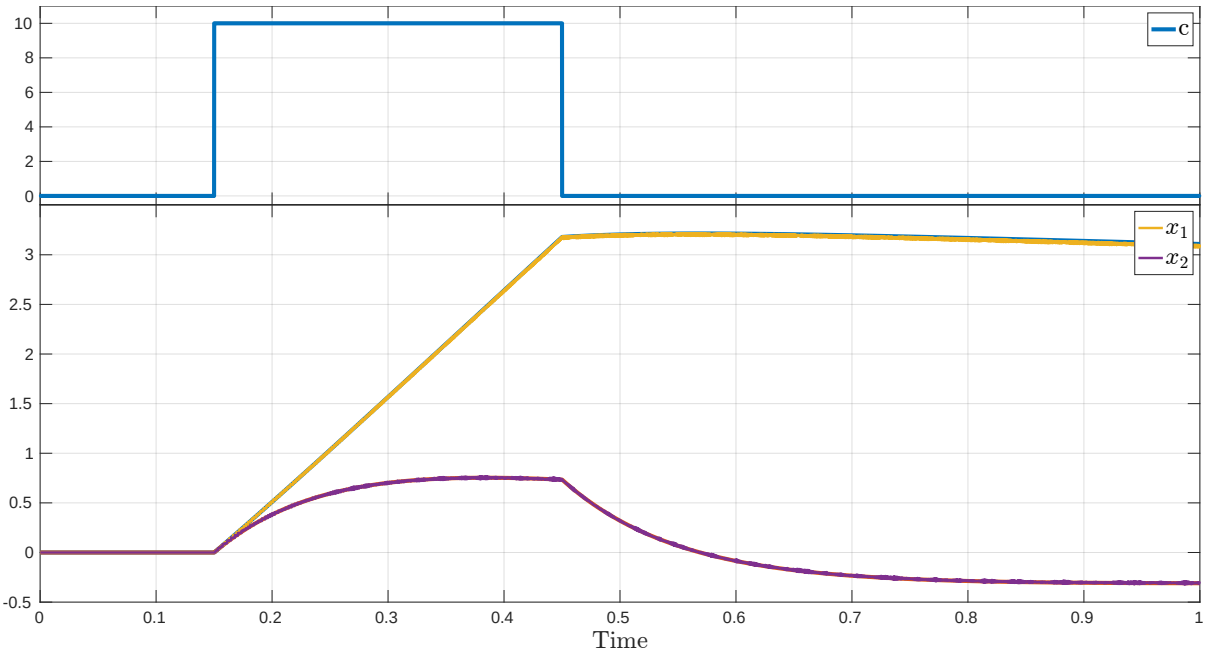



Figure 4.3.1: Simulation of our toy example with the given external input c. The network simulation perfectly overlaps with the analytical solution. Maximum error in this example is $< 0.04$.

### Testing

To measure the learning performance, we consider different metrics in order to get a complete picture.

Due to the analytical solution, we are in the fortunate position to compare the learning values to the optimal values. Yet different measures of performance can be evaluated. The most important and meaningful measure to consider is the network error after learning in a simulation.

As a reference, we plot the optimal solution of our toy example here. As can be seen from fig. 4.3.1 the network perfectly follows simulates the system's behaviour.

However also other measurements can be obtained to assess the networks learning. Alternatively, one can consider the convergence of the matrix parameters to their optimal values. Therefore, we measure the maximum relative deviation from the optimal value as

$$\Delta^{\text{rel}}\mathbf{W}_{ij} = \left| 1 - \frac{\mathbf{W}_{ij}^{\text{learned}}}{\mathbf{W}_{ij}^{\text{ex}}} \right| \tag{4.12}$$

for each parameter. From there we can measure the maximum relative deviation or compute the sum of of deviations to access the parameter convergence.

Alternatively, since we are working on matrices, we look at the convergence of

eigenvalues.
Knowing the optimal values to

$$\begin{aligned} \mathbf{W}^f &= -\left(\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\mathbf{I}\right) \\ \mathbf{W}^s &= \mathbf{\Gamma}^T\left(\mathbf{A} + \lambda_d\mathbf{I}\right)\mathbf{\Gamma} \end{aligned} \tag{4.13}$$

the eigenvalues can be computed.
For $\mathbf{W}^f$ we only need to look at the eigenvalues of $\mathbf{\Gamma}^T\mathbf{\Gamma}$ and shift them by $\mu$. $\mathbf{\Gamma}^T\mathbf{\Gamma}$ itself only has $J$ non-zero eigenvalues $\text{rank}(\mathbf{\Gamma}^T\mathbf{\Gamma}) = J$ since $\text{rank}\left(\mathbf{\Gamma}\right) = J$.
For $\mathbf{W}^s$, only J eigenvalues are nonzero since again the $\text{rank}(\mathbf{\Gamma}) = J$.
Thus, we access the learning progress by measuring the distance of the learned eigenvalues compared to the true eigenvalues.

### 4.3.2 Only learning of $\mathbf{W}^f$

In order to evaluate the learning performance individually, we first consider each learning algorithm separately. Later, we use both to access the overall performance.
This also allows us to see the effects of changing learning parameters individually. To isolate the learning of the fast weights, we set the learning rate for the slow weights to zero and initialize $\mathbf{W}^s$ to the optimal values from the analytic solution given in section 3.3. This provides optimal conditions for the learning algorithm.
We start first by looking at the convergence over learning before looking at errors in order to see whether any of the aforementioned measurements indeed give a meaningful insight into the networks performance.

The learning performance of $\mathbf{W}^f$ is seen in fig. 4.3.2. As can be seen, the two non-zero eigenvalues of the matrix converge close to the optimal values which are reached after $\approx 130$ epochs and keep hovering around. We also see that the largest relative deviation is following the same trend in the beginning, but instead of settling down at $\approx 130$ it starts jumping around. The plot only prints the absolute relative deviation, it is interesting to check the signs during learning. While the error decreases from $-1$, the starting value if $\mathbf{W}^f$ is initialized to zero, the jumping around after epoch $\approx 130$ only jumps in the positive direction.
Before the jumps the learning reaches begins, the learning algorithm manages to get

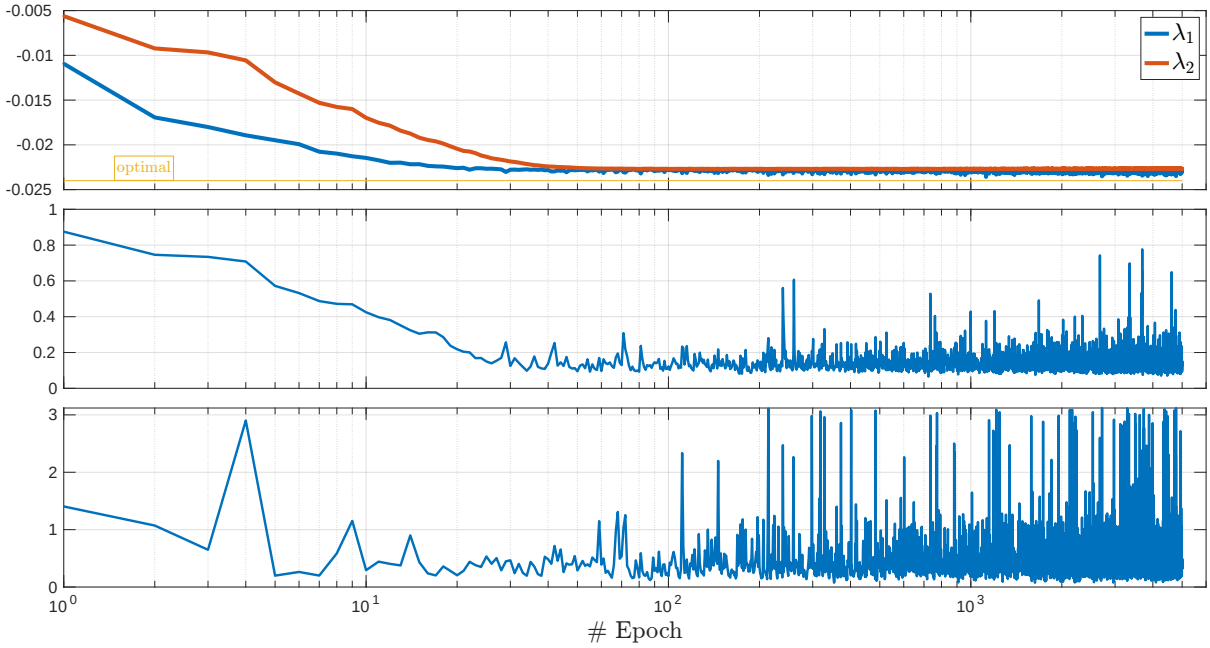


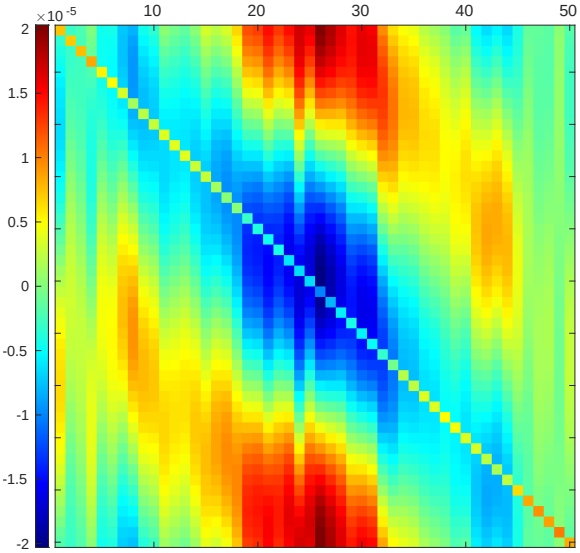

Figure 4.3.2: Learning of $\mathbf{W}^f$ matrix and the end of the epoch. Top panel shows the movement of the two non-zero eigenvalues of $\mathbf{W}^f$. The middle panel shows the largest relative deviation of $\mathbf{W}^f$ calculated from eq. (4.12). The bottom panel shows the maximal error during simulation between the learned and the optimal matrix.

the largest deviation less then $9.5\%$ away from the optimal value. Using this set of learned parameters the maximum absolute error during that simulation was $0.1187$ which corresponds to a relative error of $< 4\%$.

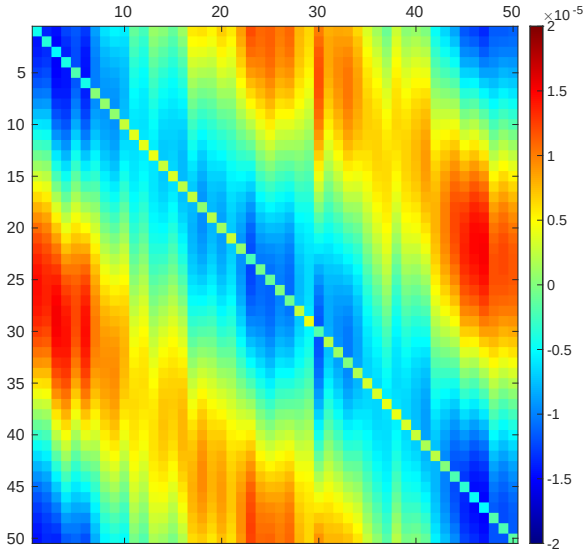
The overall best performance according to our measures was reached in epoch 724 but because of the rapid fluctuations it can be assumed that the parameter set was found randomly. Interestingly did this not also give the best error results but an relative error of more than $10\%$. The best overall performance was in epoch 1630 with an relative error of $< 3\%$. The largest relative deviation of the matrix in that epoch was $\approx 12\%$ from the optimal value.

Visualizing the matrices' differences from the optimal weights of both of these epochs in fig. 4.3.3 it is apparent to see that the columns $i = [1, \dots, 15] \cup [42, \dots, 50]$ are learned slightly more accurately. These correspond to the reset of voltages of neurons $i$. The reason this is important is that for our test example these neurons are by far the most active. Thus by learning these neurons accurately while other weights are further apart from the optimal value gives better results for our example but worse results when looking at the largest deviation.

A possible reason for this behaviour is that the weights are to much dependent on the input of the learning input sequence that is used for learning in the current epoch. This




(a) $\mathbf{W}^f$ matrix after 724 epochs of learning. Displayed are the parameter differences $\mathbf{W}^f - \mathbf{W}^f_{\text{exact}}$




(b) $\mathbf{W}^f$ matrix after 1640 epochs of learning.

Figure 4.3.3: Matrix' learning progress for the best error performance (a) and best relative deviation (b). Learned was with different smoothed Gaussian noise for each epoch for $10^5$ timesteps.

will be investigated in a later section.

But this also means that neither the eigenvalues nor the largest relative deviation give reliable ways to measure our performance without looking at the simulation error. Also other measures like the sum of all deviations do not give a good account on the learning progress. Moreover, all these measure rely on the information given by the optimal weights, which in general is not available information. From this plot it becomes clear hard to predict the network performance without actually testing it. While the eigenvalues or the largest relative deviation might indicate a good set of parameters the error can still make the whole simulation unusable as the error climbs as high as the original simulation in fig. 4.3.1. Therefore it is either necessary to find a better measure or avoid the oscillating effect by adjusting the learning parameters. Latter will be tested in the next section.

Lastly it is interesting to note that for all tests in which the the results were not completely unusable the the biggest error was after the stimulus $\mathbf{c}$ was absent. Furthermore, the error crept up over time and the maximum error was often found towards the end of the simulation. This hints that the network's accuracy is deteriorating mostly when it is evolving on its own.



Therefore a learning limiting fac that if the

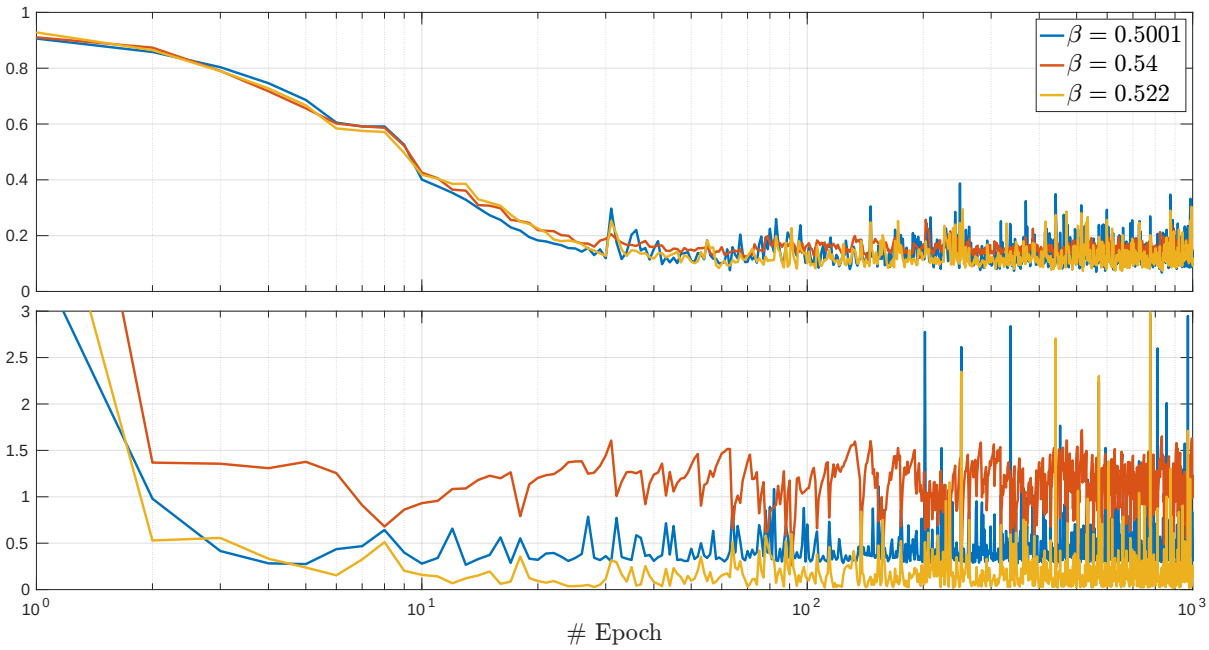



Figure 4.3.4: Learning of toy example eq. (4.11) with different $\beta$. Top panel shows the relative deviation and bottom panel the absolute error.

### Parameter analyses

$\beta$ The prime parameter for the learning of $\mathbf{W}^f$ is the parameter $\beta$. With the same learning and simulation setup as above we test different values of $\beta$. Empirically, we know that values outside the range of $(0.50, 0.54)$ do not perform well enough to compare or do not converge outright. For the learning the identical input sequences were used.

To illustrate the influence of $\beta$ we run the simulation with a value close to $0.50$, close to the upper limit and an the best empirically found value.

In fig. 4.3.4 it becomes visible that with larger $\beta$ the rapid oscillations are reduced. However with $\beta = 0.54$ in the orange line the error is prohibitively high. The eigenvalues are not shown as they all follow a very similar convergence path as seen in fig. 4.3.2.

The best performance was found empirically at $\beta = 0.522$ with comparatively good error measures while having few rapid jumps.

It is interesting to see that for different $\beta$ sudden changes in the error seem to happen after the same training epochs, indicating that the input sequence for this epoch influenced the learning enormously. This can be seen the easiest between epoch $7-180$ where spikes can be observed for each $\beta$. This indicates that the parameter changes

learned in that epoch are to dominant for the whole network and the learning rate must be adjusted.
Unfortunately, none of our previously discussed measures is able to detect these anomalies, leaving us again with no other option that explicit testing of the network in simulation.
To mitigate this either the input sequences need to be modified of the learning rate adjusted. Latter will be the topic of the next section.

**Learning rate** Adjusting the learning rate for $\mathbf{W}^f$ is an easy way to damp the oscillations seen in the previous plots, though coming with an increased computational cost. For comparison, we ran the learning algorithm with different learning rates to compare cost to performance in fig. 4.3.5. As expected reduces the learning rate the oscillations. While the oscillations still occur with the lower learning rate, its magnitude is greatly reduced. With the smallest tested learning rate $\alpha_f = 0.0001$ the noticeable oscillations occur after $\approx 2500$ epochs. The downside to that is just to reach useable results with an relative error of less than $10\%$ learning must continue for more than $1000$ epochs. In addition this is for a small system while we can expect even longer learning times for bigger systems.
On the other hand, the largest learning exhibits the largest error as well as enormous error variability. The learning rate for fig. 4.3.4 was double of the one used in fig. 4.3.5. However it is apparent that already in the first epoch the error reduces significantly.

To combine the benefits of both approaches in **??** we see gradually lowered the the learning rate after each epoch. We start with the previously used learning rate $\alpha_0^f = 0.005$ and gradually lower it to $\alpha_M^f = 0.0001$ over the course of $1000$ epochs.
For the reduction a value $p \in [0, 1]$ was chosen beforehand and the learning rate for the next epoch $m$ calculated as

$$\alpha_m^f = ((1-p)^m + 0.02) \cdot \alpha_0^f, \tag{4.14}$$

essentially allowing for and exponential decay of the learning rate with a backstop to avoid too small learning rates.
In fig. 4.3.6 learning results for different $p$ are displayed. As can be seen, all variations perform more reliable than the any approach before as well as reaching comparable performance with fewer epochs. Furthermore, the fast drop rate of $p = 0.09$ appears to

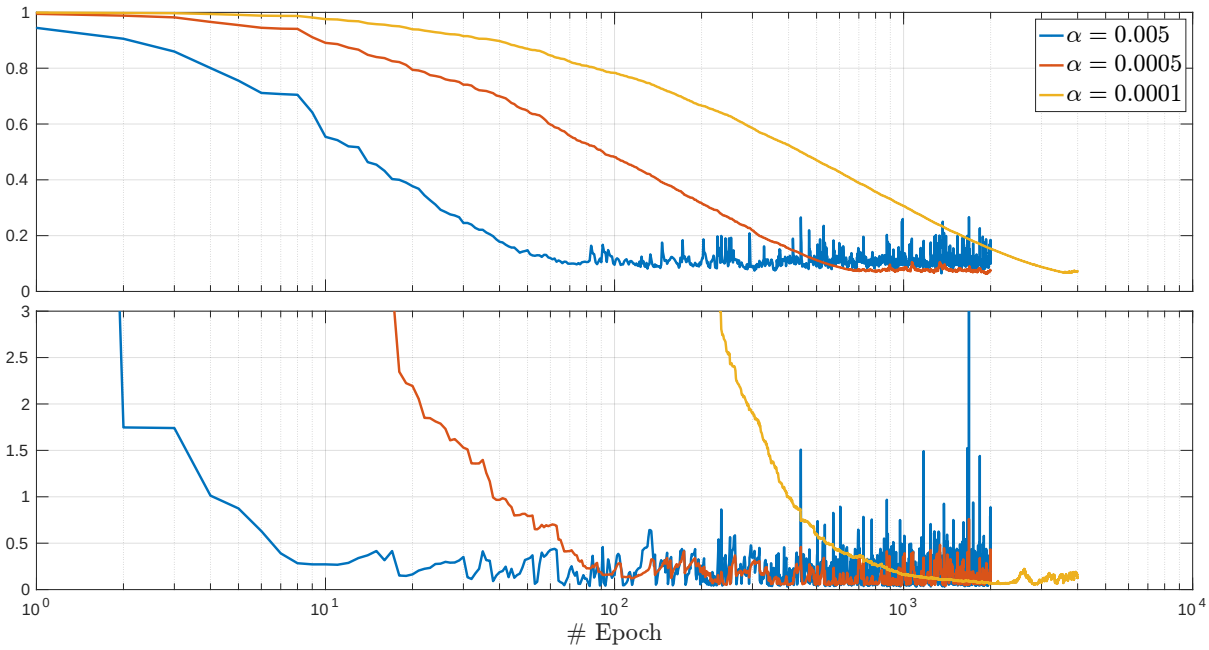



Figure 4.3.5: Learning of toy example eq. (4.11) with different learning rates $\alpha^f$. Top panel shows the relative deviation and bottom panel the absolute error.

give the more consistent results. The improvement of performance is not reflected when looking at the largest deviation, though with the reduced oscillations thanks to the parameter tuning the eigenvalues allow for a more accurate description of the learning status. While there convergence to the true values does not occur, the convergence to a value itself can be seen as the completion of learning. This can be seen in fig. 4.3.6 where learning could be halted when the eigenvalues have converged. However, the convergence depends on the initial learning rate as well as the parameter $p$ and therefore requires hand-tuning to avoid unnecessary long training times or premature stopping. This can be seen again in fig. 4.3.6, in which for $p = 0.09$, the network is performing adequate much before the eigenvalues or the largest deviations would suggest good results can be obtained. Only after $\approx 4000$ epochs the largest deviation appears to level whereas the second non zero eigenvalue still does not appear to have converged yet. The reason the largest deviations do not show appropriate results can be explained when looking at the parameters itself.

For the above example with $p = 0.09$, the weights of $\mathbf{W}^f$ are shown at different times of the learning phase in fig. 4.3.7 rows 1 & 2. The desired outcome after learning is that every parameter is close to zero. With fig. 4.3.7 it becomes apparent that certain columns of $\mathbf{W}^f$ become trained early. The in late stages of training emerging diagonal lines are caused by the optimal value of these matrix parameters being close to zero and therefore making the relative deviation more visible for errors while the whole matrix

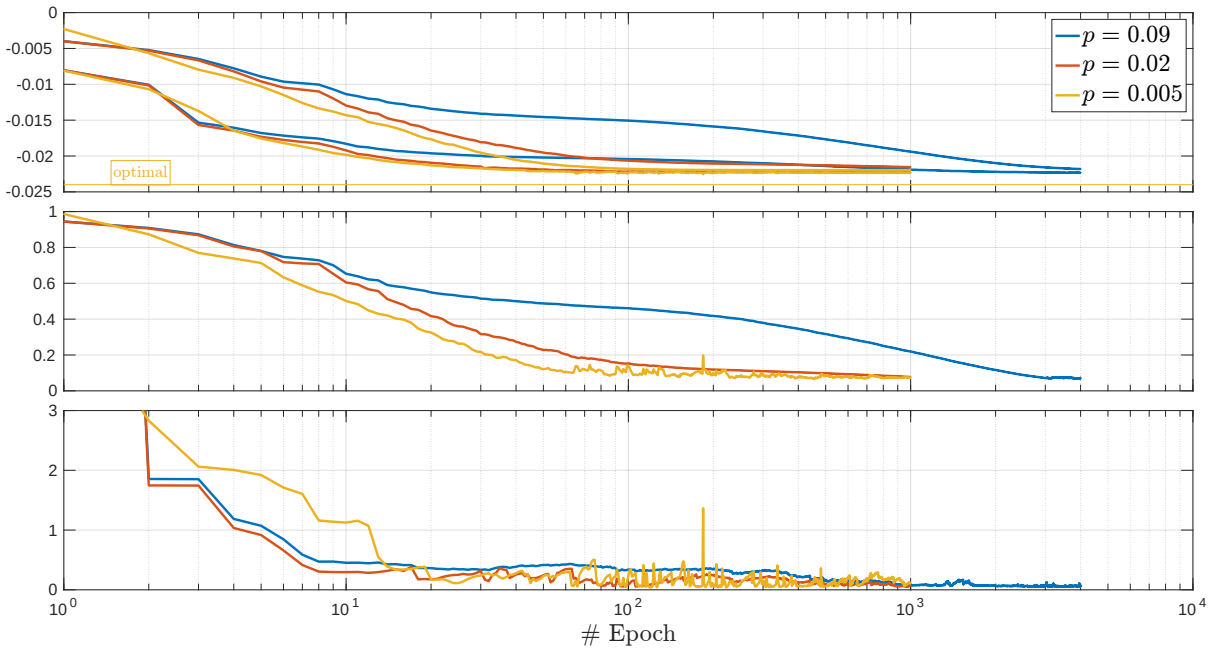



Figure 4.3.6: Learning of toy example eq. (4.11) with different learning rate drops $p$ starting from $\alpha_0^f = 0.005$. Top panel shows the learning rate for each epoch. Middle panel the relative deviation and bottom panel the absolute error.

is converging to the optimum.

The reason behind the surprisingly low error in the early learning stages lies in the choice of our input sequence $\mathbf{c}$. The choice of $\mathbf{c}$ in fig. 4.3.1 only causes the neurons with quickly learned weights to spike and therefore giving a lower error then expected. The spiking neurons and therefore the error do not yield a complete representation of the training state.

If we instead use a more complex external input sequence $\mathbf{c}(t)$ that ideally forces all neurons to spike we obtain a different picture seen in fig. 4.3.8. Note that this evaluation is on the same data used for fig. 4.3.6. However with fig. 4.3.8 the error follows both the eigenvalues as well as the largest relative deviation. It also becomes apparent that the lack of convergence seen in fig. 4.3.7 contribute two orders of magnitude of the error compared to the previous case. Moreover judging by the error plots the choice of $p = 0.005$ vastly outperforms the previous pick of $p = 0.09$. With the lower drop rate $p$ imbalances that dominate the complete learning like in fig. 4.3.7 can be corrected faster as the learning rate has not dropped significantly yet, seen by the row 3 of fig. 4.3.7.

Lastly, the choice of learning input sequences can be questioned to avoid the imbalances in early stage learning. So far for all previous examples input sequences were smoothed Gaussian white noise. The role of the input sequence will be considered

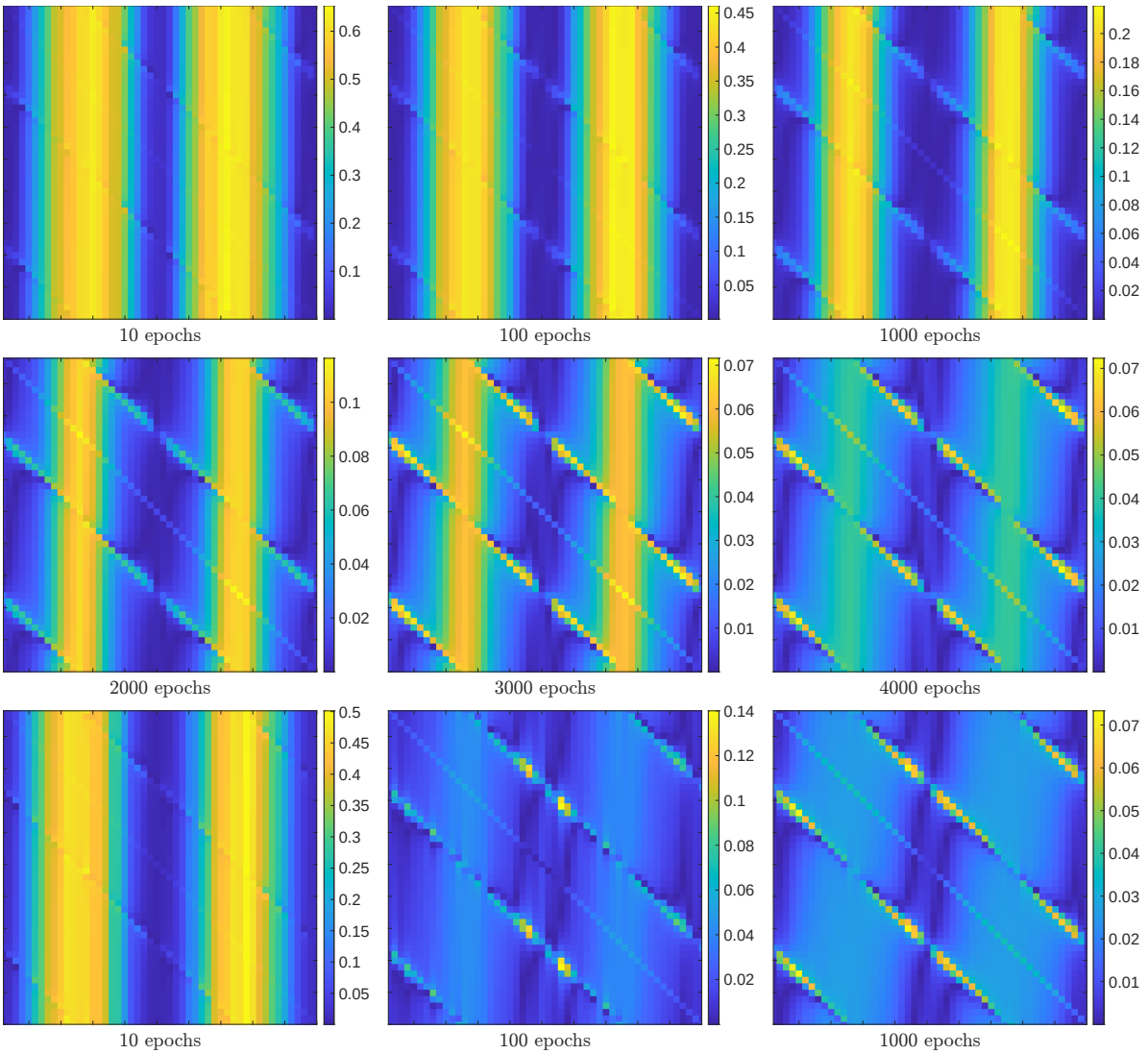



Figure 4.3.7: Relative deviation for matrix weights at different times of learning. Note the different colour scale. The top 2 rows display the learning of $\mathbf{W}^f$ with $p = 0.09$. Bottom row for $p = 0.005$. For training the same input sequences have been used.

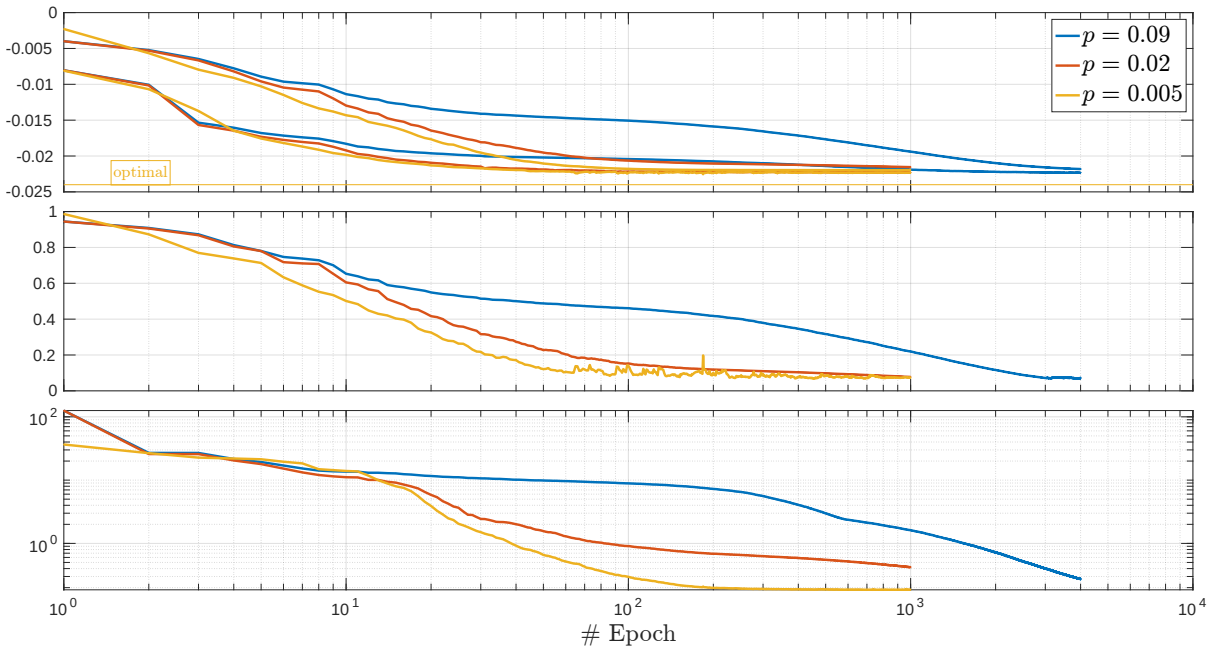



Figure 4.3.8: Evaluation of training progress using a oscillating input $\mathbf{c}(t) = [c_1(t), c_1'(t)]^T$ with $c_1(t) = (1 - e^{-4t})\sin(3\pi t)$ with different drop rates $p$. Top panel shows the non-zero eigenvalues. The middle panel the largest relative deviation and the bottom panel the maximum error during simulation in log-log scale.

when both matrices are trained.

Ein beispiel mit groesse system und mit schlech bedingunge

All in all, good approximations of $\mathbf{W}^f$ can be learning with the given learning rule, yet the hyper-parameters $p, \alpha^f$ as well as learning period are sensitive to the problem at hand and therefore require hand tuning. Moreover, the shown learning results were obtained with ideal learning conditions.

Learning networks with more neurons or random initialization of $\mathbf{W}^f$ complicates the learning phase however good learning results can still be achieved, be it with an increase of learning time.

When accessing the state of training it is crucial to choose appropriate test sequences to reflect different sources of error. This means to force all neurons to spike as well as cutting external input such that the network evolves by itself. Former examines of matrix weights have been learned for all neurons while the latter highlights small errors over time that do not become apparent when external input is dominating. The small errors incurred from suboptimal resets accumulate over time if no external input is present and $\mathbf{\Gamma}, \mathbf{W}^s$ are chosen optimally. Therefore a combination of both presented sequences should be used.

Lastly, our measures to access the network's learning progress rely on the knowledge of

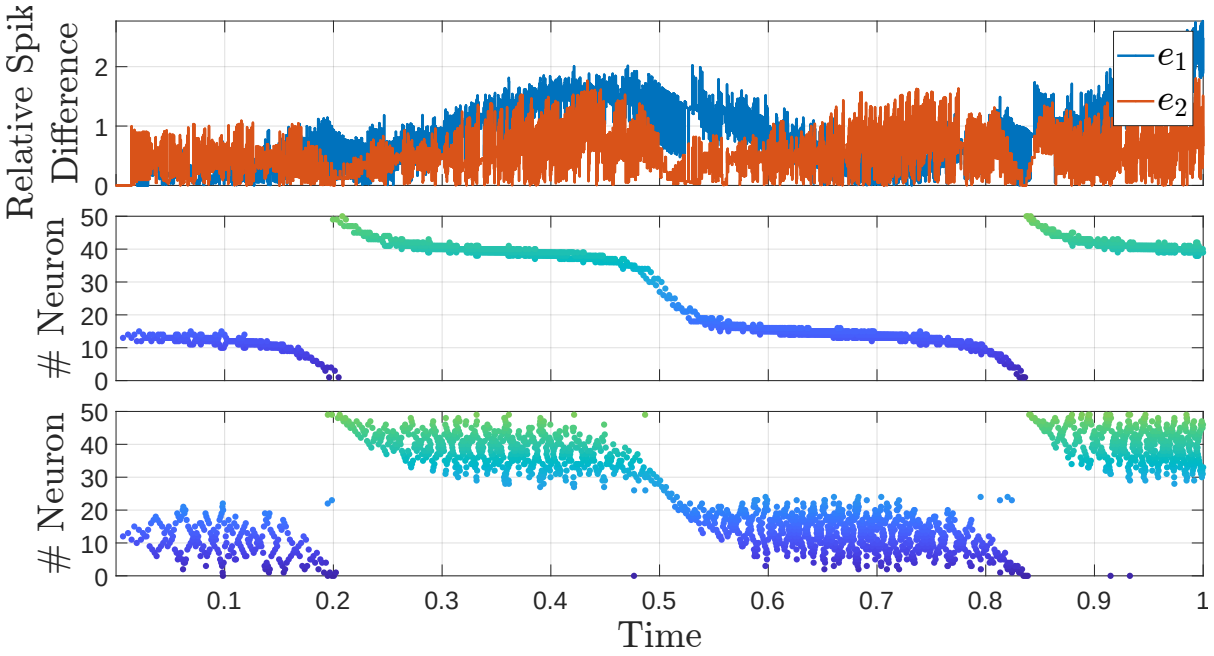



Figure 4.3.9: Top panel: Relative Spike Difference between the network with $L_2$ costs added to $\mathbf{W}^f$ after learning and no addition. Error is given in multiples of decoder norm $\|\mathbf{\Gamma}_i\|_2$. Middle panel: Raster spiking plot of $N = 50$ neurons after simulation without the $L_2$ costs added afterwards. Bottom panel: Raster plot with costs added. Both matrices were trained for 1000 epochs with identical settings and received a scaled sin wave as external stimulus $\mathbf{c}$. For these simulations, noise terms during training and simulation have been disabled.

the optimal matrices or the repeated expensive computation of eigenvalues. Without online measures frequent testing is a necessity to avoid extensive learning periods.

### Spiking behaviour

So far, we have not touched on the spiking behaviour. For the previous examples, the learning rule of eq. (3.50). Said learning rule does not incorporate $L_2$ costs in encourage distributed spiking. When looking at raster plot in the middle panel of fig. 4.3.9 this disadvantage becomes apparent. Spiking is shared among 4 neurons at most.

Redo the plot such that the y axis is not cut

This is biologically implausible. When using the adapted learning rule of eq. (3.51) other problems arise.

As noted in the supplementary of [17], this particular learning rule does not learn the optimal weights but the optimal structure. As such it is therefore only a qualitative

learning of the $\mathbf{W}^f$. The scaling factor is now known a priori. The different scale of $\mathbf{W}^f$ is detrimental to the networks convergence. In test no parameter set was found that mitigated this.
It is therefore mandatory to allow the Feed-Forward $\mathbf{\Gamma}$ weights to adjust i.e they are learned as well. With Feed-Forwards also scaling over the course of training allows the different scales to balance out.
Learning of $\mathbf{\Gamma}$ was adopted from [17] which allowed the network to converge.
After convergence was achieved, the hyper-parameter scaling factors for $\mathbf{W}^f$ and $\mathbf{\Gamma}$ were tuned.
Goal of the tuning was to empirically find a parameter set of scaling factors s.t. the scale of $\mathbf{\Gamma}$ remains unchanged throughout learning and transfer this to the learning with fixed $\mathbf{\Gamma}$. However this again led to unusably erroneous results or failure in convergence outright.
The error itself in [17] is not calculated using the learned decoder $\mathbf{\Gamma}$ (noted as $\mathbf{F}$) but the mathematically optimal decoder

$$\mathbf{F}^T = \min_{\mathbf{F}^T} \|\hat{\mathbf{x}} - \mathbf{x}\| = \min_{\mathbf{F}^T} \|\hat{\mathbf{x}} - \mathbf{F}^T\mathbf{r}\| \tag{4.15}$$

using conventional methods of regression. While the learned decoding weights do converge, they lack behind the presented precision my orders or magnitude.
It is important to note that all the above testing was conducted only on the original example with $A = -\lambda_d \mathbf{I} \longrightarrow \mathbf{W}^s = \mathbf{0}$. With custom $\mathbf{A}$ convergence was not achieve with any parameter set or configuration.
This lack of convergence, the sensitive hand-tuning of the scaling parameters for $\mathbf{W}^f$ and $\mathbf{\Gamma}$ already without the training of $\mathbf{W}^s$, the adoption of the improved learning rule eq. (3.51) was rejected.
Instead, to reestablish a more plausible spiking behaviour we artificially introduce the diagonal term from the analytic solution for $\mathbf{W}^f$. Since the first learning rule learns is approaching the $\mathbf{W}^f = \mathbf{\Gamma}^T\mathbf{\Gamma}$ the addition is the most convenient way to reach our target ideal matrix $\mathbf{W}^f = \mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\mathbf{I}$.While this addition is biologically implausible itself

Is it really? Could it be maybe somehow plausible?

we get more plausible spiking behaviour with little to no performance impact in return. The change in spiking behaviour can be seen in fig. 4.3.9 bottom panel.
In comparison to the ideal network, both simulations performed well with an absolute

error less than 0.1 compared the optimum. The dynamics between both networks also differ only marginally. In the top panel the differences in networks dynamics are displayed by the relative difference of spikes. For each spike of neuron $i$, the output signal shifts by $\mathbf{\Gamma}_i$ as it is added to the spike train. From the top panel of fig. 4.3.9 it can be seen that the differences between both simulations were never more then 3 spikes different from each other.

Is this saying smth? Maybe the absolute error is better?

nvalues
gain
esting
use
smaller
avalues
an use
when the
avalues are
hanging
nore the
ox is ok-
.

this since
e origninal
r it is
dy stated
the slow
ing rule
works
erly
the fast
ing in
em

### 4.3.3 Only learning of $\mathbf{W}^s$

### 4.3.4 Combined Learning

For this chapter we consider the combined learning performance of slow weights $\mathbf{W}^f$ and slow weights $\mathbf{W}^s$. We skip the investigation of $\mathbf{W}^s$ alone as the authors of the slow emphasize to learn both matrices simultaneously. Indeed, slow weights struggle to converge without learning of $\mathbf{W}^f$.

For the investigation of the training of $\mathbf{W}^s$ we apply the previously mentioned improvements of $\mathbf{W}^f$ too. We start by looking again at our simple system before investigating parameter and input dependencies.

To begin, we look again at results for eq. (4.11) for reference. We adjust our external input sequence $\mathbf{c}(t)$ to excite each neuron to spike so we have a better picture of whole learning progress of all neurons. Yet we keep a segment in which no input is given such than we can see the network's own evolution over time.

In fig. 4.3.10 the ideal results are displayed again with the given external $\mathbf{c}(t)$ input.

To investigate the learning we will again look at the the measures defined in the previous section. Moreover, we will measure which matrix is more detrimental to the overall performance and therefore should be learned more accurately. Lastly we will conduct a parameter analyses so identify ideal settings. Lastly we will investigate how the training sequences influence the learning behaviour. So far, all learned was done with smoothed Gaussian noise. But so far very little attention went into our choice of learning input.

Starting we first compare the trajectories of the ideal against the learned weights to see where potential discrepancies arise. In fig. 4.3.11 we can observe the networks

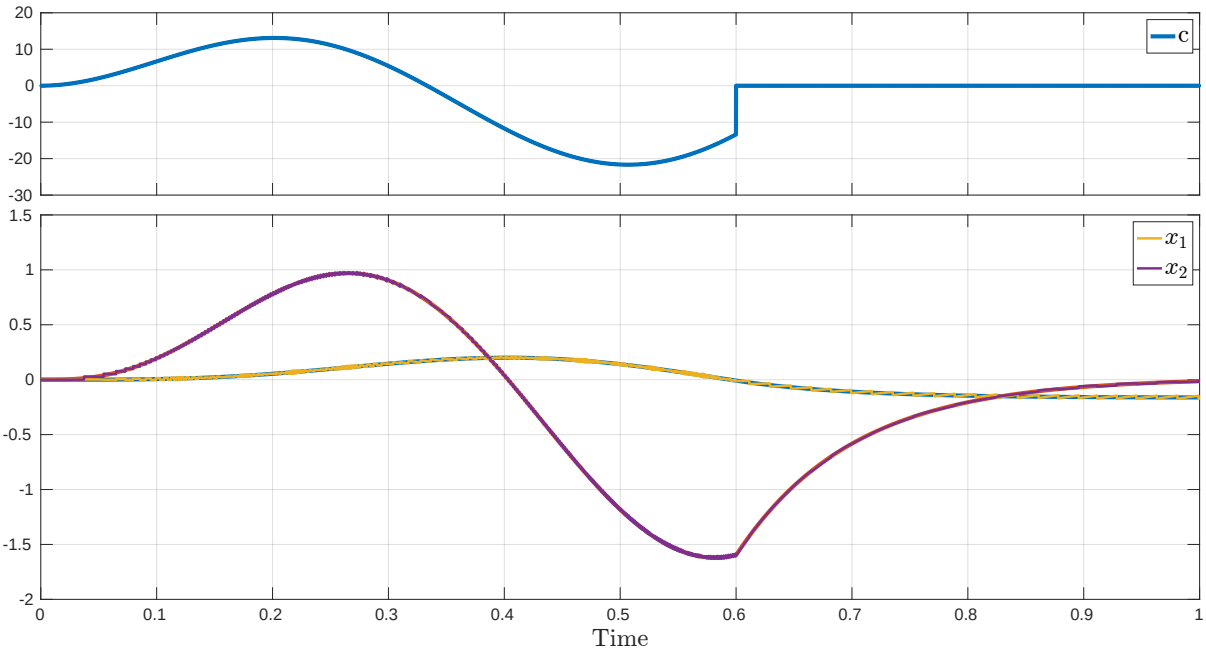



Figure 4.3.10: Exemplary results for our network with ideal weights. In the top panel the external input $\mathbf{c}(t) = [\mathbf{0}, c]^T$. Bottom panel shows the calculated system response using numerical methods and the networks response. The network superimposes the numerical results exactly.

response for learned and ideal weights after training are identical. Moreover the error is significantly better than the training of just $\mathbf{W}^f$. While the error is generally acceptable, it becomes clear that the error sources change when the external input $\mathbf{c}(t)$ is absent and the network evolves by its own. While in the beginning, the network is driven by the external input, neurons fire frequently. Therefore, resets after spike are given by $\mathbf{W}^f$ play a heightened role. Slight errors can cause a delay or pre-firing of a spike compared to the ideal weights. This can be seen in fig. 4.3.11 just before $0.1$s, where the error jumps up after the learned network fired spikes the ideal network. The error jumps back down after the ideal network catches up. However later this discrepancy becomes too large and the error appears somewhat chaotic.
With the absence of the external input the error becomes fine grained again since the repeated spiking of caused by external input disappears. Now the weights $\mathbf{W}^s$ play an important role. Errors in the slow connections can add up slowly and cause wrong neurons to fire.

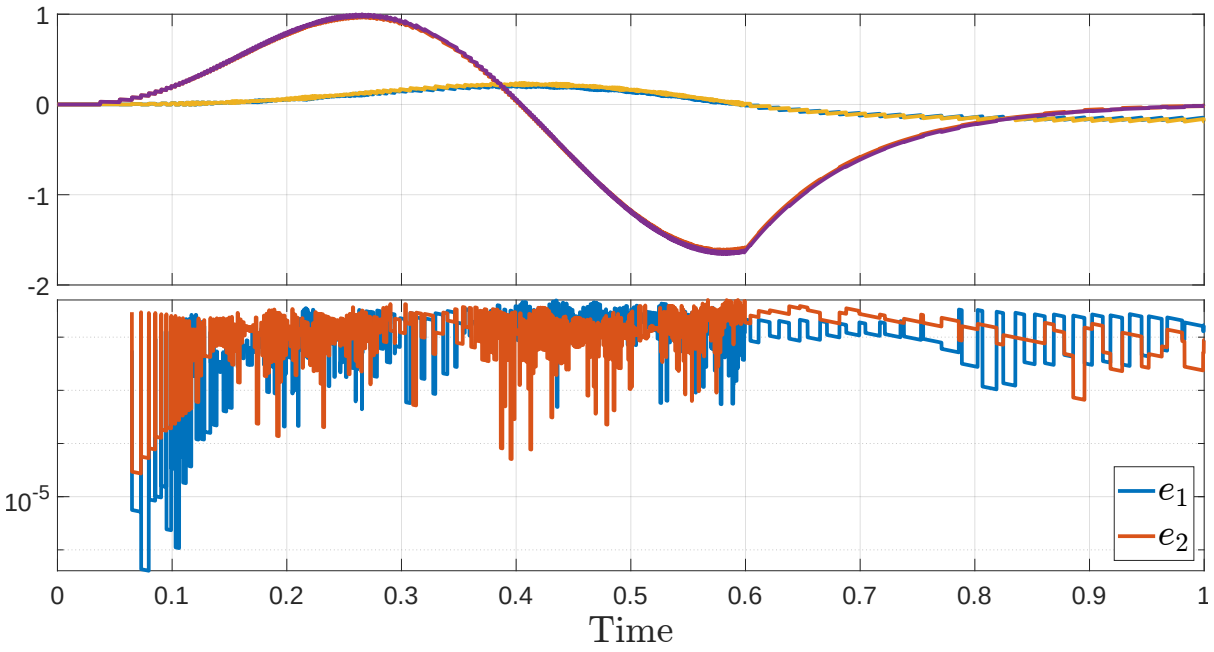



Figure 4.3.11: Computed error between ideal and learned weights. Bottom panel shows the error over time between $x_1$ and $x_2$. Matrices were trained for 4000 epochs.

### Parameter analyses

The following subsections are dedicated to investigate the parameters concerning the learning of $\mathbf{W}^s$ and the learning of the network in general. This includes the error weight $K$, the slow learning rate $\alpha^s$, the learning input sequences to train the network.

**Learning rate** Looking at fig. 4.3.12 the learning rate it appears much larger learning rates can be used for $\mathbf{W}^s$. While all tested learning rates $\alpha^s$ reach the same level of error, the larger $\alpha^s$ the faster this level is reached. Furthermore it becomes clear that the learning of $\mathbf{W}^s$ is related to the accuracy of $\mathbf{W}^f$. Only after deviations in the parameters of $\mathbf{W}^f$ are small the learning of $\mathbf{W}^s$ can reach errors less than 0.1 which corresponds to a relative error of $\approx 10\%$. It is also clear that the learning rate of $\mathbf{W}^s$ does not affect the behaviour of $\mathbf{W}^f$ and therefore can be learned with the same or larger learning rates. This is in opposition to [15] where is is recommended to learn $\mathbf{W}^s$ with a smaller learning rate compared to $\mathbf{W}^f$.
We choose $\alpha = 0.01$ for future evaluations as it marginally improves performance in the error and shows less variability in the rel. deviations. This comes at the cost of training for 100 more epochs before reaching slightly better performance.
To combat the unsteady behaviour in the first panel we again try again to reduce the

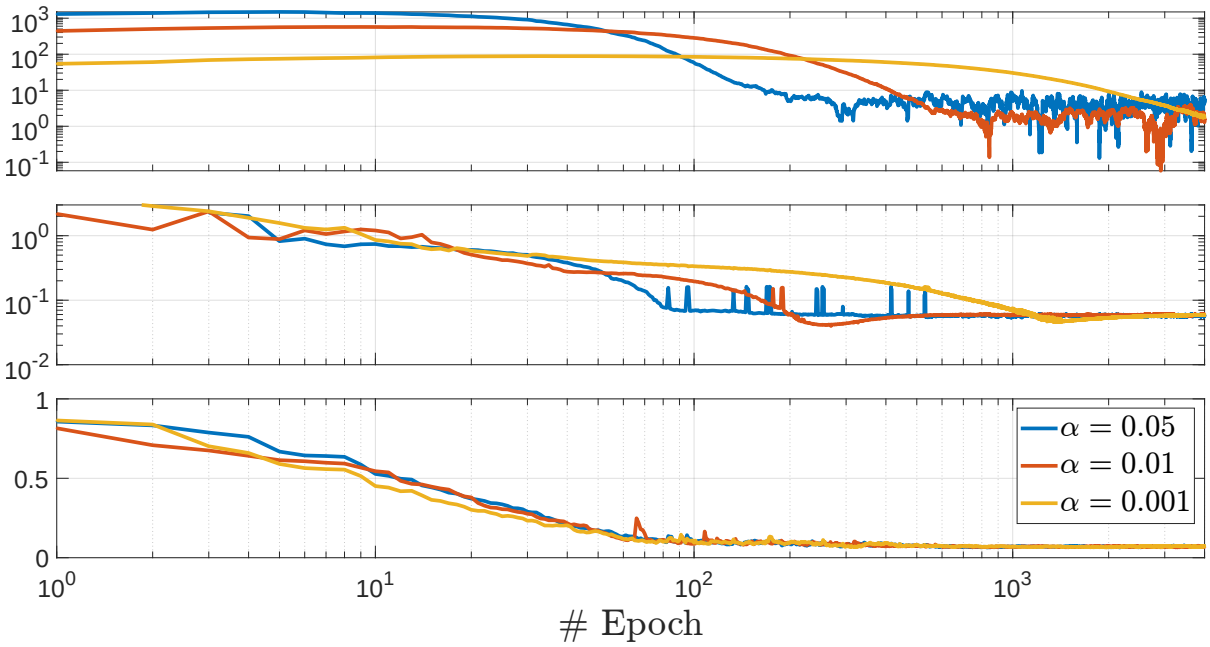



Figure 4.3.12: Top panel showing the largest rel. deviation after each epoch for different learning rates $\alpha^s$. Middle panel shows the maximum error during simulation with the learned $\mathbf{W}^f$ and $\mathbf{W}^s$ in log scale. Bottom panel shows largest rel. deviation for $\mathbf{W}^f$.

learning rate over time.

To do this we again set our learning rate according to eq. (4.14) and test different $p$. In fig. 4.3.13 we see the effect of different values for $p$. As expected was the error in the bottom panel of fig. 4.3.13 not reduced by this addition. Moreover, under a linear scale of the largest rel. deviations are not noticeable and will therefore be omitted for future tests. Lastly, we did not look at the eigenvalues of $\mathbf{W}^s$ yet. Fortunately, the non-zero eigenvalues do allow for a coarse estimate of the learning progress. In all test including fig. 4.3.12 the eigenvalues converged to the optimal values and usable results could be obtained after both eigenvalues converged.

**K** The parameter $K$ describes the influence of the error during training. With increasing $K$ the importance of the error and therefore the teacher grows.
As can be seen in fig. 4.3.14, the parameter $K$ only slightly impacts the network's learning. Also in contrast to the derivation in which $K$ is assumed to be large, the smaller the K the better the network performs. Furthermore, for this example values $K > 100$ did not give acceptable results. For $K > 500$ convergence did not in the $4000$

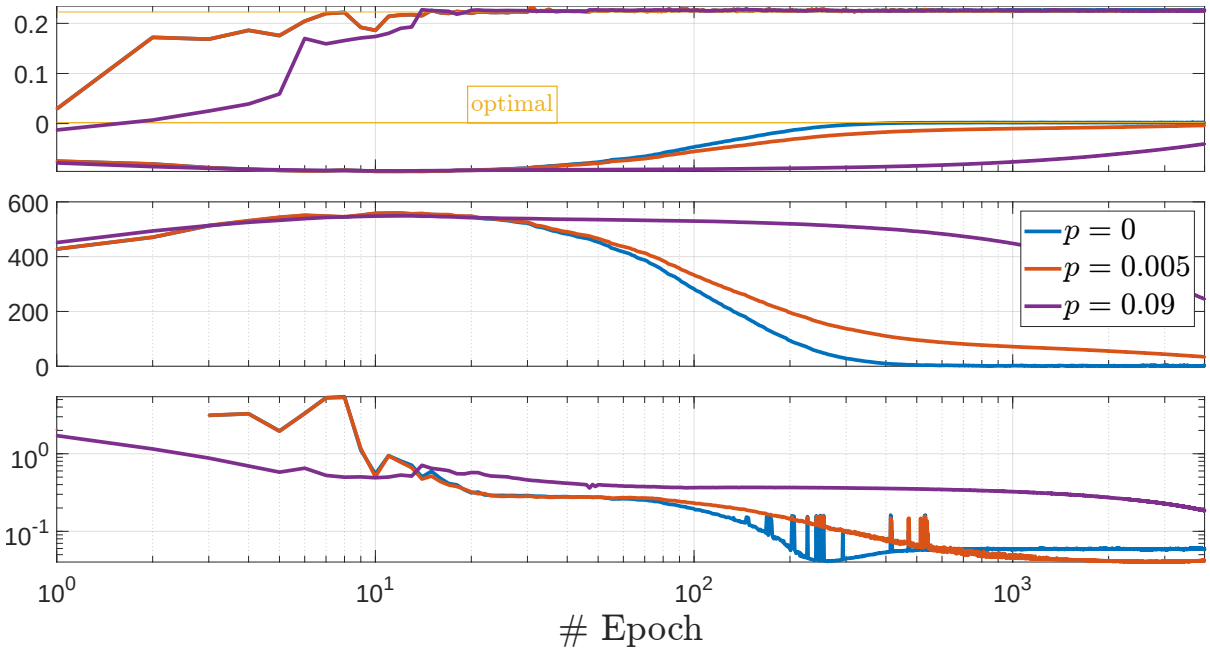



Figure 4.3.13: Top panel showing the two non-zero eigenvalues of $\mathbf{W}^s$ after each epoch for different drop rates $p$ and starting learning rate of $\alpha^s = 0.1$ . Middle panel shows largest rel. deviation for $\mathbf{W}^s$. Bottom panel shows the maximum error during simulation with the learned $\mathbf{W}^f$ and $\mathbf{W}^s$.

epochs of learning.

In the middle panel of fig. 4.3.14 tested values above $50$ ($K = 100, 200, 500, 1000$ not all shown) present a bend in the error after $\approx 1050$ epochs and the error starts increasing again. With larger $K$ the bend is more pronounced and occurs after fewer epochs.

Reason?

On the other hand values $5 > K > 0.5$ all performed almost identical to $K = 1$ shown in the plot.

In the last panel of fig. 4.3.14 we chose $K = 10$ to test the main source of error. For this we ran the simulation with $\mathbf{W}^f$ learned using the optimal $\mathbf{W}^s$ and vice versa. The results are seen in the bottom panel of fig. 4.3.14. Without a good approximation of $\mathbf{W}^f$ the error is more than 2 orders of magnitude worse than the opposite configuration. However this normalizes at around $30$ epochs after which the the error in $\mathbf{W}^s$ dominates. Only after more than $1000$ epochs $\mathbf{W}^s$ is accurate enough such that the small errors in $\mathbf{W}^f$ from learning become relevant once more. However after more than $1000$ epochs the error less than $0.04$ or $\approx 3\%$ relative error.

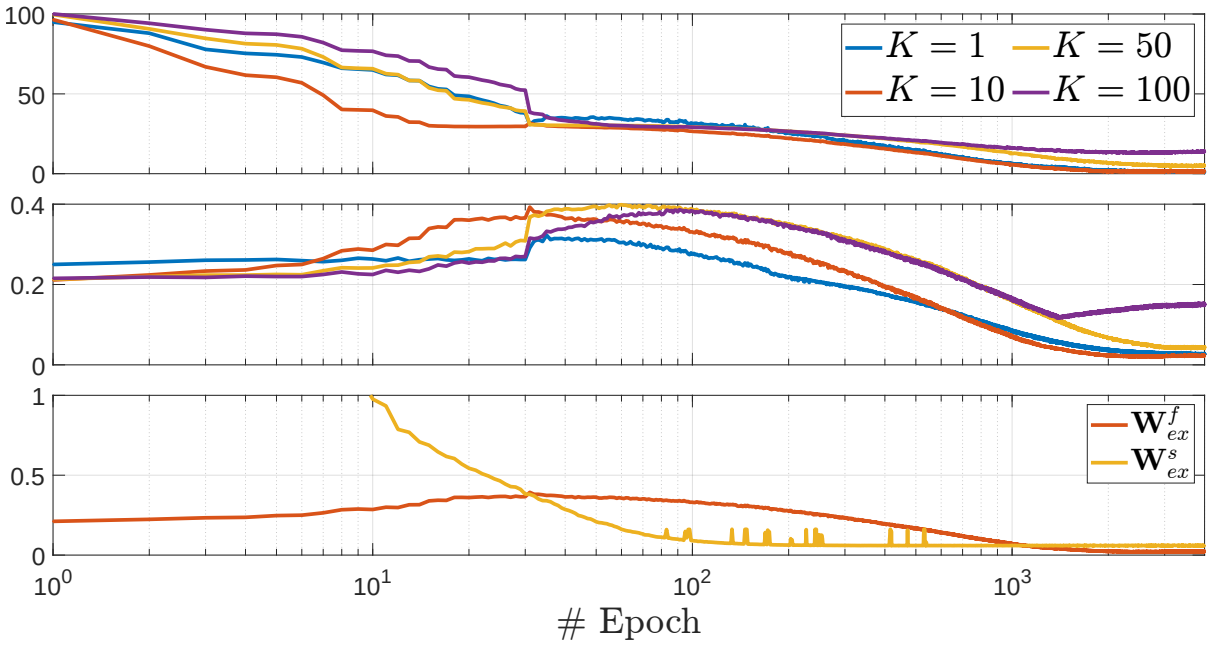



Figure 4.3.14: Top panel shows the rel. deviation for each $K$. Middle panel shows the maximum error during simulation after each epoch using both $\mathbf{W}^f$ and $\mathbf{W}^s$ from training using different $K$. Bottom panel shows the error for $K = 10$ while using either the optimal $\mathbf{W}^f$ or $\mathbf{W}^s$ respectively.

### Input learning sequences

So far we have not touched on the training input. As often seen in the literature

put some references here

, training is performed using smoothed Gaussian noise. However little detail is given on the specifics. Moreover, as speculated above, the choice of input sequence could potentially affect the learning results. Therefore, in the next section we will investigate different patterns of input and potential influence on the results.

In the right training sequence we may find ways to make our network even more accurate.

Firstly we note the general framing of our learning. So far we have used the common smooth Gaussian with an exponential kernel. Already the number of tunable parameters is large when considering different kernels to get smoother or more variable input sequences. However there are also completely different functions that can be tested. For now we compare the currently used Gaussian noise again pure sine waves and individually stimulating each neuron for training.

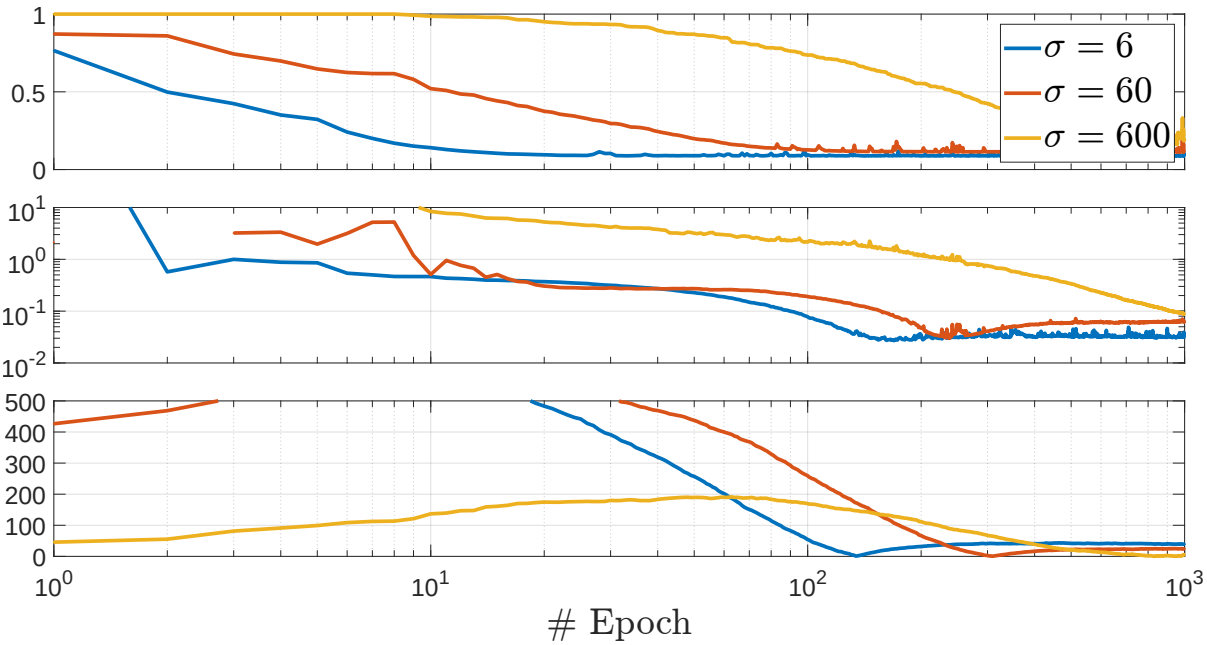



Figure 4.3.15: Illustration of smoothing with $\sigma$ changing in the kernel. F.l.t.r $\sigma = \{6 \quad 60 \quad 600\}$. Plots show only the first 10% of a learning epoch for visibility.

### Gaussian noise and smoothing

integrate the above figure in the text!

The current setup of learning is based on a sequence $\mathbf{v}$ of Gaussian noise that is generated for each epoch. One sequence is 100000 timesteps long and each gets convoluted by a exponential kernel to obtain $\mathbf{c}$. The standard deviation for $w$ was chosen tight to allow for rapid movement of $\mathbf{c}$ allowing for great variety which drives the error and subsequently the learning of $\mathbf{W}^s$.

This can be seen in fig. 4.3.16. For $\sigma = 6$, a tiny smoothing window, the networks shows the best results with an maximum absolute error of $0.03$ which is within one spike difference from the ideal with the given setup. The second best option is for $\sigma = 60$ with double the error of $\sigma = 6$. Also in the convergence of both $\mathbf{W}^f$ and $\mathbf{W}^s$ the least amount of smoothing appears to yield the best results. Though this is heavily depending on the signal that the network is being tested with. Moreover difference in error is reduced when tested with different signals $\mathbf{c}$ to input. With oscillating input similar to what was used in training the network or larger amplitude $c$, the trials with more smoothing reduce the difference in the error almost completely.

re for
rating
ifferent
othing.

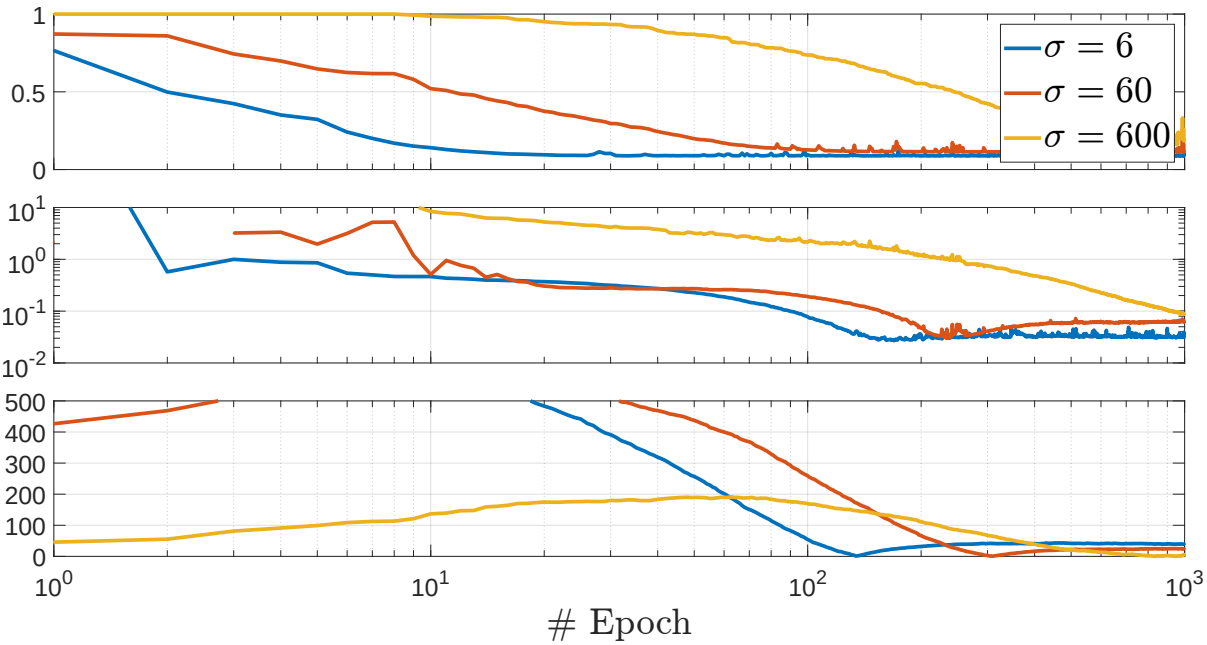



Figure 4.3.16: Learning results for different smoothing grades of $\mathbf{c}$. Smoothing is governed by the std. dev. $\sigma$ of the exponential kernel. Top and bottom panel shows the largest rel. dev. of $\mathbf{W}^f$ and $\mathbf{W}^s$ respectively. Middle panel shows the absolute error. Trained was for 1000 epochs.

**Alternative Learning inputs forms** The importance of sudden movement and therefore the necessity for tiny smoothing becomes apparent when comparatively smooth input sequences are tested. To illustrate this the network is trained with the previous white noise with $\sigma = 60$ and compared to similar magnitude sine and square waves.

To introduce some variability in the sine wave. For each epoch the phase for each input is varied.

The square wave is explicitly structured to enforce firing of 2 specific neurons at a time. As illustrated before, each neuron $i$ projects the error along $\mathbf{\Gamma}_i$ direction and fires when the threshold is reached. To target neuron $i$ directly we project the square wave on each dimension $c_j$ of the input by multiplying it with $\mathbf{\Gamma}_{ij}$. Using this approach neuron $i$ fires the earliest and will be trained. During the off cycle of the square wave the same happens with the neuron geometrically opposing $i$.

To verify this we can look at the spiking behaviour during training. In fig. 4.3.17 we can see the cumulative spiking of each neuron during training for the Gaussian noise, sine and square wave. The square wave clearly shows our desired pattern in which for each epoch a new pair of neurons is trained, illustrated by the diagonal edges. From this it can also be seen that the test used 50 neurons. At the mark of 50 epochs a

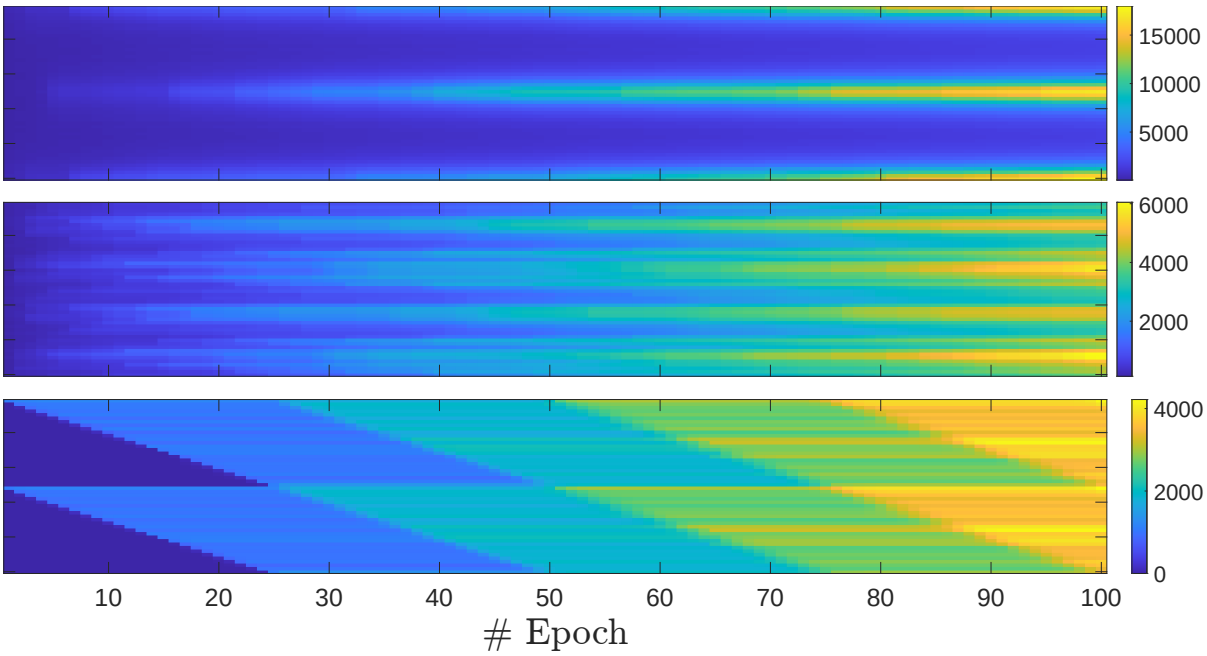



Figure 4.3.17: Cumulative spikes for each neuron over training. Top: Gaussian noise, Middle: Sine wave, Bottom: Square wave. Only a segment of the entire training process is shown. Note the different colour scales.

rotation is completed indicated by the first diagonal line ending there. Furthermore after 25 epochs each pair of neurons has been trained once, indicated by the 2 diagonal line starting from the top left corner entering the section of geometrically opposing neurons.

Surprisingly the spiking behaviour with the sine wave and the Gaussian noise exhibit an aggregation of spikes only on certain neurons. From the colour grading it becomes apparent that certain neurons spike more often during training than others. While this is caused by the lack of regularization and subsequent distribution to spikes in our learning scheme due to aforementioned problems. Solely from judging fig. 4.3.17 it would be natural to guess that the learning using square waves would perform the best thanks to the even distribution of spikes and therefore even learning. However this would be wrong as can be seen in fig. 4.3.18. While the trajectory for $x_2$ follows the main numerically calculated solution in blue, $x_1$ does not follow the expected trajectory at all. This misbehaviour is caused by the input sequences overshooting the scale of the parameters of $\mathbf{W}^s$. Similarly this occurred as well when using heavily smoothed noise. The sine and square wave created, similar to the heavily smoothed, stretches of repetitive stimulus appears to cause the corresponding weights for spiking neurons to be oversaturated. Lowering the input amplitude did improve performance at the cost of longer training times. With the reduced input the convergence of $\mathbf{W}^f$ is slowed

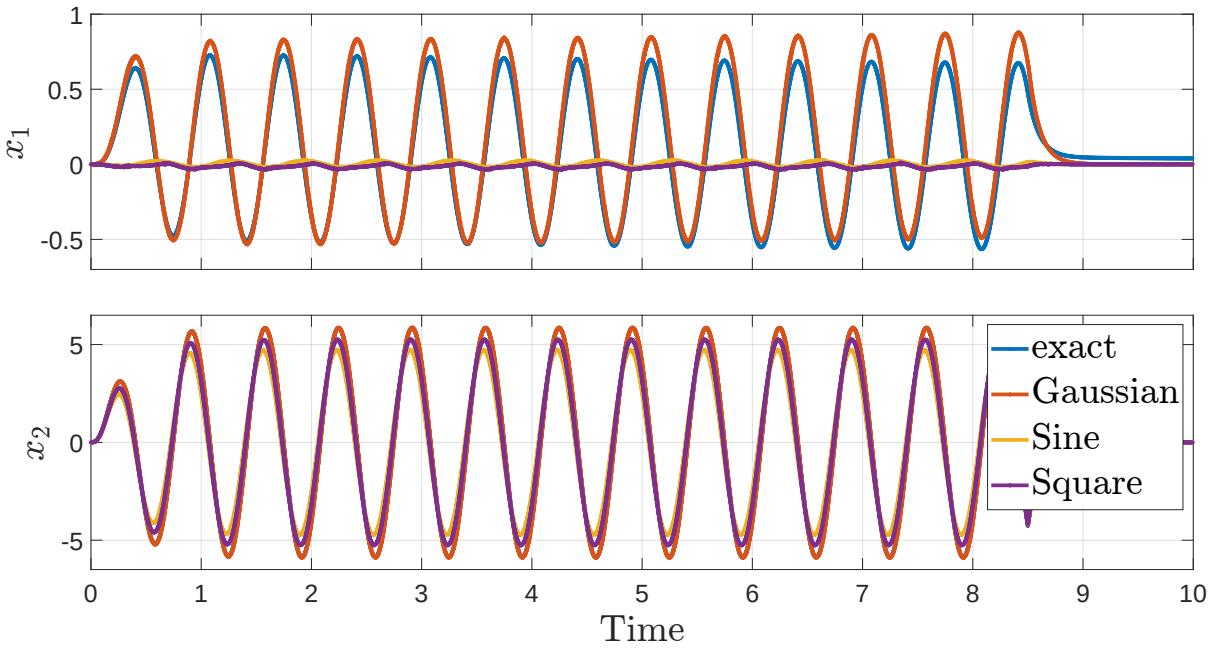



Figure 4.3.18: Network responses for each training sequence type with a oscillating input $\mathbf{c}$.

down by the reduced number of spikes necessary. Furthermore $\mathbf{W}^s$ learning rule is directly depending on the error and the rate, therefore depending on the state vector $\mathbf{x}$ that is once more dependent on the input $\mathbf{c}$.

**Spike firing** das ungleiche feuern abhanging von A die input sequenzen das manche neuronen die anderen das feuern stehlen warum die anderen sequenzen scheisse sind (fuer Ws) Anpassen der Lernsequenz so dass alle neuronen gleich oft feuern Finde die anzahl an spikes die man braucht um ausreichend gelernt zu haben

In training, each neuron is fired ideally the same number of times to facilitate evenly distributed training progress. However, when looking at the spike numbers during training this is not the case. In fact, the learned system $\mathbf{A}$ is crucial for the spike distribution. This can be seen in fig. 4.3.19, where the cumulative sum of spikes are added for each epoch on 2 different systems $\mathbf{A}$. The crucial difference is that our model problem's eigenvalues are further apart. Generally it was observed that the distance between eigenvalues relative to their absolute values was the determining factor.
The reason of lies in $\mathbf{W}^s$. If $\mathbf{A}$ exhibits a large difference in its eigenvalues this is reflected in $\mathbf{W}^s$. During training, neuron voltages increase more for neurons projecting in the dimension of the large eigenvalue, causing it to spike. This can be seen again in fig. 4.3.19, where the neurons with the highest spike count

using less
smooth inp
helps distri
spikes bette

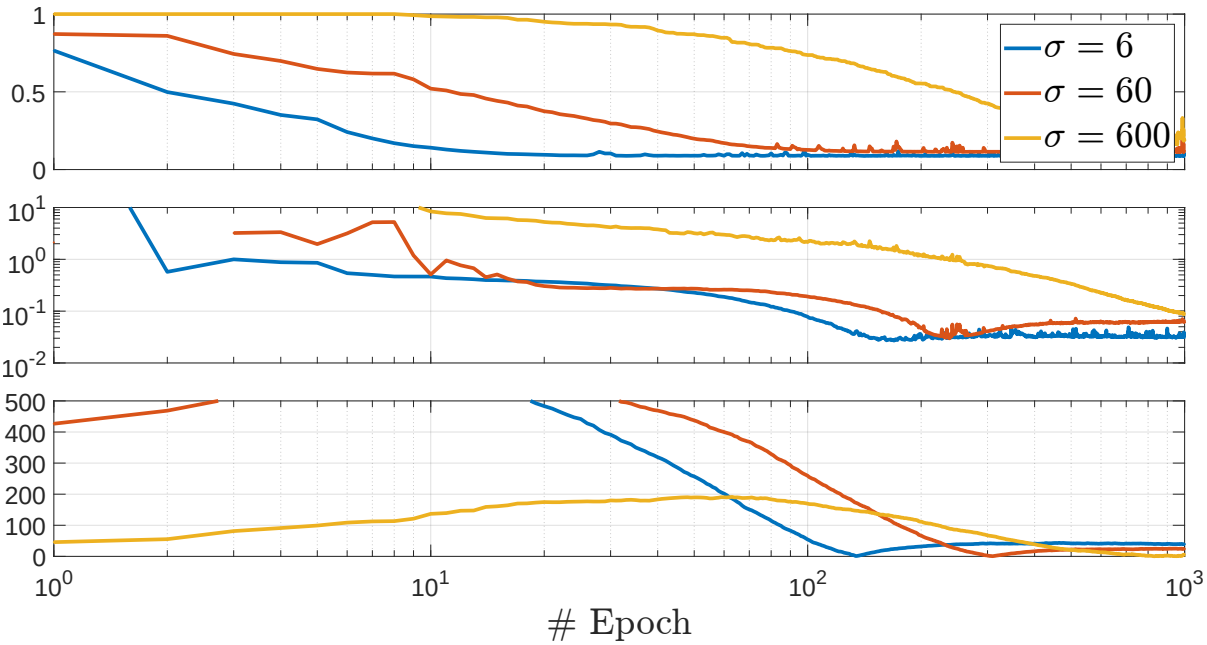



Figure 4.3.19: Spike distribution for our model problem and $A = -10\mathbf{I}$. Cumulative sum of all spikes for each epoch.

### Random init of matrices

As we have done in the section section 4.3.2, results so far have only considered optimal conditions for learning. In light of more challenging conditions, especially the initialization of $\mathbf{W}^f$ and $\mathbf{W}^s$ to zero is a significant aide towards the learning and network performance.

To test the influence of initial weights in both matrices we initialize the matrices with random numbers scaled to the appropriate range of $\mathbf{W}^f$ and $\mathbf{W}^f$ respectively.

For the first test both matrices were initialized to random number of the same scale as their optimal values. Results showed that from 1000 epochs of training 920 networks did not converge at all while in the remaining 80 epochs results were unusable. It was observed that while training the matrix eigenvalues diverged further from their optimal counterparts. This was also observed when measuring the largest rel. deviation.

When initializing with random values 50% of the scale convergence was obtained during all of training. However the relative error was more than 100% therefore rendering results useless.

If only either $\mathbf{W}^f$ or $\mathbf{W}^s$ is initialized randomly it becomes apparent that the $\mathbf{W}^s$ is the reason for the aforementioned problems. When initializing $\mathbf{W}^s$ to zero, acceptable results were obtained with $\mathbf{W}^f$ initialized to the previously impossible 100% noise scale. However this was only achieved with at least doubling the training time.

On the contrary, the initialization of $\mathbf{W}^s$ just 5% noise scale prohibited any convergence during training. The hypothesis, that potentially flipped signs in $\mathbf{W}^s$ being the reason for the lack of convergence was rejected as there was no change when each $\mathbf{W}^s_{ij}$ was given the same sign of the respective analytical value.
Thus, $\mathbf{W}^s$ has massive influence on $\mathbf{W}^f$. This can also be seen when attempting to run a simulation with learned $\mathbf{W}^s$ and analytical $\mathbf{W}^f$ as the network fails to converge. Yet when simulating the opposite configuration results can be obtained, although of subpar quality to the completely learned configuration.
This shows that $\mathbf{W}^s$ influences $\mathbf{W}^f$ to adapt to the (wrong) structure of itself. Hence it is advisable to only allow noise to impact $\mathbf{W}^f$. While this notion deviates from biologic plausibility since noise is inherent to any system and connectivities between neurons are not nullified at when learning commences, this is an manageable restriction for the sake of usable results.

### Limitations

Limitations in the learning of $\mathbf{W}^s$ arise when the eigenvalues of $\mathbf{A}$ have positive real parts. For purely imaginary eigenvalues, results depend on the magnitude of the imaginary part. For increasing imaginary parts the matrices converge to the correct structure be it not the correct magnitude. This can be combatted by tuning of hyperparameters though it depends on the $\mathbf{A}$, making it infeasible for wide range use. Without tuning, the error remains high and results unusable.
The second big limitation comes with the choice of $\mathbf{\Gamma}$. During testing, the choice of $\mathbf{\Gamma}$ impacted network accuracy. With randomly chosen $\mathbf{\Gamma}$ on $S^n$, the network error was too large to be useful This was especially the case when working with higher dimensional systems ($\mathbf{J} \geq 4$). The necessary accuracy for $\mathbf{W}^f$ to yield a functioning network was not learned anymore. Without sufficiently approximated $\mathbf{W}^f$, training was incapable to produce acceptable network performance.
Instead, $\mathbf{\Gamma}$ was chosen as equidistant points on $S^n$ to have uniform error projection and retain accurate results.
Apart from the choice of $\mathbf{\Gamma}$, higher order problems pose a problem for the networks learning. Due to small inaccuracies in training, errors add up in the integration of higher order problems. Especially $\mathbf{W}^s$ becomes the major factor in network error.
However this becomes significant only when no or little external input $\mathbf{c}(t)$ is applied. If $\mathbf{c}(t)$ is zero for large parts of the simulation time, errors in $\mathbf{W}^s$ compound to the point

where the network produces unusably erroneous results. If $\mathbf{c}(t)$ is non-zero, errors in $\mathbf{W}^s$ become less grave compared to $\mathbf{c}$ and the network output acceptable.

be specific here?

It is unclear what causes $\mathbf{W}^s$ to fail to reach the same accuracy compared to lower dimensional problems. No parameter set or amount of learning yielded the same level of precision seen in the results above. It is therefore necessary that the problem exhibits continuing external input.

Relative error at worst times 25%

maybe explain what divergence means for our contexts

Elaborate. What are other limitations?

long sim times

## 4.4 Results of the learned control objective

Now we review the results when we use the learned matrices $\mathbf{W}^f$ and $\mathbf{W}^s$ together with the control idea of $\mathbf{c} = \dot{\hat{\mathbf{x}}} - \mathbf{A}\hat{\mathbf{x}}$.
Since it was not possible to incorporate the control matrix $\mathbf{B}$ in the controller network but instead assumed to be the identity matrix, control results are only of limited interest.
Nonetheless we start by looking at looking at a jump response for our model problem we learned earlier.

### 4.4.1 Working as a open loop controller

To analyse the final block of results we take the trained matrices from section 4.3.4 with the control realization of section 4.2.3 that $\mathbf{c}$ is the major controlling force. With this we set up a classical simulation as in section 3.3 with the external input $\mathbf{c}$ defined analogously to eq. (3.36). For the computation we apply our reference state vector as well as its derivative making it a open loop controller without any feedback from the network.

Figure 4.4.1: Control results for the ideal and learned matrices $\mathbf{W}^f$ and $\mathbf{W}$ on a sine wave target. Top panel shows the $x_1$ state and bottom $x_2$. Matrices were trained for 1000 epochs using the previously studied optimal parameters.

Figure 4.4.2: Control results from fig. 4.4.1 but with only either $\mathbf{W}^f$ or $\mathbf{W}$ trained. Top panel shows the $x_1$ state and bottom $x_2$. Matrices were trained for 1000 epochs using the previously studied optimal parameters.

Network parameters are set such that every part is working on the same scale.

### First results

We again use our model problem and test different target trajectories with it.
To begin we set a sine wave as our target. As can be seen in fig. 4.4.1, the aforementioned control using $\mathbf{c}$ appears to deliver good results using the optimal weights. Both state trajectories almost perfectly overlap the target. However dynamics quickly diverge due to small imperfections in the learned matrices. With the open loop control, these error slowly add up leading to an almost complete loss of the target trajectory. The error shown in fig. 4.4.1 is almost purely due to inaccuracies in $\mathbf{W}^s$. If the control is rerun but instead the optimal $\mathbf{W}^s$ are used with the learned $\mathbf{W}^f$, much improved results can be obtained.
In fig. 4.4.2 the results show that if the ideal $\mathbf{W}^s$ is used, control accuracy is greatly improved. Small error still accumulate over time. These are from the remaining error in $\mathbf{W}^f$ that is caused by small errors from training but also because $\mathbf{W}^f$ adapts to the potential errors in $\mathbf{W}^s$ during training. The opposite configuration yields even worse results, proofing the principal error contribution stems from $\mathbf{W}^s$.
The main problem is that due to the open loop design very precise system information is necessary. With training, only a perturbed system $\mathbf{A}' = \mathbf{A} + \varepsilon$ of the system matrix is learned. Without state feedback, control over $\mathbf{A}'$ produces different dynamics which cannot be compensated with state feedback.
This limits the applicability of any learned matrices.

### Limitations

Due to the open loop design the network is susceptible to noise inside the network. Noise in the target trajectory is a lesser problem than noise inherent in the network.

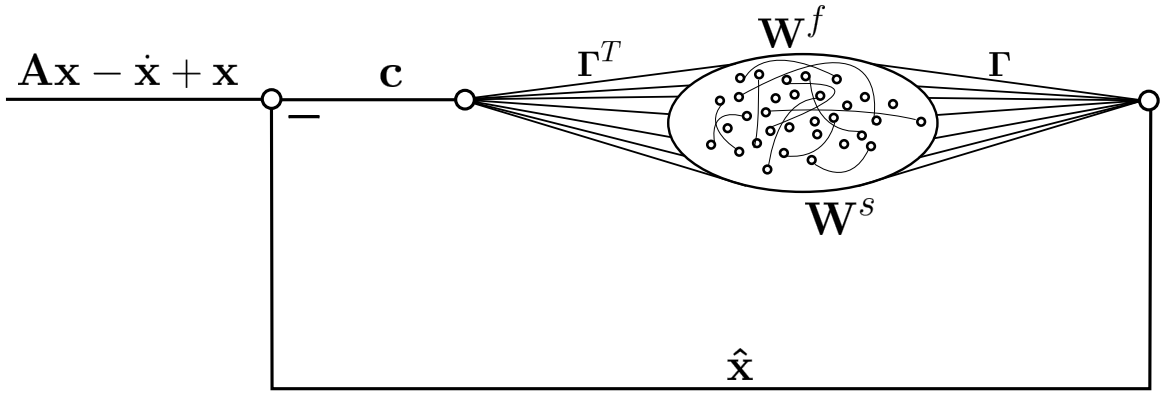



Figure 4.4.3: Add extra loop to introduce feedback into our control scheme. The outside loop can be integrated into the network similarly to $\mathbf{W}^s$.

The former can be directly translated into the network since we are implicitly working with $\mathbf{B} = I_{J\times J}$. Therefore each state is controlled separately and the noise only influence its respective state. The network noise inside the network is impossible to be accounted for due to the lack of feedback. Therefore inherent noise in the network is completely unchecked. The only method to combat this is by scaling the network with extremely small $\mathbf{\Gamma}$. Spikes caused by noise are negligible compared to the large number of spikes to represent a signal. The scaling basically converts it to a classic rate network.

### Adding Feedback

In order to build a more robust controller, we aim to add feedback to the control scheme. This is done similarly to the idea of how the feedback of fast and slow weights $\mathbf{W}^f$, $\mathbf{W}^s$ were introduced.
So far the external input $\mathbf{c}$ is computed via

$$\mathbf{c} = \mathbf{Ax} - \dot{\mathbf{x}} \tag{4.16}$$

where $\mathbf{x}$ is the target. The result is fed to the network. However to more accurately follow our target with state feedback, an additional error term is added to $\mathbf{c}$ for potential deviations of the state vector from the target. fig. 4.4.3 illustrates this idea. We therefore adapt the computation of $\mathbf{c}$ to

$$\mathbf{c} = \dot{\mathbf{x}} - \mathbf{Ax} + \mathbf{x} - \hat{\mathbf{x}} \tag{4.17}$$

Figure 4.4.4: Control results from fig. 4.4.1 but with the added feedback loop seen in fig. 4.4.3.

To measure its effectiveness, we first run the simulation with the extra added loop. In fig. 4.4.4 we can clearly see the network performance is greatly improved. Both trajectories follow their target almost perfectly.
Moreover with the addition of feedback, the robustness against noise has increased significantly during testing.

### Learning the new feedback loop

As a next step the outer feedback loop can be integrated in the network. With the given feed-forward/decoder, the loop can be added inside the network by the matrix

$$\mathbf{W}^{s_2} = \mathbf{\Gamma}^T \mathbf{\Gamma} \mathbf{r}. \tag{4.18}$$

Instead of introducing another matrix that is multiplied by the with the neuron firing rates, we can adjust the definition of $\mathbf{W}^s$ to

$$\mathbf{W}^s = \mathbf{\Gamma}^T \left( \mathbf{A} + (\lambda_d - 1)\mathrm{I} \right) \mathbf{\Gamma}. \tag{4.19}$$

This matrix can further be learned by the previous training algorithms by subtracting the identity matrix from $\mathbf{A}$ before training.
We check if the learning of $\mathbf{A} - \mathbf{I}$ gives similar to results with the extra loop in fig. 4.4.4. We retrain our model problem using the identical parameter set as before with the identity matrix subtracted.
To measure whether the training was able to reproduce the subtraction we calculate the inner learned matrix using regression.

$$\mathbf{M}^* = \min_{\mathbf{M}} \| \mathbf{\Gamma}^T \left( \mathbf{M} + \lambda_d \mathbf{I} \right) \mathbf{\Gamma} - \mathbf{W}^s \|_2^2 \tag{4.20}$$

It shows that in this test case the training was able to resolve the subtraction with a relative error of less than 8%.

Figure 4.4.5: Control results from fig. 4.4.4 but with the added feedback loop learned and integrated into the network directly.

Idk if this is really correctly written. What I want to say is that after both trainings the subtraction is clearly visible with the off diagonals being off from zero less than 0.023 and the main diagonals being of from an identity matrix by at most 0.075 which is kind of impressive.

As can be seen in fig. 4.4.5 this is sufficient to lead to almost perfect results as well.

Rewrite this to sound a bit more professional

Continue here or write another sentence to the above section.

### Control with rand init matrices for learing

Due to the feedback, it might be possible to use the matrices generated with random initialization from section 4.3.4. However this is not the case. Due to the lack of convergence in $\mathbf{W}^f$ the network control cannot compensate. Unfortunately the error in both matrices were too large to yield usable results.

This is part of the limitations

Explain how the feedback of the autoencoder gave rise to Ws. Same for Wf. Do it in the explanation of the efficient coding network. So basically the spikes are the input and the filtered spikes is the leaky integration of the input. But first add that the Fast weights made a feedback loop and the afterwards the slow weights as well...

### Limitations

Show the noise is in fact improved Fast signals All the problems of the previous methods Big system? A plot with different strengths of noise? Show how we improved with feedback Jumps? Step? Plausibility??

# Chapter 5

# <Conclusions>

Describe the conclusions (reflect on the whole introduction given in Chapter 1).

Discuss the positive effects and the drawbacks.

Describe the evaluation of the results of the degree project.

Describe valid future work.

The sections below are optional but could be added here.

In the end, we build a controller in three stages to control a linear system. The controller is based on SNNs using the Efficient balanced coding scheme with biologic plausibility. In this framework the first stage, the simulation, was capable of simulating any given dynamical system using analytically calculated values.
As a second step, a controller was devised to create the control signal $\mathbf{u}$. The controller was using the same SNN architecture and was supposed to be used in conjunction with the previous stage. The control signal or even the spikes would have been fed to the simulating network. However this idea was dismissed due to its problems in the implementation and function of the controller. Furthermore, the necessary conditions for the controller to work are prohibitively high that a realisation was deemed to only bring marginal benefits. Furthermore the dynamics of the controlling capabilities were predominantly stemming from the definition of the external inputs $\mathbf{c}$ instead of the whole network. For control, $\mathbf{c}$ acted as a forcing term to the network bridging the necessary input to push the network into following the desired trajectory. This purely open loop design faced the classic issues of open loop designs e.g in aspects of noise. In a separate similar derivation the emerging explicit error term was insignificant in most

test cases as well. These limitations in addition to the lack of a conceivable learning rule resulted that only the definition of $\mathbf{c}$ was adopted for control.
Lastly, the analytically derived network dynamics were learned with local learning rules. The fast recurrent dynamics were learned by an unsupervised learning rule while the slow dynamics trained by a supervised student-teacher approach. The learning performance was examined and parameters tuned to allowing for a significant improvement in network accuracy for our model problem. Challenges in the learning process were primarily connected to the random initialization of the matrices intended for training, the scaling of learning rate, form of training input sequences and initialization of $\mathbf{\Gamma}$. Higher-dimensional problems, in general, were only applicable in specific cases with manual parameter tuning, yielding mixed results overall.
In the last step, the previously learned matrices were run with the simulation build in stage one with the control scheme copied from the approaches in stage two. Due to the open loop design and the inherent perturbations in the learned matrices, the control was failing. With the addition of an extra feedback loop into the network, the definition of $\mathbf{c}$ was slightly altered to incorporate error feedback. Although the feedback greatly improved performance, the hope it was unable to work on non-zero initialized trained matrices that previously led to divergence even in mere simulation was not fulfilled.
Fortunately, the introduced feedback loop could be integrated the network itself and moreover learned automatically, given the initialization was zero.

Conclusion ist dass es nicht geeignet ist fuer ieine anwendung aber es schon ein system regeln kann.
Dann in final words sagen was wir gemacht haben und iwie auf das intro gehen

## 5.1 Discussion

Answer the questions of the problem!!!!

### 5.1.1 Future Work

Es so machen dass es waehrend dem task noch lernt. Z.b mit EWC oder dem Zenke ansatz fuer snn

Maybe different noise models.. Brown noise Adjust the input such that the imbalance can be negated and training is faster.
find nonlinearity
Investigate the spiking imbalance end point control
make the network acting as controller learn as well.
Test if learning with lower amplitude towards the end can refine the performance even more.

Double rate
adjustment
In epoch ar
overall epo

Further work:
Maybe learning methods to control nonlinear dynamic systems
Maybe we can even do the adverserial attack to try to screw with the network.
Implement this on neuromorphic hardware

### 5.1.2 Final Words

# Todo list

**If you are using mendeley to manage references, you might have to export them manually in the end as the automatic ways removes the ”date accessed” field**

# Bibliography


[1] Abdolrasol, Maher G. M., Hussain, S. M. Suhail, Ustun, Taha Selim, Sarker, Mahidur R., Hannan, Mahammad A., Mohamed, Ramizi, Ali, Jamal Abd, Mekhilef, Saad **and** Milad, Abdalrhman. "Artificial Neural Networks Based Optimization Techniques: A Review". **in**: *Electronics* 10.21 (3 **november** 2021), **page** 2689. ISSN: 2079-9292. DOI: 10.3390/electronics10212689. URL: https://www.mdpi.com/2079-9292/10/21/2689 (**urlseen** 10/02/2023).

[2] Adrian, E. D. **and** Zotterman, Yngve. "The impulses produced by sensory nerve-endings: Part II. The response of a Single End-Organ". **in**: *The Journal of Physiology* 61.2 (23 **april** 1926), **pages** 151–171. ISSN: 00223751. DOI: 10.1113/jphysiol.1926.sp002281. URL: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1926.sp002281 (**urlseen** 16/11/2022).

[3] Alemi, Alireza, Machens, Christian, Denève, Sophie **and** Slotine, Jean-Jacques. "Learning arbitrary dynamics in efficient, balanced spiking networks using local plasticity rules". **in**: (2017). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1705.08026. URL: https://arxiv.org/abs/1705.08026 (**urlseen** 06/12/2023).

[4] Almomani, Ammar, Alauthman, Mohammad, Alweshah, Mohammed, Dorgham, O. **and** Albalas, Firas. "A comparative study on spiking neural network encoding schema: implemented with cloud computing". **in**: *Cluster Computing* 22.2 (**june** 2019), **pages** 419–433. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-018-02891-0. URL: http://link.springer.com/10.1007/s10586-018-02891-0 (**urlseen** 15/11/2022).

[5] Amin, Hesham H. "Automated Adaptive Threshold-Based Feature Extraction and Learning for Spiking Neural Networks". **in**: *IEEE Access* 9 (2021), **pages** 97366–97383. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3094262.

URL: https://ieeexplore.ieee.org/document/9471863/ (**urlseen** 20/08/2023).

[6] Andrew, Alex M. "Spiking Neuron Models: Single Neurons, Populations, Plasticity". **in**: *Kybernetes* 32.7 (1 **october** 2003). ISSN: 0368-492X. DOI: 10.1108/k.2003.06732gae.003. URL: https://www.emerald.com/insight/content/doi/10.1108/k.2003.06732gae.003/full/html (**urlseen** 15/11/2022).

[7] Azevedo, Frederico A. C., Carvalho, Ludmila R. B., Grinberg, Lea T., Farfel, José Marcelo, Ferretti, Renata E. L., Leite, Renata E. P., Jacob Filho, Wilson, Lent, Roberto **and** Herculano-Houzel, Suzana. "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain". **in**: *The Journal of Comparative Neurology* 513.5 (10 **april** 2009), **pages** 532–541. ISSN: 1096-9861. DOI: 10.1002/cne.21974.

[8] Bekolay, Trevor, Bergstra, James, Hunsberger, Eric, DeWolf, Travis, Stewart, Terrence C., Rasmussen, Daniel, Choo, Xuan, Voelker, Aaron Russell **and** Eliasmith, Chris. "Nengo: a Python tool for building large-scale functional brain models". **in**: *Frontiers in Neuroinformatics* 7 (2014). ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048. URL: http://journal.frontiersin.org/article/10.3389/fninf.2013.00048/abstract (**urlseen** 30/11/2023).

[9] Bengio, Y., Simard, P. **and** Frasconi, P. "Learning long-term dependencies with gradient descent is difficult". **in**: *IEEE Transactions on Neural Networks* 5.2 (**march** 1994), **pages** 157–166. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/72.279181. URL: https://ieeexplore.ieee.org/document/279181/ (**urlseen** 07/02/2023).

[10] Bing, Zhenshan, Baumann, Ivan, Jiang, Zhuangyi, Huang, Kai, Cai, Caixia **and** Knoll, Alois. "Supervised Learning in SNN via Reward-Modulated Spike-Timing-Dependent Plasticity for a Target Reaching Vehicle". **in**: *Frontiers in Neurorobotics* 13 (3 **may** 2019), **page** 18. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00018. URL: https://www.frontiersin.org/article/10.3389/fnbot.2019.00018/full (**urlseen** 31/03/2022).

[11] Boerlin, Martin, Machens, Christian K. **and** Denève, Sophie. "Predictive Coding of Dynamical Variables in Balanced Spiking Networks". **in**: *PLOS Computational Biology* 9.11 (14 **november** 2013). Publisher: Public Library

of Science, e1003258. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003258. URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003258 (**urlseen** 20/09/2022).

[12] Bohté, Sander M., Kok, Joost N. **and** Poutré, Han La. "SpikeProp: backpropagation for networks of spiking neurons". **in**: *The European Symposium on Artificial Neural Networks*. 2000. URL: https://api.semanticscholar.org/CorpusID:14069916.

[13] Bouganis, Alexandros **and** Shanahan, Murray. "Training a spiking neural network to control a 4-DoF robotic arm based on Spike Timing-Dependent Plasticity". **in**: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010 International Joint Conference on Neural Networks (IJCNN). Barcelona, Spain: IEEE, **july** 2010, **pages** 1–8. ISBN: 978-1-4244-6916-1. DOI: 10.1109/IJCNN.2010.5596525. URL: http://ieeexplore.ieee.org/document/5596525/ (**urlseen** 10/08/2023).

[14] Bourdoukan, Ralph, Barrett, David, Deneve, Sophie **and** Machens, Christian K. "Learning optimal spike-based representations". **in**: *Advances in Neural Information Processing Systems*. **byeditor**F. Pereira, C. J. Burges, L. Bottou **and** K. Q. Weinberger. **volume** 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/3a15c7d0bbe60300a39f76f8a5ba6896-Paper.pdf.

[15] Bourdoukan, Ralph **and** Denève, Sophie. "Enforcing balance allows local supervised learning in spiking recurrent networks". **in**: *Advances in Neural Information Processing Systems*. **byeditor**C. Cortes, N. Lawrence, D. Lee, M. Sugiyama **and** R. Garnett. **volume** 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/3871bd64012152bfb53fdf04b401193f-Paper.pdf.

[16] Bourdoukan, Ralph **and** Denève, Sophie. "Enforcing balance allows local supervised learning in spiking recurrent networks". **in**: ().

[17] Brendel, Wieland, Bourdoukan, Ralph, Vertechi, Pietro, Machens, Christian K. **and** Denève, Sophie. "Learning to represent signals spike by spike". **in**: *PLOS Computational Biology* 16.3 (16 **march** 2020). Publisher: Public Library of Science, e1007692. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1007692.

URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007692 (**urlseen** 20/09/2022).

[18] Brette, Romain. “Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain”. **in**: *Frontiers in Systems Neuroscience* 9 (10 **november** 2015). ISSN: 1662-5137. DOI: 10.3389/fnsys.2015.00151. URL: http://journal.frontiersin.org/Article/10.3389/fnsys.2015.00151/abstract (**urlseen** 16/11/2022).

[19] Bullock, Daniel, Grossberg, Stephen **and** Guenther, Frank H. “A Self-Organizing Neural Model of Motor Equivalent Reaching and Tool Use by a Multijoint Arm”. **in**: *Journal of Cognitive Neuroscience* 5.4 (1 **october** 1993), **pages** 408–435. ISSN: 0898-929X, 1530-8898. DOI: 10.1162/jocn.1993.5.4.408. URL: https://direct.mit.edu/jocn/article/5/4/408/3102/A-Self-Organizing-Neural-Model-of-Motor-Equivalent (**urlseen** 10/08/2023).

[20] Chen, Yunhua, Mai, Yingchao, Feng, Ren **and** Xiao, Jinsheng. “An adaptive threshold mechanism for accurate and efficient deep spiking convolutional neural networks”. **in**: *Neurocomputing* 469 (16 **january** 2022), **pages** 189–197. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.10.080. URL: https://www.sciencedirect.com/science/article/pii/S092523122101568X (**urlseen** 20/08/2023).

[21] Clarke, D.D. **and** Sokoloff, L. “Circulation and energy metabolism of the brain”. **in**: *Basic Neurochemistry: Molecular, Cellular, and Medical Aspects* (1999), **pages** 637–669.

[22] Demin, Vyacheslav **and** Nekhaev, Dmitry. “Recurrent Spiking Neural Network Learning Based on a Competitive Maximization of Neuronal Activity”. **in**: *Frontiers in Neuroinformatics* 12 (15 **november** 2018), **page** 79. ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00079. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00079/full (**urlseen** 20/03/2023).

[23] Denève, Sophie **and** Machens, Christian K. “Efficient codes and balanced networks”. **in**: *Nature Neuroscience* 19.3 (**march** 2016), **pages** 375–382. ISSN: 1097-6256, 1546-1726. DOI: 10.1038/nn.4243. URL: http://www.nature.com/articles/nn.4243 (**urlseen** 18/10/2022).

[24] DeWolf, Travis, Stewart, Terrence C., Slotine, Jean-Jacques **and** Eliasmith, Chris. “A spiking neural model of adaptive arm control”. **in**: *Proceedings of the Royal Society B: Biological Sciences* 283.1843 (30 **november** 2016), **page** 20162134. ISSN: 0962-8452, 1471-2954. DOI: 10.1098/rspb.2016.2134. URL: https://royalsocietypublishing.org/doi/10.1098/rspb.2016.2134 (**urlseen** 15/12/2022).

[25] Diehl, Peter U., Neil, Daniel, Binas, Jonathan, Cook, Matthew, Liu, Shih-Chii **and** Pfeiffer, Michael. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”. **in**: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015 International Joint Conference on Neural Networks (IJCNN). Killarney, Ireland: IEEE, **july** 2015, **pages** 1–8. ISBN: 978-1-4799-1960-4. DOI: 10.1109/IJCNN.2015.7280696. URL: http://ieeexplore.ieee.org/document/7280696/ (**urlseen** 17/11/2022).

[26] Diehl, Peter U., Zarrella, Guido, Cassidy, Andrew, Pedroni, Bruno U. **and** Neftci, Emre. “Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware”. **in**: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016 IEEE International Conference on Rebooting Computing (ICRC). San Diego, CA, USA: IEEE, **october** 2016, **pages** 1–8. ISBN: 978-1-5090-1370-8. DOI: 10.1109/ICRC.2016.7738691. URL: http://ieeexplore.ieee.org/document/7738691/ (**urlseen** 17/11/2022).

[27] Eliasmith, Chris **and** Anderson, Charles H. *Neural engineering: computational, representation, and dynamics in neurobiological systems*. 1. MIT Press paperback ed. Computational neuroscience. Cambridge, Mass.: MIT Press, 2004. 359 **pagetotals**. ISBN: 978-0-262-55060-4.

[28] Eshraghian, Jason K., Ward, Max, Neftci, Emre O., Wang, Xinxin, Lenz, Gregor, Dwivedi, Girish, Bennamoun, Mohammed, Jeong, Doo Seok **and** Lu, Wei D. “Training Spiking Neural Networks Using Lessons From Deep Learning”. **in**: *Proceedings of the IEEE* 111.9 (**september** 2023), **pages** 1016–1054. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2023.3308088. URL: https://ieeexplore.ieee.org/document/10242251/ (**urlseen** 30/11/2023).

[29] Feldman, Daniel E. “The Spike-Timing Dependence of Plasticity”. **in**: *Neuron* 75.4 (**august** 2012), **pages** 556–571. ISSN: 08966273. DOI: 10.1016/j.neuron.2012.08.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0896627312007039 (**urlseen** 21/03/2023).

[30] Goodfellow, Ian, Bengio, Yoshua **and** Courville, Aaron. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. 775 **pagetotals**. ISBN: 978-0-262-03561-3.

[31] Graves, Alex, Eck, Douglas, Beringer, Nicole **and** Schmidhuber, Juergen. “Biologically Plausible Speech Recognition with LSTM Neural Nets”. **in**: *Biologically Inspired Approaches to Advanced Information Technology*. **byeditor**Auke Jan Ijspeert, Masayuki Murata **and** Naoki Wakamiya. redactor David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi **and** Gerhard Weikum. **volume** 3141. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, **pages** 127–136. ISBN: 978-3-540-27835-1. DOI: 10.1007/978-3-540-27835-1_10. URL: http://link.springer.com/10.1007/978-3-540-27835-1_10 (**urlseen** 14/12/2022).

[32] Guo, Wenzhe, Fouda, Mohammed E., Eltawil, Ahmed M. **and** Salama, Khaled Nabil. “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems”. **in**: *Frontiers in Neuroscience* 15 (4 **march** 2021), **page** 638474. ISSN: 1662-453X. DOI: 10.3389/fnins.2021.638474. URL: https://www.frontiersin.org/articles/10.3389/fnins.2021.638474/full (**urlseen** 20/03/2023).

[33] Hochreiter, Sepp **and** Schmidhuber, Jürgen. “Long Short-Term Memory”. **in**: *Neural Computation* 9.8 (1 **november** 1997), **pages** 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (**urlseen** 07/02/2023).

[34] Hodgkin, A. L. **and** Huxley, A. F. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. **in**: *The Journal of Physiology* 117.4 (1952). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764,

**pages** 500–544. ISSN: 1469-7793. DOI: 10.1113/jphysiol.1952.sp004764. URL: https://onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764 (**urlseen** 21/09/2022).

[35] Hodgkin, A. L. **and** Huxley, A. F. "Currents carried by sodium and potassium ions through the membrane of the giant axon of *Loligo*". **in**: *The Journal of Physiology* 116.4 (28 **april** 1952), **pages** 449–472. ISSN: 0022-3751, 1469-7793. DOI: 10.1113/jphysiol.1952.sp004717. URL: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004717 (**urlseen** 21/03/2023).

[36] Huang, Fuqiang. "Dynamics and Control in Spiking Neural Networks". **in**: (15 **december** 2019). Publisher: Washington University in St. Louis. DOI: 10.7936/YA3F-RK28. URL: https://openscholarship.wustl.edu/eng_etds/495 (**urlseen** 14/10/2022).

[37] Huang, Fuqiang **and** Ching, ShiNung. "Spiking networks as efficient distributed controllers". **in**: *Biological Cybernetics* 113.1 (**april** 2019), **pages** 179–190. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/s00422-018-0769-7. URL: http://link.springer.com/10.1007/s00422-018-0769-7 (**urlseen** 23/10/2022).

[38] Huang, Fuqiang, Riehl, James **and** Ching, ShiNung. "Optimizing the dynamics of spiking networks for decoding and control". **in**: *2017 American Control Conference (ACC)*. 2017 American Control Conference (ACC). ISSN: 2378-5861. **may** 2017, **pages** 2792–2798. DOI: 10.23919/ACC.2017.7963374.

[39] Indiveri, Giacomo **and** Sandamirskaya, Yulia. "The Importance of Space and Time for Signal Processing in Neuromorphic Agents: The Challenge of Developing Low-Power, Autonomous Agents That Interact With the Environment". **in**: *IEEE Signal Processing Magazine* 36.6 (**november** 2019), **pages** 16–28. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2019.2928376. URL: https://ieeexplore.ieee.org/document/8887553/ (**urlseen** 09/12/2022).

[40] Ioffe, Sergey **and** Szegedy, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2 **march** 2015. arXiv: 1502.03167[cs]. URL: http://arxiv.org/abs/1502.03167 (**urlseen** 07/02/2023).

[41] Izhikevich, E.M. “Simple model of spiking neurons”. **in**: *IEEE Transactions on Neural Networks* 14.6 (**november** 2003). Conference Name: IEEE Transactions on Neural Networks, **pages** 1569–1572. ISSN: 1941-0093. DOI: 10.1109/TNN.2003.820440.

[42] Jacobs, Nathan S., Allen, Timothy A., Nguyen, Natalie **and** Fortin, Norbert J. “Critical Role of the Hippocampus in Memory for Elapsed Time”. **in**: *Journal of Neuroscience* 33.34 (21 **august** 2013). Publisher: Society for Neuroscience Section: Brief Communications, **pages** 13888–13893. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.1733-13.2013. URL: https://www.jneurosci.org/content/33/34/13888 (**urlseen** 19/08/2023).

[43] Jaeger, Herbert. “The “echo state” approach to analysing and training recurrent neural networks – with an Erratum note”. **in**: (2010).

[44] Jin, Yingyezhe **and** Li, Peng. “Performance and robustness of bio-inspired digital liquid state machines: A case study of speech recognition”. **in**: *Neurocomputing* 226 (22 **february** 2017), **pages** 145–160. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.11.045. URL: https://www.sciencedirect.com/science/article/pii/S0925231216314606 (**urlseen** 14/12/2022).

[45] Johnson, Erik C., Jones, Douglas L. **and** Ratnam, Rama. “A minimum-error, energy-constrained neural code is an instantaneous-rate code”. **in**: *Journal of Computational Neuroscience* 40.2 (**april** 2016), **pages** 193–206. ISSN: 0929-5313, 1573-6873. DOI: 10.1007/s10827-016-0592-x. URL: http://link.springer.com/10.1007/s10827-016-0592-x (**urlseen** 24/11/2022).

[46] Johnston, Daniel **and** Wu, Samuel Miao-sin. *Foundations of cellular neurophysiology*. Cambridge, Mass: MIT Press, 1995. 676 **pagetotals**. ISBN: 978-0-262-10053-3.

[47] Kempter, Richard, Gerstner, Wulfram **and** Hemmen, J. Leo van. “Hebbian learning and spiking neurons”. **in**: *Physical Review E* 59.4 (1 **april** 1999), **pages** 4498–4514. ISSN: 1063-651X, 1095-3787. DOI: 10.1103/PhysRevE.59.4498. URL: https://link.aps.org/doi/10.1103/PhysRevE.59.4498 (**urlseen** 21/03/2023).

[48] Kirkpatrick, James, Pascanu, Razvan, Rabinowitz, Neil, Veness, Joel, Desjardins, Guillaume, Rusu, Andrei A., Milan, Kieran, Quan, John, Ramalho, Tiago, Grabska-Barwinska, Agnieszka, Hassabis,

Demis, Clopath, Claudia, Kumaran, Dharshan **and** Hadsell, Raia. “Overcoming catastrophic forgetting in neural networks”. **in**: *Proceedings of the National Academy of Sciences* 114.13 (28 **march** 2017), **pages** 3521–3526. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1611835114. URL: https://pnas.org/doi/full/10.1073/pnas.1611835114 (**urlseen** 30/08/2023).

[49] Lee, Jun Haeng, Delbruck, Tobi **and** Pfeiffer, Michael. “Training Deep Spiking Neural Networks Using Backpropagation”. **in**: *Frontiers in Neuroscience* 10 (8 **november** 2016). ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. URL: http://journal.frontiersin.org/article/10.3389/fnins.2016.00508/full (**urlseen** 10/02/2023).

[50] Legenstein, Robert, Pecevski, Dejan **and** Maass, Wolfgang. “A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback”. **in**: *PLoS Computational Biology* 4.10 (10 **october** 2008). **byeditor**Lyle J. Graham, e1000180. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000180. URL: https://dx.plos.org/10.1371/journal.pcbi.1000180 (**urlseen** 30/10/2023).

[51] Li, Xiangang **and** Wu, Xihong. *Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition*. 10 **may** 2015. arXiv: 1410.4281[cs]. URL: http://arxiv.org/abs/1410.4281 (**urlseen** 07/02/2023).

[52] Liu, Yuxiang **and** Pan, Wei. “Spiking Neural-Networks-Based Data-Driven Control”. **in**: *Electronics* 12.2 (7 **january** 2023), **page** 310. ISSN: 2079-9292. DOI: 10.3390/electronics12020310. URL: https://www.mdpi.com/2079-9292/12/2/310 (**urlseen** 06/09/2023).

[53] Maass, Wolfgang. “Liquid State Machines: Motivation, Theory, and Applications”. **in**: Cooper, S Barry **and** Sorbi, Andrea. *Computability in Context*. IMPERIAL COLLEGE PRESS, **february** 2011, **pages** 275–296. ISBN: 978-1-84816-277-8. DOI: 10.1142/9781848162778_0008. URL: http://www.worldscientific.com/doi/abs/10.1142/9781848162778_0008 (**urlseen** 31/10/2022).

[54] Maass, Wolfgang. “Networks of spiking neurons: The third generation of neural network models”. **in**: *Neural Networks* 10.9 (**december** 1997), **pages** 1659–1671. ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7. URL: https:

//linkinghub.elsevier.com/retrieve/pii/S0893608097000117 (**urlseen** 09/12/2022).

[55] Maass, Wolfgang, Joshi, Prashant **and** Sontag, Eduardo D. "Computational Aspects of Feedback in Neural Circuits". **in**: *PLoS Computational Biology* 3.1 (19 **january** 2007). **byeditor**Rolf Kotter, e165. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.0020165. URL: https://dx.plos.org/10.1371/journal.pcbi.0020165 (**urlseen** 07/11/2022).

[56] Maass, Wolfgang **and** Markram, Henry. "On the computational power of circuits of spiking neurons". **in**: *Journal of Computer and System Sciences* 69.4 (**december** 2004), **pages** 593–616. ISSN: 00220000. DOI: 10.1016/j.jcss.2004.04.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0022000004000406 (**urlseen** 07/11/2022).

[57] Mayer, Hermann, Gomez, Faustino, Wierstra, Daan, Nagy, Istvan, Knoll, Alois **and** Schmidhuber, Jurgen. "A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks". **in**: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. Beijing, China: IEEE, **october** 2006, **pages** 543–548. ISBN: 978-1-4244-0258-8. DOI: 10.1109/IROS.2006.282190. URL: http://ieeexplore.ieee.org/document/4059310/ (**urlseen** 07/02/2023).

[58] McKennoch, S., Dingding Liu **and** Bushnell, L.G. "Fast Modifications of the SpikeProp Algorithm". **in**: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. The 2006 IEEE International Joint Conference on Neural Network Proceedings. Vancouver, BC, Canada: IEEE, 2006, **pages** 3970–3977. ISBN: 978-0-7803-9490-2. DOI: 10.1109/IJCNN.2006.246918. URL: http://ieeexplore.ieee.org/document/1716646/ (**urlseen** 21/08/2023).

[59] Nair, Vinod **and** Hinton, Geoffrey E. "Rectified Linear Units Improve Restricted Boltzmann Machines". **in**: (2010).

[60] Neftci, Emre O., Mostafa, Hesham **and** Zenke, Friedemann. "Surrogate Gradient Learning in Spiking Neural Networks". **in**: (2019). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1901.09948. URL: https://arxiv.org/abs/1901.09948 (**urlseen** 21/08/2023).

[61] Nielsen, Michael A. “Neural Networks and Deep Learning”. **in**: (2015). Publisher: Determination Press. URL: http://neuralnetworksanddeeplearning.com (**urlseen** 10/02/2023).

[62] Pascanu, Razvan, Mikolov, Tomas **and** Bengio, Yoshua. *On the difficulty of training Recurrent Neural Networks*. 15 **february** 2013. arXiv: 1211.5063[cs]. URL: http://arxiv.org/abs/1211.5063 (**urlseen** 07/02/2023).

[63] Patel, Jigneshkumar **and** Goyal, Ramesh. “Applications of Artificial Neural Networks in Medical Science”. **in**: *Current Clinical Pharmacology* 2.3 (1 **september** 2007), **pages** 217–226. ISSN: 15748847. DOI: 10.2174/157488407781668811. URL: http://www.eurekaselect.com/openurl/content.php?genre=article&issn=1574-8847&volume=2&issue=3&spage=217 (**urlseen** 02/12/2022).

[64] Pfeiffer, Michael **and** Pfeil, Thomas. “Deep Learning With Spiking Neurons: Opportunities and Challenges”. **in**: *Frontiers in Neuroscience* 12 (2018). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2018.00774 (**urlseen** 15/12/2022).

[65] Putney, Joy, Conn, Rachel **and** Sponberg, Simon. “Precise timing is ubiquitous, consistent, and coordinated across a comprehensive, spike-resolved flight motor program”. **in**: *Proceedings of the National Academy of Sciences* 116.52 (26 **december** 2019). Publisher: Proceedings of the National Academy of Sciences, **pages** 26951–26960. DOI: 10.1073/pnas.1907513116. URL: https://www.pnas.org/doi/10.1073/pnas.1907513116 (**urlseen** 14/12/2022).

[66] Sak, Haşim, Senior, Andrew **and** Beaufays, Françoise. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. 5 **february** 2014. arXiv: 1402.1128[cs,stat]. URL: http://arxiv.org/abs/1402.1128 (**urlseen** 07/02/2023).

[67] Shrestha, Sumit Bam **and** Song, Qing. “Adaptive learning rate of SpikeProp based on weight convergence analysis”. **in**: *Neural Networks* 63 (**march** 2015), **pages** 185–198. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.12.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608014002676 (**urlseen** 21/08/2023).

[68] Soures, Nicholas **and** Kudithipudi, Dhireesha. “Deep Liquid State Machines With Neural Plasticity for Video Activity Recognition”. **in**: *Frontiers in Neuroscience* 13 (2019). ISSN: 1662-453X. URL: https://www.frontiersin.org/articles/10.3389/fnins.2019.00686 (**urlseen** 10/08/2023).

[69] Stagsted, Rasmus, Vitale, Antonio, Binz, Jonas, Renner, Alpha, Bonde Larsen, Leon **and** Sandamirskaya, Yulia. “Towards neuromorphic control: A spiking neural network based PID controller for UAV”. **in**: *Robotics: Science and Systems XVI*. Robotics: Science and Systems 2020. Robotics: Science **and** Systems Foundation, 12 **july** 2020. ISBN: 978-0-9923747-6-1. DOI: 10.15607/RSS.2020.XVI.074. URL: http://www.roboticsproceedings.org/rss16/p074.pdf (**urlseen** 10/08/2023).

[70] Stimberg, Marcel, Brette, Romain **and** Goodman, Dan Fm. “Brian 2, an intuitive and efficient neural simulator”. **in**: *eLife* 8 (20 **august** 2019), e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314. URL: https://elifesciences.org/articles/47314 (**urlseen** 30/11/2023).

[71] Sun, Hongze, Cai, Wuque, Yang, Baoxin, Cui, Yan, Xia, Yang, Yao, Dezhong **and** Guo, Daqing. *A Synapse-Threshold Synergistic Learning Approach for Spiking Neural Networks*. 3 **april** 2023. arXiv: 2206.06129[cs,q-bio]. URL: http://arxiv.org/abs/2206.06129 (**urlseen** 20/08/2023).

[72] Sun, Shiliang, Cao, Zehui, Zhu, Han **and** Zhao, Jing. *A Survey of Optimization Methods from a Machine Learning Perspective*. 23 **october** 2019. arXiv: 1906.06821[cs,math,stat]. URL: http://arxiv.org/abs/1906.06821 (**urlseen** 10/02/2023).

[73] Tanaka, Gouhei, Yamane, Toshiyuki, Héroux, Jean Benoit, Nakane, Ryosho, Kanazawa, Naoki, Takeda, Seiji, Numata, Hidetoshi, Nakano, Daiju **and** Hirose, Akira. “Recent advances in physical reservoir computing: A review”. **in**: *Neural Networks* 115 (**july** 2019), **pages** 100–123. ISSN: 08936080. DOI: 10.1016/j.neunet.2019.03.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608019300784 (**urlseen** 29/10/2022).

[74] Tang, Zaiyong **and** Fishwick, Paul A. “Feedforward Neural Nets as Models for Time Series Forecasting”. **in**: *ORSA Journal on Computing* 5.4 (**november** 1993), **pages** 374–385. ISSN: 0899-1499, 2326-3245. DOI: 10.1287/ijoc.5.

4.374. URL: http://pubsonline.informs.org/doi/10.1287/ijoc.5.4.374 (**urlseen** 02/02/2023).

[75] Tavanaei, Amirhossein **and** Maida, Anthony. "BP-STDP: Approximating backpropagation using spike timing dependent plasticity". **in**: *Neurocomputing* 330 (**february** 2019), **pages** 39–47. ISSN: 09252312. DOI: 10.1016/j.neucom.2018.11.014. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231218313420 (**urlseen** 22/08/2023).

[76] Thiruvarudchelvan, Vaenthan, Crane, James W. **and** Bossomaier, Terry. "Analysis of SpikeProp convergence with alternative spike response functions". **in**: *2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI)*. 2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI). Singapore, Singapore: IEEE, **april** 2013, **pages** 98–105. ISBN: 978-1-4673-5901-6. DOI: 10.1109/FOCI.2013.6602461. URL: http://ieeexplore.ieee.org/document/6602461/ (**urlseen** 21/08/2023).

[77] Thorpe, Simon, Delorme, Arnaud **and** Van Rullen, Rufin. "Spike-based strategies for rapid processing". **in**: *Neural Networks* 14.6 (**july** 2001), **pages** 715–725. ISSN: 08936080. DOI: 10.1016/S0893-6080(01)00083-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608001000831 (**urlseen** 19/08/2023).

[78] Uncini, Aurelio. "Audio signal processing by neural networks". **in**: *Neurocomputing* 55.3 (**october** 2003), **pages** 593–625. ISSN: 09252312. DOI: 10.1016/S0925-2312(03)00395-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231203003953 (**urlseen** 02/02/2023).

[79] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz **and** Polosukhin, Illia. *Attention Is All You Need*. 5 **december** 2017. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (**urlseen** 02/12/2022).

[80] Verstraeten, D., Schrauwen, B., D'Haene, M. **and** Stroobandt, D. "An experimental unification of reservoir computing methods". **in**: *Neural Networks* 20.3 (**april** 2007), **pages** 391–403. ISSN: 08936080. DOI: 10.1016/j.neunet.2007.04.003. URL: https:

//linkinghub.elsevier.com/retrieve/pii/S089360800700038X (**urlseen** 07/11/2022).

[81] Vigneron, Alex **and** Martinet, Jean. "A critical survey of STDP in Spiking Neural Networks for Pattern Recognition". **in**: *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020 International Joint Conference on Neural Networks (IJCNN). Glasgow, United Kingdom: IEEE, **july** 2020, **pages** 1–9. ISBN: 978-1-72816-926-2. DOI: 10.1109/IJCNN48605.2020.9207239. URL: https://ieeexplore.ieee.org/document/9207239/ (**urlseen** 22/08/2023).

[82] Webb, Andrew, Davies, Sergio **and** Lester, David. "Spiking Neural PID Controllers". **in**: *Neural Information Processing*. **byeditor**Bao-Liang Lu, Liqing Zhang **and** James Kwok. **volume** 7064. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, **pages** 259–267. ISBN: 978-3-642-24964-8. DOI: 10.1007/978-3-642-24965-5_28. URL: http://link.springer.com/10.1007/978-3-642-24965-5_28 (**urlseen** 10/08/2023).

[83] Williams, R. J. **and** Zipser, D. "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity". **in**: *Back-propagation:Theory, Architectures and Applications*. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1995, **pages** 433–486.

[84] Yamazaki, Kashu, Vo-Ho, Viet-Khoa, Bulsara, Darshan **and** Le, Ngan. "Spiking Neural Networks and Their Applications: A Review". **in**: *Brain Sciences* 12.7 (30 **june** 2022), **page** 863. ISSN: 2076-3425. DOI: 10.3390/brainsci12070863. URL: https://www.mdpi.com/2076-3425/12/7/863 (**urlseen** 22/08/2023).

[85] Yang, Mingchuan, Xie, Bingyu, Dou, Yingzhe **and** Xue, Guanchang. "Cascade Forward Artificial Neural Network based Behavioral Predicting Approach for the Integrated Satellite-terrestrial Networks". **in**: *Mobile Networks and Applications* 27.4 (**august** 2022), **pages** 1569–1577. ISSN: 1383-469X, 1572-8153. DOI: 10.1007/s11036-021-01875-6. URL: https://link.springer.com/10.1007/s11036-021-01875-6 (**urlseen** 02/02/2023).

[86] Yi, Zexiang, Lian, Jing, Liu, Qidong, Zhu, Hegui, Liang, Dong **and** Liu, Jizhao. "Learning rules in spiking neural networks: A survey". **in**: *Neurocomputing* 531 (**april** 2023), **pages** 163–179. ISSN: 09252312. DOI: 10.1016/j.neucom.

2023.02.026. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231223001662 (**urlseen** 20/03/2023).

[87] Zenke, Friedemann **and** Ganguli, Surya. "SuperSpike: Supervised learning in multi-layer spiking neural networks". **in**: *Neural Computation* 30.6 (**june** 2018), **pages** 1514–1541. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco_a_01086. arXiv: 1705.11146[cs,q-bio,stat]. URL: http://arxiv.org/abs/1705.11146 (**urlseen** 21/08/2023).

[88] Zenke, Friedemann, Poole, Ben **and** Ganguli, Surya. "Continual Learning Through Synaptic Intelligence". **in**: (2017). Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.1703.04200. URL: https://arxiv.org/abs/1703.04200 (**urlseen** 30/08/2023).

[89] Zhang, Yong, Li, Peng, Jin, Yingyezhe **and** Choe, Yoonsuck. "A Digital Liquid State Machine With Biologically Inspired Learning and Its Application to Speech Recognition". **in**: *IEEE Transactions on Neural Networks and Learning Systems* 26.11 (**november** 2015). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, **pages** 2635–2649. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2015.2388544.

[90] Zheng, Shengjie, Qian, Lang, Li, Pingsheng, He, Chenggang, Qin, Xiaoqin **and** Li, Xiaojian. "An Introductory Review of Spiking Neural Network and Artificial Neural Network: From Biological Intelligence to Artificial Intelligence". **in**: *arXiv:2204.07519 [cs]* (9 **april** 2022). arXiv: 2204.07519. URL: http://arxiv.org/abs/2204.07519 (**urlseen** 20/09/2022).

[91] Zhou, Shibo, Chen, Ying, Li, Xiaohua **and** Sanyal, Arindam. "Deep SCNN-Based Real-Time Object Detection for Self-Driving Vehicles Using LiDAR Temporal Data". **in**: *IEEE Access* 8 (2020). Conference Name: IEEE Access, **pages** 76903–76912. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2990416.

# Appendix - Contents

# Appendix A

# First Appendix

This is only slightly related to the rest of the report

# Appendix B

# Second Appendix

this is the information