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

This is a template for writing thesis reports for the ICT school at KTH. I do not own any of the images provided in the template and this can only be used to submit thesis work for KTH.

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

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
DS	Dynamic System
RNN	Recurrent Neural Network
NN	Neural Network
GD	Gradient Descent
ANN	Artificial Neural Network
SNN	Spiking neural network
GPU	Graphics Processing Unit
SRDP	Spike Rate 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
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator

Contents

Chapter 1

Introduction

Provide a general introduction to the area for the degree project. Use references!
Link things together with references. This is a reference to a section:

The human brain is a brilliant computing unit comprised of around 86 billion[6] 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 raw human computing power by several orders of magnitude, yet some 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. Furthermore with newer more biologically inspired networks we are able to solve a broader range of problems. One prospect of these are SNNs which go so far as to simulate the discrete spiking behaviour of natural neural networks. So with this in mind 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 by far 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[51] or Natural Language Processing (NLP) (using Transformers[61]).
This abstract representation bears 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 [24] and sequential data input but loose some of the advantages compared to the Feed-Forward due to its increased complexity.
A third generation[44] 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[40] or more performant with previous feed-forward implementations. This comes 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 300W for modern chips[2]. The brain however is estimated to only consume about 20W [15] for immense computing capacity. Accompanying the SNN with neuromorphic hardware can yield a similar boost in efficiency with processors energy consumption in the pJ per Synaptic Operation (SOP)[32] offering a huge potential power savings.

This is a separator

Present the background for the area. Give the context by explaining the parts that are needed to understand the degree project and thesis. (Still, keep in mind that this is an introductory part, which does not require too detailed description).

[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.
[2] e.g. a NVidia RTX 3090

Use references[3]

Detailed description of the area should be moved to Chapter 2, where detailed information about background is given together with related work.

1.2 Problem

Now list the goal: We want to do it for DS and check how good they are. Then method and then work. Take from below

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 [59][64][60] or to quantize the whole input if the complete time horizon is available. For example with recorded audio data.
Instead, recurrent neural networks are often proposed for these kinds of tasks. However recurrent neural networks experience problems when training with back-propagation[7]. 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[33], using alternative activation functions(ReLU)[48] or gradient clipping[50] to name a few. For recurrent models in particular different architectures have been suggested, most prominently among them the LSTM cell [26] with enormous success [42, 47, 54].
Yet, these recurrent designs are not a plausible representation of biological networks. For the control of biological movements (often) Linear Quadratic Regulator (LQR) is a proposed model for biological control [41] Secondly when it comes to simulation of biologic dynamic systems usually Linear Quadratic Gaussian (LQG) control is used. LQG has been used widely for modelling biological movements and control.

find refs

Add that LQ made many improveme but is also trash becau of the matr inversion, and nonline things in

this is another smaller separator

SNNs are inherently designed in time since neural impulses are integrated over time. This makes the use for temporal data natural.

[3]You can also add footnotes if you want to clarify the content on the same page.

The use of snn is online, can deal with much noise and offers an alternative way to reason for motor movement.

add other snn models. e.g reach, sorn maybe or others,FORCE

We hope that with SNN we have even better performance
From very biological to very abstract there have been many proposals
Cost performance trade off.
Spiking networks have gained similar or exceeding performance compared to the artificial one in some areas-> refs
Key advantage is in the temporal dimensional gain.
One field they are suited well is the control of dynamic systems
In this thesis we use a spiking neural network to control a linear dynamic system
The usual way to simulate biological dynamic systems is using LQG control -> ref
I believe because of the energy minimization
So we can compare them with usual NN and control in terms of performance... i guess
We start by giving an intro into spiking neural networks
Then spiking neural networks for dynamic systems
After control theory with SNN and maybe regular LQG control
Lately the learnign of SNNs for the control of dynamic system

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

Problem:
Problem is that it is unnatural for classic NN to use temporal data.
They usually quantize it and make a big input layer -> ref
There are recurrent networks but ... they need to have smth bad as well
The LQG control is also not great for some reason I need to find
There are many spkiking network archetypes like poisson and GLm and balanced
Also problem is that for some spiking networks learning rules could be hard to come by.
There are many prospects though as for example .....->refs

Also usually learning rules smth of an inverse and that the brain does not have or do I believe arvind said

Method:
We use a SNN to to control any arbitrary DS
Balanced networks show a some key motives seen in the brain like poisson distribution and smth else ->ref
The SNN is to be trained with a STDP rule
Then compared to optimal weights
Then investigated about robustness and other things as many before
One part of robustness is trying to get the most essential nodes of the snn to function well.
¸ Then we have the potential to find a classic nn and train it with that ???
With that out of the way we can compare the performance of all the methods.
Then we could study the usability for biological interpretation.
Maybe even train time over performance or smth whatever

Work:
Explain the controller method aka what the math of the controller
In method explain the balanced and the derivation
In work summarize the implementation
Same for the conventional NN
Summarize the training method
Explain and derive the training method in method

Results:
To everthign mentioned in method for performance and so on
Answer the questions of the problem!!!!

NN have excelled at many fields
Fields where they are not fit
aka temporal data
They have ways to compromise on that

-> reference
Spiking nn inherently temporal
more natural choice
However they also have problems
like the following:::: reference!!

1.3 Purpose

The purpose of the degree project/thesis is the purpose of the written material, i.e., the thesis. The thesis presents the work / discusses / illustrates and so on.

It is not “The project is about” even though this can be included in the purpose. If so, state the purpose of the project after purpose of the thesis).

Probably delete as a own paragraph but mention smth like that.

1.4 Goal

The goal means the goal of the degree project. Present following: the goal(s), deliverables and results of the project.

The goal of this project is to create a SNN that can control any given linear Dynamic System (DS). Furthermore should the Neural Network (NN) be robust against failing neurons or connections.

We expect better results to conventional NNs because of the SNN’s natural way to use temporal data. For the SNN itself we desire to find the optimal 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. In addition to that we seek performance when we control the system.

To mimic the brain’s learning, we want to use local training rules that are biologically plausible. The network should be converge to the optimal parameters.
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 optimum.
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 network is expected to improve on itself as it working the task at hand. Lastly, adjusting neural networks to a specific task is usually done by hand and requires time consuming hand tuning of parameters to achieve optimal results. Goal here is to automate as much of the process as possible i.e. the user does not need to adjust hyper-parameters himself. The network should be able to set itself up to find the best set of hyper-parameters given the task at hand, independent of the given control command or size of the system. We do not seek a perfect spike similar representation of spiking dynamics found in nature but construct a more plausible attempt that could be used in neuromorphic hardware. Lastly we are interested in the quality of the results if we restrict our methods to the natural limits of the brain.

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 does not require expensive computation necessary for e.g. LQG controller.

1.5 Methodology

Introduce, theoretically, the methodologies and methods that can be used in a project and, then, select and introduce the methodologies and methods that are used in the degree project. Must be described on the level that is enough to understand the contents of the thesis.

Use references!

Preferably, the philosophical assumptions, research methods, and research approaches are presented here. Write quantitative / qualitative, deductive / inductive / abductive. Start with theory about methods, choose the methods that are used in the thesis and apply.

Detailed description of these methodologies and methods should be presented in Chapter 3. In chapter 3, the focus could be research strategies, data collection, data

analysis, and quality assurance.

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 pros and cons. We seek an architecture that lies in between the most accurate biologic spiking model and yet does not abstract to many features of nature.
After that we implemented a LQG controller a baseline reference. 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 this goal by implementing a SNN that allows to simulate, not control, any given linear system with given external inputs. With this network in place it was set out to a second SNN that acts as the controller and combining those two. The controller would generate a control signal that would be used as external input to return the system behaviour to the controller. In the end, this approach depended on hand-tuning hyperparameters to set the controller to give usable results which was in opposition to our goals.

To find these magic numbers we tried to implement a learning regime based on a similar approach. However this was unfruitful.

Instead the original learning SNN approach is used to control the system directly.

1.6 Outline

In text, describe what is presented in Chapters 2 and forward. Exclude the first chapter and references as well as appendix.

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. Discuss what is found useful and what is less useful. Use valid arguments.

Explain what and how prior work / prior research will be applied on or used in the degree project /work (described in this thesis). Explain why and what is not used in the degree project and give valid reasons for rejecting the work/research.

Use references!

2.1 Use headings to break the text

Do not use subtitles after each other without text in between the sections.

2.2 Related Work

Spiking neural networks for control have been used in many different contexts, e.g for robotic movement, digit recognition[40] or object detection[56][68].
In [10] 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 [14], 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 [18] using the Neural

Engineering Framework (NEF)[21], in which different regions of the brain have been simulated to generate the trajectory as well as the control signals.

is to
ral?

The authors of [8] 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 inherit 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 [63] 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 [**stagsted_towards_2020**] 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.

TODO start looking at the F.Zenke stuff

Check the review papers that are open

This one only learn the kinematics and not any dynamic system.

Name prewitten libs for neural simulation brian nengo NeMo norse snntorch spinnaker bindsnet & snntorch based on pytorch

You should probably keep a heading about the related work here even though the entire chapter basically only contains related work.

Here just what has been done for each of the headlines

Previous efforts were already made to control dynamic systems with SNNs.

List here also efforts with other concepts apart from Balanced Networks

Neural networks in general spiking neural networks and their differences and what

they are better for. neuron models, iwazishi neuron and maybe one more mein neuron model und warum ich es ausgewaelt habe: einfach zu implementieren. Bereits fuer dynamische systeme verwendet, Nachteile dieses modells. Vlt vergleich mit einem anderen modell. Ganz kurzer ausflug in die regelung von dynamischen systemen.

What is a neural network? -> not here ref a paper. kurze erkl'rung in der einfuerung in der einfuhurng vlt auch hodgekin huxley erwaehen :)

2.3 Autoencoder

An autoencoder is a type of neural network for learning a representation of its input. It consists of an encoder and decoder function $z = f(x)$ & $\hat{x} = g(z)$. The encoder function 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 representation is as accurate as possible. This is easy in case the if $\mathcal{Z}$ is equal or larger than the input space. Then every possible input can be encoded by its own value as

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

which is not useful. $\mathcal{Z}$ has to many DOF and can "memorize" each input. In most cases however $\mathcal{Z}$ is constrained that the autoencoder has to find the relevant 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[23].

2.4 Neuron model

2.4.1 Biological Neuron model

The first biologically accurate model of neuron spiking behaviour is the Hodgkin–Huxley (HH) model from 1952[28]. 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 summed for the different ion channels in question, most commonly 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 [38].

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

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 α and β 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 [27] or [38]. The gates for each ion channel's conductance are found to be

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

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

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.

2.4.2 ”IF and LIF”

In contrast of the HH model in eq. (2.7), the simplest models of neurons are the Integrate and Fire (IF) and Leaky-integrate-and-fire (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 spiking neuron

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

is not truly
ct. Forgot
nts, but at
ame time
when there
nore than 1
on

and t_k being the time of the k-th spike. 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}.

2.4.3 Izhikevich Neuron

While the above models deliver a useful and cheap simplification, they lack in accuracy. The Izhikevich model [34] 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 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

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

be shitty
nation,
n could be
dded on.

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 [34].

2.4.4 Synaptic intelligence

In addition to the neuron model

2.5 Neural Networks

e clear
nction
een
ard nns
ann. Bcs
rently they
ot the
!

2.5.1 Artificial Neural Networks

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 no consent in the design decisions. As a result different choices are made by different researches leading to various approaches that are similar and yet different. Biologic realness can be chosen on different parts of the network. Therefore it is impossible to inspect each

nuisance explicitly. Apart from very specific implementations we illustrate below many SNNs can be categorized in different segments of which we highlight certain designs seen in the literature.

Neuron model

As outlined above the neuron model can have different complexities and accuracy. By far the most used model in the literature are LIF and IF model.

Encoding

The coding of information plays a crucial role it 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[**thorpe_spike-based_2001**].

Phase coding Phase coding is a slight variation of the TTFS code. In certain areas of the brain show oscillations similar to a clock[**jacobs_critical_2013**]. 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 λ. Using a Poisson point process comes with the drawback of comparatively high spiking to represent a value accurately due to noise.

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 distinguished between hierarchical and recurrent topologies. In hierarchical topologies there are segments of neurons that follow a (often unidirectional) structure. Feed-forward NNs 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[**chen_adaptive_2022**, **amin_automated_2021**, **ding_biologically_2022**]. One can adjust the frequency or likeliness of a neuron firing by lowering or increasing making it harder or easier for a neuron to spike respectively. This can used to model refractory periods after a spike was fired. Threshold adaptation can also be in conjunction to weight adaption

policy[**sun_synapse-threshold_2023**].

- Adjusting weights
- Dynamic threshold
- synaptic intelligence

Learning Algorithms

- (Reward modulated) STDP and many variations
- Anti-Hebbian
- BBTT
- Real time
- Surrogate Gradients
- Spike-prop
- SuperSpike
- spike time dependent back prop
- …

Decoding

- Linear
- Quadratic
- Non-linear
- see the open paper!

2.5.2 Plasticity

STDP
Gradient descent
Hebbian
maybe?

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[67]. There are various ways to train a network. For ANNs, gradient based algorithms are often the method of choice. They use the error from the loss function do compute the derivative $\frac{\partial L}{\partial \theta_{ij}}$ to adjust each parameter θ, i.e. weights and biases with

$$\theta_{ij} = \theta_{ij} - \eta \frac{\partial L}{\partial \theta_{ij}} \tag{2.13}$$

until a local minima 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 [23] or [49].

There are a plethora of different learning techniques available, see [1][57] for a review. The most fundamental distinction can be made between supervised, unsupervised and reinforcement learning rules. One needs to remember that ANNs and SNNs require completely different learning algorithms because of their different transport of information. A short summary of learning rules in SNN is given in a later chapter.

2.6 Spiking Neural Networks

A spiking Neural network is one step closer to a biologic representation of a brain. Instead of conveying information using a gradient in conventional NNs, information is propagated using discrete spikes of excitation, similar to biological neurons. Hereby one can distinguish between several ideas of implementation.

main
rence is
notion of

2.6.1 Rate Networks

Poisson 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[13]. This makes a strong contrast to the approach chosen in this paper, where every spike will be timed exactly 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[17].

be add that
pproach
not
rate out
pletely.

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[3] and can be trained with different training algorithms e.g. (anti-)Hebbian, STDP, gradient based or more involved training methods[16]. See [65] 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 $\frac{1}{\sqrt{N}}$[9]. This means to get accurate results a huge amount of spikes need to be fired. This is intuitive as the time between spikes is exponentially distributed and the more samples are available the better the rate parameter can be estimated. Theoretically using a large number of spikes is unproblematic and even with large spike counts neuromorphic hardware is still much more energy efficient than deep neural networks when deployed[32].

I can delive
the derivati
of that num
if necessary

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[4]. This means that the rate decoding is to slow to capture the fast travelling information[25] and there need to be faster ways to transmit information. The second problem is that due to the Poisson process one needs more spikes to represent an average firing rate. An action potential consumes a lot of the cells energy[5], thus making it unfeasible to use a large number of spikes if one tries to model the brain. There are more efficient ways to transmit information

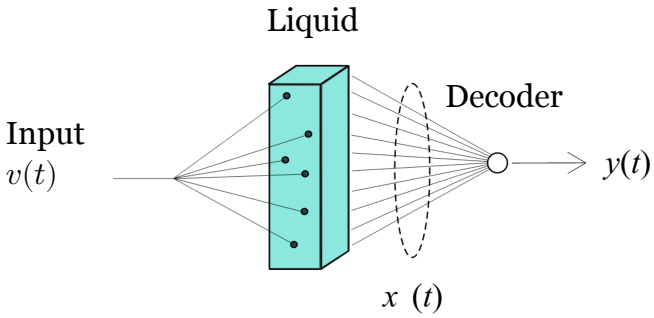


Figure 2.6.1: Abstract idea of Reservoir Computing. Adapted from [43]

Although the above mentioned problems, rate encoded SNNs have seen interest by 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 afterwards convert the trained network to a SNN[52] using rate encoding[20][19].

2.6.2 Liquid state machines

write how the offline computing is pretty bad for brain things, but good for chess for example. The online computing is what the brain does and it is not yet as developed.

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 approach[62].
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[43]. A reservoir can be a mathematical abstract formulation or physical object, e.g. a literal bucket of water [58].
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[35]. This is a considerable time saver 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

$$z^{(n)} = G(z, z^{(1)}, z^{(2)}, \dots, z^{(n-1)}) + u \tag{2.14}$$

given a sufficiently large liquid and a suitable feedback and decoder[46] and have been used for speech recognition[36][66] or object detection[56]. The systematic structure can be in seen in fig. 2.6.2. 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.

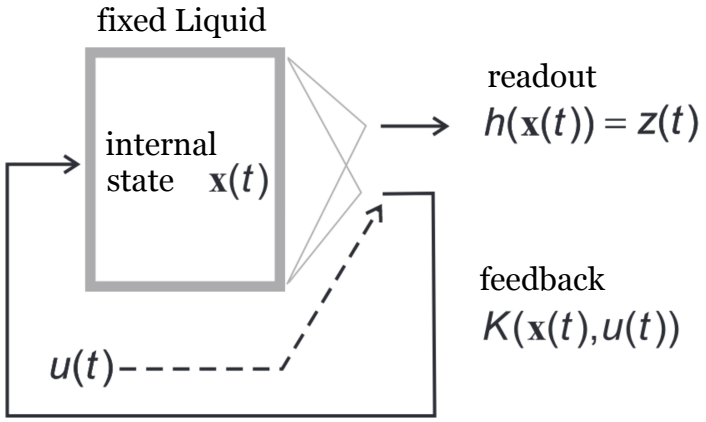


Figure 2.6.2: Adding suitable feedback allows LSMs to be universal approximator. Adapted from [45]

2.6.3 Balanced Networks

The idea of tightly balanced spiking networks was first proposed by Boerlin et al.[9]. It uses predictive coding in combination with spikes to simulate arbitrary linear systems. The technical derivation will be described in section 3.2.1. The approach defines a cost function measuring the networks' accuracy in addition with regularization terms that moderates the spiking behaviour. Using a greedy 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 [37] utilized.
Additionally, every neuron is given an specific area of the system to surveil. The projection of the system error therefore has therefore immense influence on an individual neuron's spiking behaviour. On the other hand on 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.

more
rences prob
explain
why we
his one.

2.6.4 Plasticity

It's important to remember that ANNs and SNNs require completely different learning algorithms because of their different transport of information.
Gradient based methods require differentiability and therefore continuity, thus are only applicable for ANNs. This means that they cannot be used for SNNs since spikes introduce discontinuities. Methods have been proposed to use backpropagation in spiking networks[40] yet they do not provide a biologically plausible way to learn. The

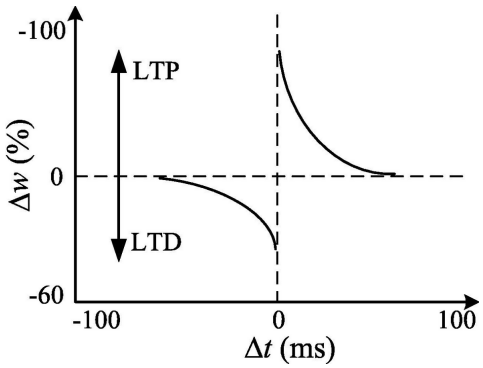


Figure 2.6.3: Graphical representation of STDP learning rule. Weight change depending on the time between pre- and postsynaptic activation. Negative time is the time between the postsynaptic neuron firing before the presynaptic. Graphic taken from [65].

problem is that in biology neurons do not have access to the global error from the loss function but only to their pre-synaptic neurons.

(Anti-) Hebbian Learning and STDP

The Hebbian learning rule is one of the oldest learning rules for neural networks with large experimental evidence in biology(see for example [22] for a summary). Its key idea can be summarized as ”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. 2.6.3 and build the bases for the spike time dependent plasticity rule (STDP). 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. 2.6.3 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. 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)[39], which have been summarized in [65].

Chapter 3

<Engineering-related content, Methodologies and Methods>

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

Research question: Develop a biologically sensible SN 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[18]. In order to stay biologically more realistic we ignore purely rate based spiking networks as there is evidence that the precise spike timing is relevant in nature[13][53]. On the other side of the specturm it makes little sense to use HH's model. Even though it is very biologically plausible its increased complexity and little abstraction makes it more suited for solely accurate biological neural simulation and less for the engineering task at hand. Additionally there are no training/learning rules available to solve such a high level problem.

Explain choice of SNN architecture
HH doesnt make sense because to complex.
Recurrent models are possible but also not biologic enough because we want to use the spiking property
LSM was considered but in a LSM you only learn the decoder. For the problem at hand

is is more natural to have the dynamics in a neural network. Plus with this approach we learn the decoder as well so there it makes it for a more independent approach.

Design a LQG controller to start with as reference First found how to simulate a dynamical system with given input c from Boerlin
Then finding the controller structure to find exteral input c to control the system in a desired way
Then notice you need to have magic numbers to get it to work properly.
Then trying to bring the learning mechanics to that approach.
Deemed difficult
finding way to control a dynamical system with the learning SNN framework
Also not so far to do nonlinear systems

Describe the engineering-related contents (preferably with models) and the research methodology and methods that are used in the degree project.

Most likely it generally describes the method used in each step to make sure that you can answer the research question.

3.2 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

add reference to the goal section

. We begin with the formal derivation of the network dynamics.

3.2.1 Balanced network simulation

This section follows the derivation found in [9] and [29]. 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

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

$\mathbf{\Gamma}$ 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[12].
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

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

explain how this is better than just rate encoding

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

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

3.2.2 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. (3.22) 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 ϵ 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 and the constant Right Hand Side (RHS) as the voltage threshold T_i

$$V_i > T_i = \frac{\|\mathbf{\Gamma}_i\|^2+\nu+\mu}{2}. \tag{3.16}$$

Remember that i read somewhere that the noi is necessary Maybe men that here to And find th reference

3.2.3 Neuron Voltage

As mentioned above, a neuron spikes if it meets the condition eq. (3.16). But so far it is unclear how neuron voltage evolves over time. Denote $\mathbf{L}$ the the left pseudo-inverse of $\mathbf{\Gamma}$

$$\mathbf{L}=\left(\mathbf{\Gamma}\mathbf{\Gamma}^T\right)^{-1}\mathbf{\Gamma} \tag{3.17}$$

such that $\mathbf{L}\mathbf{\Gamma}^T=\mathbf{I}$.
Next, taking the derivative of eq. (3.29) yielding

$$\dot{\mathbf{V}}(t)=\mathbf{\Gamma}^T\left(\dot{\mathbf{x}}(t)-\dot{\hat{\mathbf{x}}}(t)\right)-\mu\dot{\mathbf{r}}(t). \tag{3.18}$$

Now using the pseudo-inverse to rewrite the voltage equation eq. (3.29) as

$$\begin{aligned} \mathbf{V}(t) &= \mathbf{\Gamma}^T(\mathbf{x}(t)-\hat{\mathbf{x}}(t))-\mu\mathbf{r}(t) \\ \mathbf{L}\mathbf{V}(t) &= (\mathbf{x}(t)-\hat{\mathbf{x}}(t))-\mu\mathbf{L}\mathbf{r}(t) \\ \mathbf{x}(t) &= \mathbf{L}\mathbf{V}(t)+\hat{\mathbf{x}}(t)+\mu\mathbf{L}\mathbf{r}(t) \end{aligned} \tag{3.19}$$

Now the derivative terms in eq. (3.18) are replaced with with their respective equations eq. (3.1), eq. (3.2) and eq. (3.3). Lastly we substitute eq. (3.19) in eq. (3.18) and obtain

$$\begin{aligned}\dot{\mathbf{V}} &= \mathbf{\Gamma}^T\mathbf{A}\mathbf{L}\mathbf{V} \\ &+ \left(\mathbf{\Gamma}^T\mathbf{A}\mathbf{\Gamma} + \mu\mathbf{\Gamma}^T\mathbf{A}\mathbf{L} + \lambda_d\mathbf{\Gamma}^T\mathbf{\Gamma} + \boldsymbol{\mu}\boldsymbol{\lambda}_\mathbf{d}\right)\mathbf{r} \\ &+ \left(\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\right)\mathbf{o} + \mathbf{\Gamma}^T\mathbf{c}.\end{aligned} \tag{3.20}$$

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. (3.16) will not get dominated by cost terms μ & ν, 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

$$\begin{aligned}\dot{\mathbf{V}} &= -\lambda_V\mathbf{V} + \mathbf{W^s}\mathbf{r} + \mathbf{W^f}\mathbf{o} + \mathbf{\Gamma}^T\mathbf{c} \\ \mathbf{W^s} &= \mathbf{\Gamma}^T(\mathbf{A} + \boldsymbol{\lambda}_d\mathbf{I})\mathbf{\Gamma} \\ \mathbf{W^f} &= -\left(\mathbf{\Gamma}^T\mathbf{\Gamma} + \mu\mathbf{I}\right)\end{aligned} \tag{3.21}$$

find a coherent name for the matrix

3.2.4 Regularization

Two regularization terms are added to influence spiking behaviour.

$$E(t) = \int_0^t \left(\|\mathbf{x}(u) - \hat{\mathbf{x}}(u)\|_2^2 + \nu\|\mathbf{r}(u)\|_1 + \mu\|\mathbf{r}(u)\|_2^2\right) du \tag{3.22}$$

The parameter ν controls the amount of spiking by penalizing the total number of spikes as

$$||r(t)||_1 = \sum_i |r_i(t)| = \sum_i r_i(t). \tag{3.23}$$

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 kernel has the form

$$\Gamma = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{3.24}$$

Ignoring the cost terms in eq. (3.16) the threshold is set at

$$V_i > \frac{\|\mathbf{\Gamma}_i\|^2}{2} \tag{3.25}$$

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

$$V_i = V_i + \mathbf{W}^{\mathbf{s}}_{ii} = V_i + \|\mathbf{\Gamma}_i\|_2^2 \quad \text{with } \mathbf{W}^{\mathbf{f}} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.26}$$

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

$$T_i = \frac{\|\mathbf{\Gamma}_i\|^2}{2} = \frac{-\text{diag}(\mathbf{W}^{\mathbf{f}})}{2}. \tag{3.27}$$

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 pattern repeats and destroys network performance.

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

maybe a pi

from

$$\mathbf{V} = \mathbf{V} + \mathbf{W^f o} = \mathbf{V} + \mathbf{W^f} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathbf{V} + \mathbf{W}^{\mathbf{f}}_{:0}$$
$$\mathbf{V} = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} \tag{3.28}$$

$\mathbf{W}^{\mathbf{f}}_{00}$ whereas the second neuron gets pushed up to 0.5 , causing a spike. This in turn reverts the changes of eq. (3.28) 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 we set the threshold slightly higher. As seen above in eq. (3.16), 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 do not make a difference since it is penalizing the global number of spikes but does not discern where the spikes are fired. With the quadratic cost the norm of spike rates distributed among many neurons is reduced compared to few.

The first was termed ”ping-pong” effect and is described in the supplementary material of [9]. To understand the issue, we imagine a minimal network consisting of 2 neurons with equal kernel but opposite sign.

e better
ing pong
t! Maybe

The second regularization comes into play when there are kernels with different magnitude. Kernels with small kernel magnitude reach their threshold sooner and therefore fire more frequently. In the extreme case, only small number of neurons fire rapidly while the majority remains idle. By penalizing the rate in the 2-norm it forces the network to spread the firing among the whole network.

the right
to explain

The dynamic variable x is tracked by firing spikes in when the defined ”pseudo voltage” of a neuron surpasses its threshold. The voltage for each neuron is defined by

$$V_i(t) = \mathbf{\Gamma}^T(\mathbf{x}(t) - \hat{\mathbf{x}}(t)) - \mu\lambda_d r_i(t) \quad i = 1 \dots N. \tag{3.29}$$

Add figure

Figure 3.3.1: Schematic to illustrate the use of balanced networks as controllers.

For negligible quadratic cost μ the voltage can be understood as measure of the error projected on $\mathbf{\Gamma}_i$. The explicit derivation of the above equation is found in [9] and will be adapted .

Where? He in the appe of at all?

3.3 Control of Dynamic systems using SNNs

3.3.1 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 [31] and is illustrated in fig. 3.3.1. 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. 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.

3.3.2 Dynamics

The derivation of this method is similar to the one in section 3.2. 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, [31] 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 [29] is similar to [9] 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 [31]. The results summarize to eqs. (3.33) to (3.38).

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

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

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

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

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

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

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 $\mathbf{\Omega}$.

3.3.3 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 ϵ 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.

3.3.4 Extension with direct Error feedback

The same group of [31] later published a new but very similar approach in [30] which is based on the same idea, however the approach in [30] 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. (3.18) 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 [31], similarly done in eq. (3.20) and the subsequent neglect of certain terms, changes the definition of $\mathbf{W}^{\mathbf{s}}$. In the derivation in [30] this is not done and therefore terms

remain changing $\mathbf{W^s}$'s definition to

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

add that there is always noise somewhere

3.4 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.

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

3.4.2 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. (3.20) 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 [11] is used.
For this we consider a "learner" dynamic system of the form

$$\dot{\hat{\mathbf{x}}} = \mathbf{M}\hat{\mathbf{x}} + \mathbf{c}(t) \tag{3.46}$$

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

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}(t). \tag{3.47}$$

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

$$\dot{\hat{\mathbf{x}}} = (\mathbf{M} + K\mathbf{I})\,\hat{\mathbf{x}} + \mathbf{c}(t) + K\mathbf{x} \tag{3.48}$$

is then calculated by using minimizing the loss

$$L = \frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{3.49}$$

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

$$\dot{M}_{ij} = -\frac{\partial L}{\partial M_{ij}} = \left(\frac{\partial \hat{\mathbf{x}}}{\partial M_{ij}}\right)^T \mathbf{e}. \tag{3.50}$$

3.5 Engineering-related and scientific content:

Applying engineering related and scientific skills; modelling, analysing, developing, and evaluating engineering-related and scientific content; correct choice of methods based on problem formulation; consciousness of aspects relating to society and ethics (if applicable).

As mentioned earlier, give a theoretical description of methodologies and methods and how these are applied in the degree project.

was ist meine research question?

zusammensetzung von den beiden systeme: dynamisches system und neuronales netz. mehr oder weniger die herleitung kopieren aus dem paper. Dann mit learning von den gewichten.

Here I describe what how it needs to be done. So this is the place for the derivation The concept and the process whatever that means Later there comes the how I implemented it. Here is what we needs to be implemented.

Here very detailed explanation of the Balanced network for this problem
Very detailed way for the regular NN for this problem Basics of the controller design used in this comparison aka LQG controller
Method of learning the weights for the SNN Method of comparison

Chapter 4

<The work>

Describe the degree project. What did you actually do? This is the practical description of how the method was applied. How do we make the SNN MAtlab Balanced spiking network (say why to use that)
maybe pseudo code Ideally some theorem (convergence???) Simulation? nein kommt in den naechsten part

4.1 Writing the balanced network simulation

4.2 Writing the balanced network controller

4.3 Learning the dynamics

Write that the most of all the control is pretty picky for parameters. This means that it is not very versatile. So instead we want to learn our own parameters so we have adaptability. Enter in the learning paper of Deneve and Brendel and why it would be nice. It would be nice bcs we can learn the parameters independently and can preserve some structure or the fact that we want to include B matrix somewhere This doesnt work for some reasons I need to point out. Then the bourdoukan paper Then we set this for control. And results

Chapter 5

Results

Describe the results of the degree project. Analyses of results

How does the optimal network simulate? Influence of parameters
What are the magic parameters? mu etc

The control with optimal works fine too. Influence of the parameters Magic number tuning required for x,y,z etc

Learning works fine with magic numbers Examples. What works better? What works worse? What if I only learn 1 part? What is the bigger error. Convergence over time usually very bad Compared to the eigenvalues maybe convergence improves How do the parameters look? Heatmap How to judge the accuracy of the closeness of the parameters to the optimal What is the influence of the input You have to juggle the values of learning rate and input amplitude to reach ideal results

5.1 Results on the Simulation from ??

5.1.1 Toy Example

The results from simulating the network with given input $\mathbf{c}(t)$ can be seen in fig. 5.1.1. In this scalar example the ODE

$$\dot{x} = -10x + c(t) \tag{5.1}$$

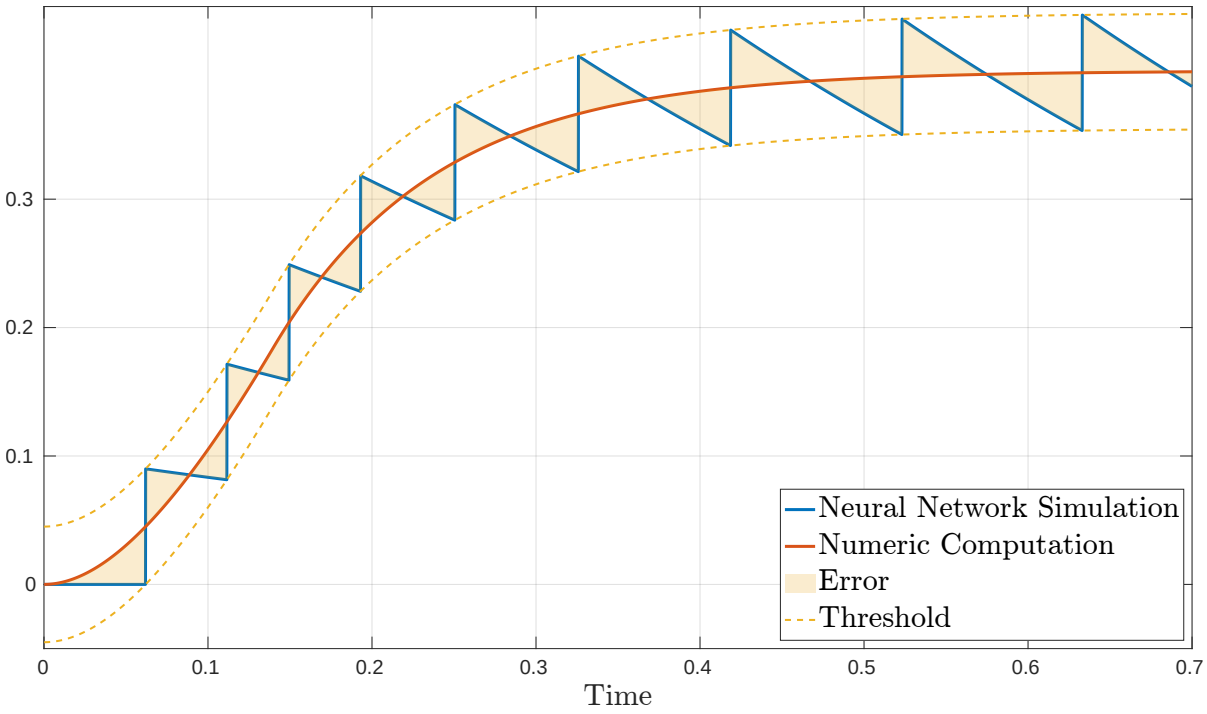
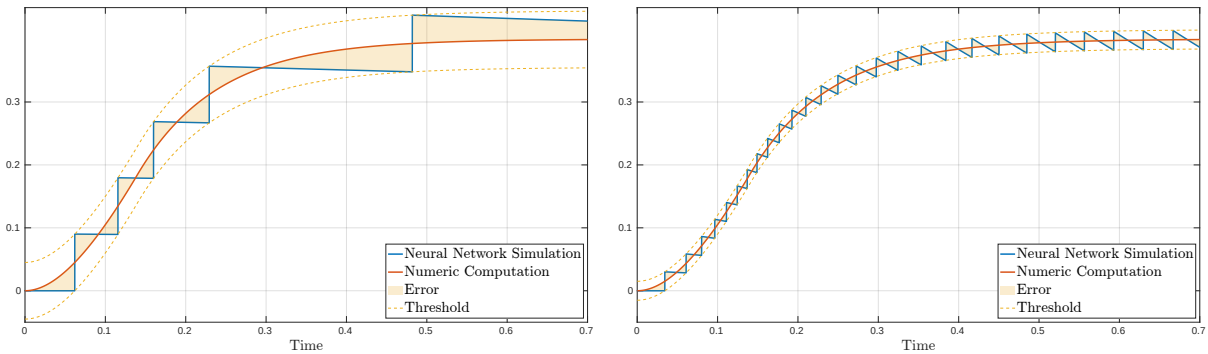


Figure 5.1.1: Baseline example

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 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. 5.1.1, 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 5.1.2: Variation of Readout Decay and Decoder for simple 1D system.

In fig. 5.1.2a the readout decay is reduced compared to fig. 5.1.1 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. 5.1.2b. Since the threshold is closely tied to the Decoding weights this also reduces the spike threshold and therefore yields more accurate results.

5.1.2 Toy example in 2D

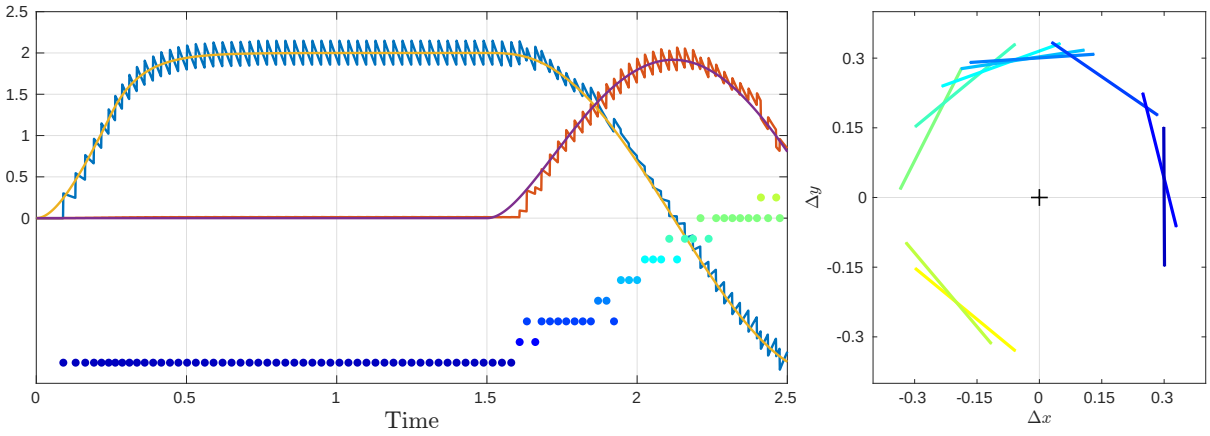


Figure 5.1.3: Simple 2D example with numerical solution and spike response. Curves for x in yellow/blue. Curves for y in purple/red. The networks output closely tracks the perfect numerical solution. For the network each decoding vector was chosen from a normal distribution and normalized to $\|\mathbf{\Gamma}_i\|_2 = 0.3$. Beneath a raster plot for each spiking neuron
On the left the threshold for each neuron's projected error.

5.1.3 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. (5.1) but in two dimensions. We have the corresponding results in fig. 5.1.4 on the right. On the left we a phase plot in the xy-axis.

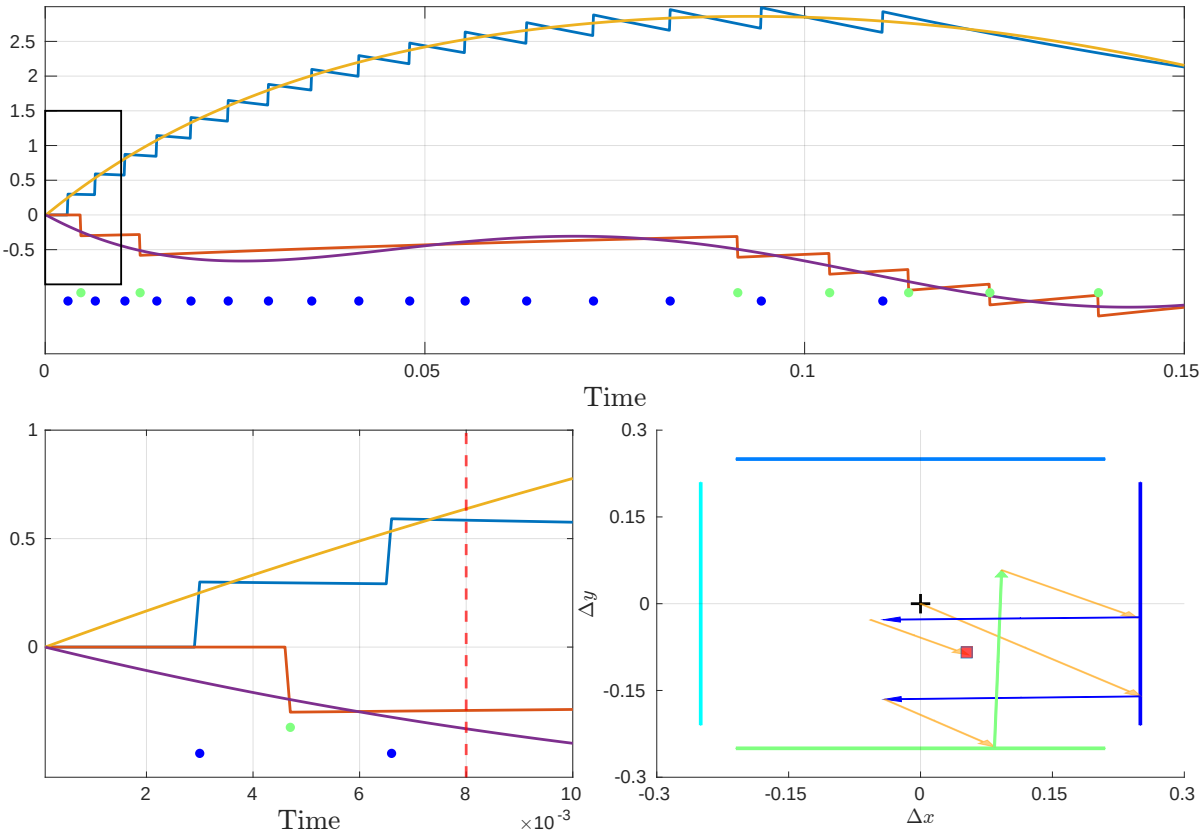
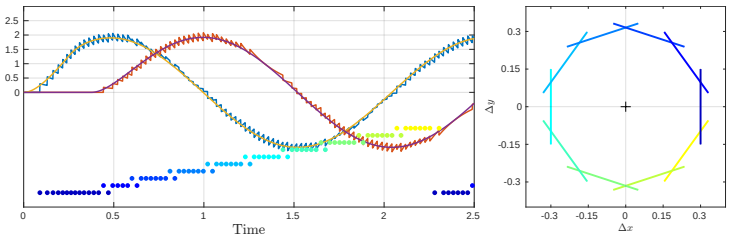


Figure 5.1.4: Simple 2D example with numerical solution and spike response. Curves for x in yellow/blue. Curves for y in purple/red. The networks output closely tracks the perfect numerical solution. For the network each decoding vector was chosen from a normal distribution and normalized to $\|\mathbf{\Gamma}_i\|_2 = 0.3$. Beneath a raster plot for each spiking neuron
On the left the threshold for each neuron's projected error.

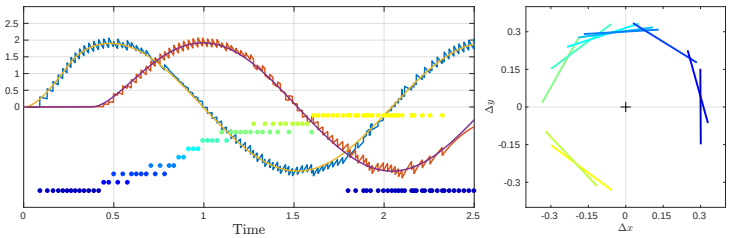
5.1.4 Importance of Feedforward/Decoding weights

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. 5.1.5. Here we simulate again a simple leaky integration as in eq. (5.1) however this time in 2D. With 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 ith column $\mathbf{\Gamma}$ added when the ith 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. 5.1.5 l to always be on top of the true trajectory of $[x,\ y]^T$, the network's error is just deviation from the origin.

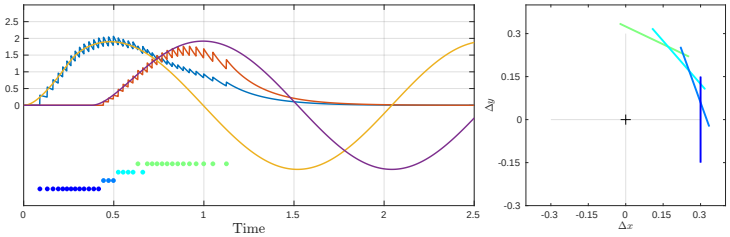
Explain phase plot explain the normal to the decoding weights Explain the 3 plots Dont forget to explain the Trichter in the bad plots



(a) Network



(b) Reduced Threshold



(c) Reduced Threshold

Figure 5.1.5: Network output after leaky integration of oscillating input with different decoding vectors.

5.1.5 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. 5.1.6 we simulated a linear n-mass spring system with the dynamics

$$m_i \ddot{x}_i = K \cdot \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{bmatrix} + c_i(t). \tag{5.2}$$

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.

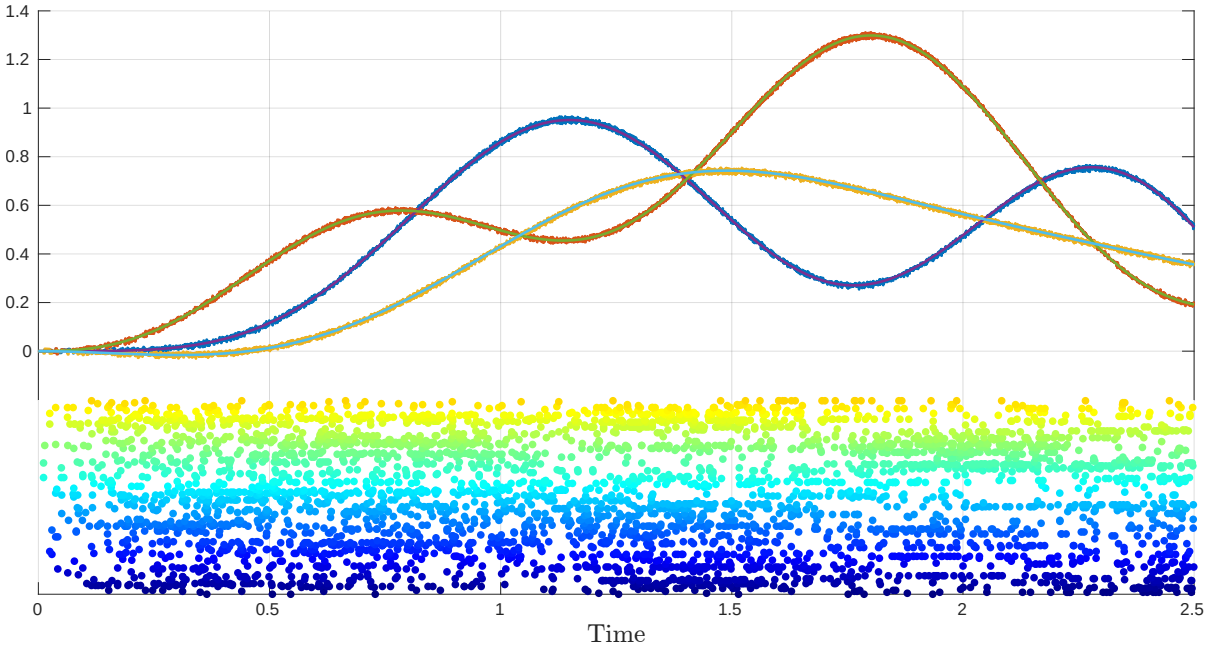
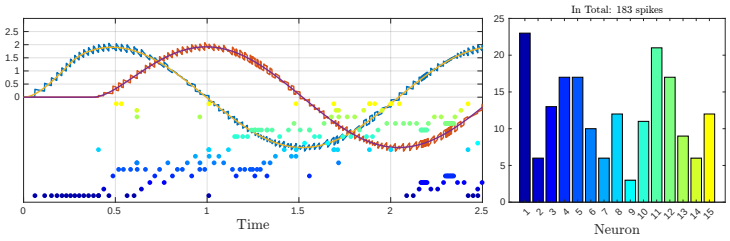
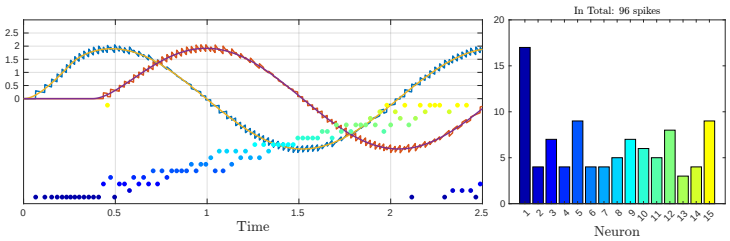


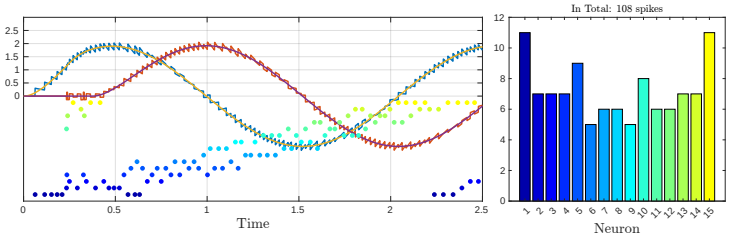
Figure 5.1.6: Trajectory of dynamic mass spring system with 100 masses and 2 thousand neurons. Only the first 3 trajectories are plotted and the spikes for the first 200 neurons.



(a) Network



(b) Reduced Threshold



(c) Reduced Threshold

Figure 5.1.7: Network output after leaky integration of oscillating input with different decoding vectors.

5.1.6 Varying cost parameters μ, ν

5.2 Results on the control

For the evaluation of the results for the control problem derived in section 3.3 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 3.3.2 and 3.3.4 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 3.4.

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

5.2.1 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 5.1.

```
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 5.1: 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 5.2.

```
1
2 for t_step = 1:N_step
3   V = update_Voltages(V,dt,Ws,Wf,c,...);
4   spiking_neurons = V>Threshold; % procudes a list
5   spikes(spiking_neurons,t_step) = 1;
6   V = V + Wf(:,spiking_neurons);
7   ...
8 end
```

Listing 5.2: 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 timestep 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 5.3

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

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 [31] in two ways. Firstly in the original derivation the rate is scaled by λ_d i.e.

$$\hat{r}(t) = \lambda_d r(t). \tag{5.3}$$

. Substituting this in for the equations in section 3.3.2 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

$$\begin{aligned}\dot{\mathbf{V}}(t) =& \boldsymbol{\Omega}^T\mathbf{B}^T\left(\dot{\hat{\mathbf{x}}}(t) - \dot{\mathbf{x}}(t)\right) - \mu\lambda_d\dot{\mathbf{r}}(t) \\ =& \boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{A}\mathbf{L}\mathbf{V}(t) \\ &+ \left(\mu\lambda_d\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{A}\mathbf{L} + \mu\lambda_d^2\mathbf{I} - \frac{1}{\lambda_d}\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{B}\boldsymbol{\Gamma}\right)\mathbf{r}(t) \\ &- \left(\boldsymbol{\Omega}^T\mathbf{B}^T\mathbf{B}\boldsymbol{\Omega} + \mu\lambda_d^2\mathbf{I}\right)\mathbf{o}(t) \\ &+ \boldsymbol{\Omega}^T\mathbf{B}^T(-\mathbf{A}\hat{\mathbf{x}}(t) + \dot{\hat{\mathbf{x}}}(t))\end{aligned} \tag{5.4}$$

where

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

is the pseudo-inverse of $\left(\mathbf{B}\boldsymbol{\Omega}\right)^T$. Our focus is now set on the terms in front of the rate. As [31] and the original derivation in [9] 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 λ_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 $\boldsymbol{\Gamma} = \mathbf{0}$ 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. 5.2.1. 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. 5.2.2 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. 5.2.1 and certainly cannot match the behaviour shown in [31].

Table Mit extra term ohne extra term Fehler in einem beispiel mit extra anzahl neuron

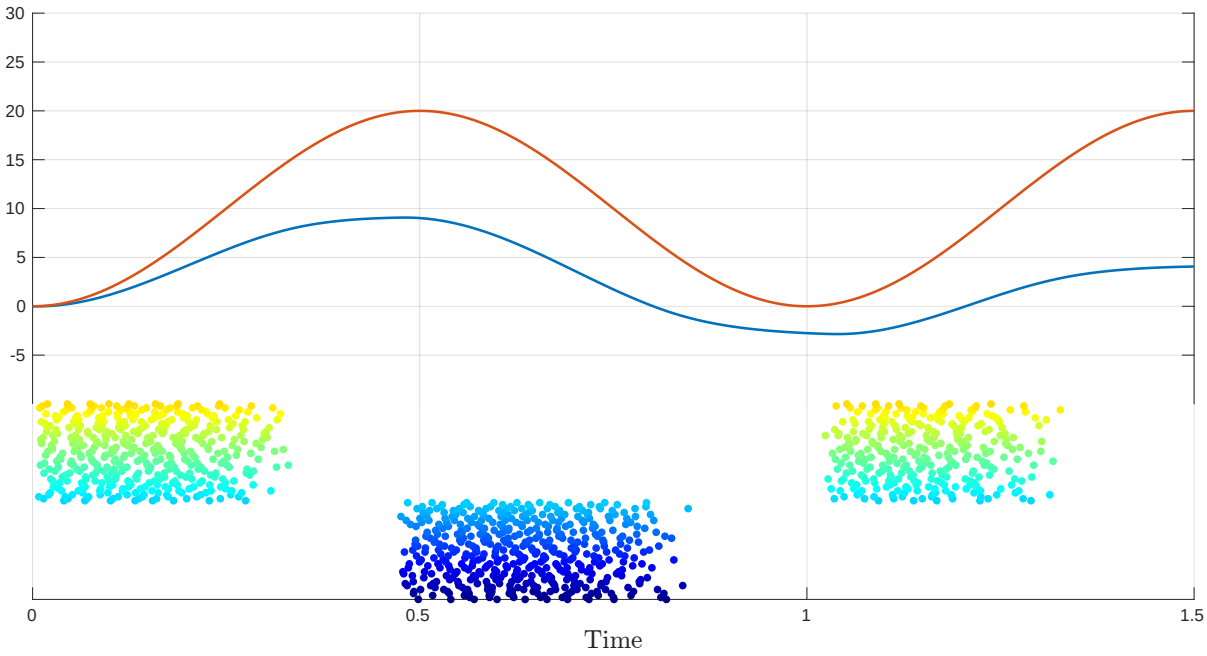


Figure 5.2.1: Control performance of the same the example in figure 2 from [31] without the extra $\lambda_d \mu^2 \mathbf{I}$ term. Simulation with $N = 100$ neurons.

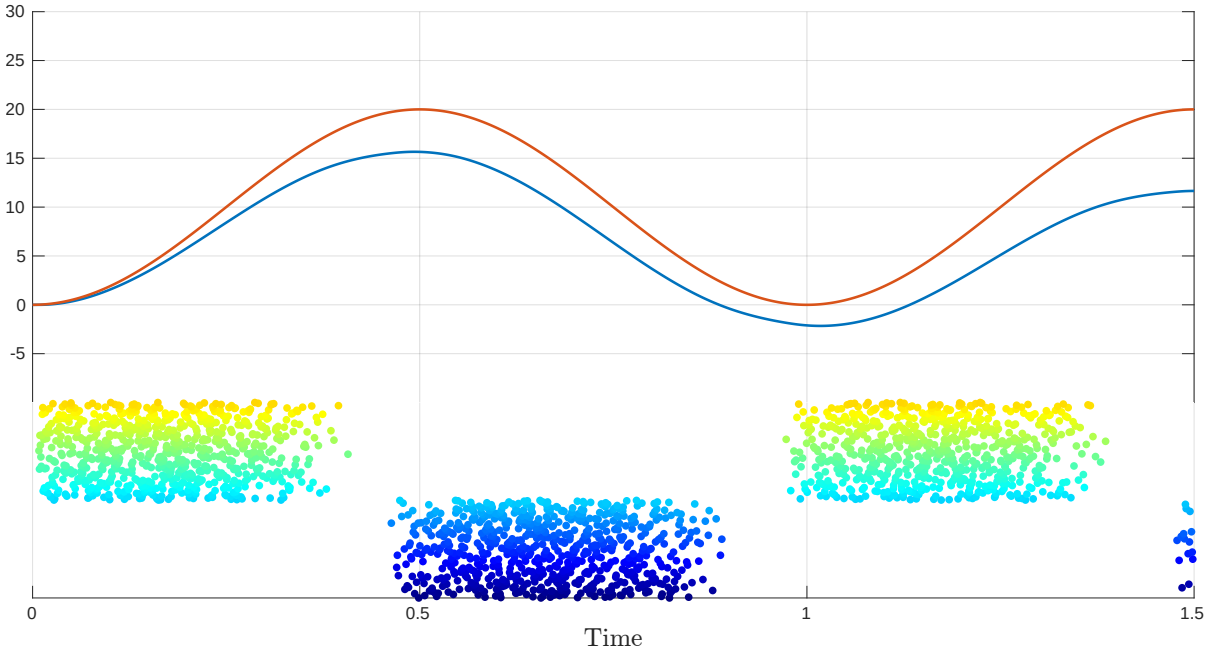


Figure 5.2.2: Control performance of the same the example in figure 2 from [31] without the extra $\mu \lambda_d^2 \mathbf{I}$ but $N = 500$ neurons to compensate.Neuron are split 50:50 as before.

die man hinzufuegen muss um ausgleich zu haben

Error sign The second important implementational detail is that concerns the error term in eq. (3.44) and the corresponding plots shown in [30]. Using the derivation found therein it was not possible to reproduce the results. After investigating the

problem was found to be in the sign of the error term. To illustrate this it is useful to consider their basic example of a scalar system

$$\dot{x} = -10x + u \tag{5.5}$$

but only two neurons with weights

$$\begin{aligned} \mathbf{\Omega} &= c \cdot [-1, 1] \quad c > 0 \\ \mathbf{\Gamma} &= [0, 0] \end{aligned} \tag{5.6}$$

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

$$V_2 = 1c\,(x - \hat{x}) < \frac{c^2}{2} \tag{5.7}$$

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

$$\begin{aligned} \dot{\mathbf{V}} &= \mathbf{\Omega B}^T \mathbf{A} e(t) + \dots \\ \dot{\mathbf{V}} &= -10c \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} e(t). \end{aligned} \tag{5.8}$$

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.

One way to resolve this issue is to add an artificial minus sign the error. However this only brings changes the problem to the case when $\mathbf{A} = 10$. The other solution is to interpret the phrasing of the original paper literally and assume $\mathbf{A}$ as a literal "gain" which therefore is usually greater than zero and take $|A|$ for the computation, yielding proper results. How either of these additions could emerge out of the derivation is so far unclear.

Only with either of these additions it was possible to reproduce the results show in the

ble check
xample
verify it.
tionally
be ask

paper [30].
However it is interesting to note that in different test scenarios it was found that the error term contributes only a small amount to the voltage in eq. (3.44) and therefore could be omitted in many cases directly. This holds also true for the example here where the omitted error makes the network less accurate but still usable compared to the method introduced in the original work which diverges rapidly.

Output Feedback How can I understand the output feedback? ASK ARVIND Useless Output feedback???

Adjustment for the Outputfeedback, even though it is useless

If you have output feedback you still need to give the whole state to the network so even though you have output, same goes for the derivative. So it is just useful if you want to give some output but not you know the whole system and the state so you can control. Which kind of makes it a bit useless.
Because this formulation does not allow e to be the error between network and reference. It has to be error between states. So my controller is just a greedy P controller?

5.2.2 Numerical treatment of spikes? Aka scale by 1/dt

Do it for the control signal but also for the reference signal (when there are jumps in the derivative for example)

5.2.3 Performance Comparison

Test1: kind of step function 1d with a couple neurons Test2: many dimensions random neuron decoders and some oscilating ref Accuracy # spikes Looks like they perform identical???

5.2.4 Limits

prob noise in the inputs? noise in the network B'C' = 0 bereits klar 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?

5.2.5 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 $\mathbf{y} = \mathbf{C}\mathbf{x}$, the derivation breaks down. 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?

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 PD 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 the derivative of the output

that we dont know the state

that the C is not square

5.3 Results on the learning

5.4 Results of the learned control objective

Chapter 6

<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.

6.1 Discussion

6.1.1 Future Work

find nonlinearity
end point control

6.1.2 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] 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).

[4] 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).

[5] Attwell, David **and** Laughlin, Simon B. "An Energy Budget for Signaling in the Grey Matter of the Brain". **in**: *Journal of Cerebral Blood Flow & Metabolism* 21.10 (**october** 2001), **pages** 1133–1145. ISSN: 0271-678X, 1559-7016. DOI:

10.1097/00004647-200110000-00001. URL: http://journals.sagepub.com/doi/10.1097/00004647-200110000-00001 (**urlseen** 24/11/2022).

[6] 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.

[7] 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).

[8] 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).

[9] 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).

[10] 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).

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

[12] 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).

[13] 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).

[14] 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).

[15] 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.

[16] 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).

[17] 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).

[18] 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).

[19] 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).

[20] 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).

[21] 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: 978-0-262-55060-4 978-0-262-05071-5.

[22] 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).

[23] 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.

[24] 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).

[25] 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).

[26] 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).

[27] 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).

[28] 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).

[29] 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).

[30] 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).

[31] 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.

[32] 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).

[33] 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).

[34] 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.

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

[36] 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).

[37] 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).

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

[39] 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).

[40] 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).

[41] Li, Weiwei **and** Todorov, Emanuel. "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems." **in**: *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics, (ICINCO 2004)*. **volume** 1. **january** 2004, **pages** 222–229.

[42] 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).

[43] 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).

[44] 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).

[45] 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).

[46] 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).

[47] 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).

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

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

[50] 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).

[51] 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).

[52] 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).

[53] 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).

[54] 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).

[55] Shu, Huailin **and** Pi, Youguo. “PID neural networks for time-delay systems”. **in**: *Computers & Chemical Engineering* 24.2 (**july** 2000), **pages** 859–862. ISSN: 00981354. DOI: 10.1016/S0098-1354(00)00340-9. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098135400003409 (**urlseen** 10/08/2023).

[56] 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).

[57] 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).

[58] 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).

[59] 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).

[60] 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).

[61] 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).

[62] 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).

[63] 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: 978-3-642-24964-8 978-3-642-24965-5. 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).

[64] 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).

[65] 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).

[66] 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.

[67] 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).

[68] 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