Think local, act global: robust and real-time movement encoding in spiking neural networks using neuromorphic hardware

A thesis submitted for the degree
"Doctor rerum naturalium"
of the Georg-August University of Göttingen

within the doctoral program Physics of Biological and Complex Systems (PBCS)
of the Georg-August University School of Science (GAUSS)

submitted by

Carlo Michaelis

born on 29th of September 1989 in Dieburg

Göttingen, November 2021

Thesis committee

Dr. Christian Tetzlaff
Third Institute of Physics - Biophysics, Georg-August University of Göttingen

Prof. Dr. Stefan Klumpp
Third Institute of Physics - Biophysics, Georg-August University of Göttingen

Prof. Dr. Alexander Gail
Sensorimotor Group, German Primate Center, Göttingen

Examination board

Dr. Christian Tetzlaff (*1st referee*)
Third Institute of Physics - Biophysics, Georg-August University of Göttingen

Prof. Dr. Stefan Klumpp (*2nd referee*)
Third Institute of Physics - Biophysics, Georg-August University of Göttingen

Prof. Dr. Alexander Gail
Sensorimotor Group, German Primate Center, Göttingen

Prof. Dr. Ulrich Parlitz
Max Planck Research Group Biomedical Physics, Max Planck Institute for Dynamics and Self-Organization, Göttingen

Prof. Dr. Alexander Ecker
Institute of Computer Science, Georg-August University of Göttingen

Prof. Dr. Fabian Sinz
Institute of Computer Science, Georg-August University of Göttingen

Date of the oral examination: 2022-01-13

Abstract

It is still a mystery how information is processed in the brain, dynamically and reliably at the same time, in particular when considering that a central nervous system operates mainly with information that is processed locally at and between single neurons. To analyze how movement could be represented in the brain, I aimed to find new approaches for solving basic neural encoding problems with local-only mechanisms, performed on the neuromorphic hardware Loihi. The theoretical hierarchical motor selection and execution (HMSE) model suggests biologically plausible mechanisms for movement encoding. A crucial issue for movement execution is the so-called stability-variability trade-off, the problem of obtaining enough variability in neural activity to provide sufficient information for learning complex movement trajectories, while keeping a sufficient level of robustness against noise at the same time. To achieve this, I analysed and successfully applied a new type of network which is based on an inhomogeneous connectivity structure, called anisotropic network. But even with stable and variable data at hand, it is still unclear how this information can be used to represent low-dimensional trajectories, like movements. Thus, I applied a reward-based supervised output learning mechanism, premised on the ReSuMe learning rule, that uses purely local synaptic mechanisms and allows learning of functions based on the spiking behavior of a network. In addition, most simulations were performed using the neuromorphic hardware Loihi. In order to enable the above mentioned simulations on this specialized hardware, I developed a variety of open source tools supporting neuroscientific experiments on Loihi, including a reservoir computing framework, an emulator for faster prototyping and a tool for translating parameters of existing neuron models to the Loihi neuron model. With these tools and frameworks at hand, I demonstrated that the neuromorphic chip Loihi can successfully be used for neuroscientific research. Overall, the results provide new insights into possible encoding schemes of biologically plausible spiking neural networks in general and suggest neural structures for movement encoding in particular.

Contents

List of abbreviations

AdEx	Adaptive-exponential integrate and fire
AM	Add-and-minimize
API	Application programming interface
BG	Basal ganglia
CPU	Central processing unit
CB	Cerebellum
CMOS	Complementary metal-oxide-semiconductor
DYNAP	Dynamic neuromorphic asynchronous processors
EPSP	Excitatory postsynaptic potential
ESN	Echo State Networks
HC	Hippocampus
HMSE	Hierarchical motor selection and execution
IoT	Internet of Things
ISI	Interspike interval
LIF	Leaky integrate-and-fire
LSM	Liquid State Machines
LTD	Long-term depression
LTP	Long-term potentiation
M1	Primary motor cortex
MSE	Mean squared error
NEF	Neural Engineering Framework
ODIN	Online-learning digital spiking neuromorphic processor
PCA	Principle component analysis
PPC	Posterior parietal cortex
preSMA	Pre supplementary motor area
PM	Premotor cortex

ReSuMe Remote supervised method

re-ReSuMe Reward-based remote supervised method

RNN Recurrent neural network

SMA Supplementary motor area

SoC System on a chip

SDK Software development kit

SNN Spiking neural network

SSSP Single source shortest path

STD Short-term depression

STDP Spike-timing-dependent plasticity

STP Short-term potentiation

Acknowledgements

I want to specifically thank Dr. Christian Tetzlaff for his supervision and support throughout my time as PhD student. I also highly appreciate the members of the thesis advisory committee Prof. Dr. Stefan Klumpp and Prof. Dr. Alexander Gail for their availability and helpful feedback in our meetings.

Special thanks goes to all members of our "Tetzlab" group. Everyone was always willing to lend an ear and ready to think and discuss any imaginable idea. Sincere thanks is given to Andrew B. Lehr. Many comprehensive, constructive and fruitful discussions with him were a significant factor for my work. Special thanks goes also to Dr. Jannik Luboeinski who supported me with helpful organizational advice, especially in the final phase of my thesis.

I also want to thank Winfried Oed, who invested a lot of time and effort to work together with us on the emulator and the tuning dashboard. Together with him, we obtained a deep understanding of the functionalities of the Loihi chip. Finn Schünemann and Benjamin Schulz did a lot of valuable experiments with the anisotropic network. Their work helped us a lot to obtain further insights into this promising type of network. Further was Elena C. Offenberg a very engaged student, helping us to develop a novel path finding algorithm for Loihi.

Thanks goes also to Ursula Hahn-Wörgötter who was always available for organizational questions and issues. The same holds for Thomas Geiling, who always helped to set up computer systems, especially the server for running Loihi. Furthermore, I want to thank Antje Erdmann and Frauke Bergmann for their continuous support within the PCBS program.

I want to thank Dr. Jannik Luboeinski, Andrew B. Lehr, Dr. Sebastian Schmitt, Winfried Oed and Elena C. Offenberg for feedback on parts of my thesis.

I finally want to lovely thank my wife Sarina Michaelis for her persistent and unlimited support.

Chapter 1

Introduction

> "Watching a child makes it obvious that the development of his mind comes about through his movements." Montessori (1959)

In 1922 Albert Einstein was honored with the Nobel Prize for his discovery of the law of the photoelectric effect, but surprisingly not for his relativity theory. At that time, the relativity theory was seen as speculative (Friedman, 2005), even though no one was able to prove its falseness for years and first experiments provided strong evidence for the theory (e.g. Dyson, Eddington, & Davidson, 1920). One important aspect of the special relativity theory is the interdependence of space and time which can mathematically be formulated in the so called Minkowski space (Minkowski, 1908). At that time, it was hard to imagine that time could be something different from Newton's absolute time. And still today, students intuitively may ask the question how time and space can be connected, since time feels more like a progress, independent from space. Even from our daily speech we are familiar with expressions like "time passes" or "time is running out", which is more related to our daily experience of body movement than to its mathematical or physical understanding. And indeed, experiments show that the time perception differs when subjects look at objects moving in different speeds (Brown, 1995). There is even evidence that time perception and the timing of motor actions is directly linked and processed in the same way (Treisman, Faulkner, & Naish, 1992).

The example of the connection between time and movement demonstrates that at least parts of our thinking are strongly connected to movement. But time is probably not the only type of perception that depends on movement. In recent years, the theory of *embodied cognition* was developed (for an introduction see Anderson, 2003; Shapiro, 2019), suggesting that our thinking depends essentially on our body, and in particular on its movements. The theory was born in 1979 with the elaborated work from Lakoff and Johnson (2008) who postulated that metaphors coming from direct physical experience are the basis for all abstract understanding in human thinking. A famous neuroscientific example for the relation between movement and cognition comes from educational science and shows that counting small numbers activates motor regions related to finger movements (Andres & Pesenti, 2015; Tschentscher, Hauk, Fischer,

& Pulvermüller, 2012). Gallese and Lakoff (2005) argue that abstract representations could be stored in sensory-motor areas in general. An overview of present theories describing this relation is given by Gentsch, Weber, Synofzik, Vosgerau, and Schütz-Bosbach (2016). In addition, there is rising evidence that the cerebellum processes not only the coordination of movements but is also important for many cognitive functions (Barinaga, 1996; Dum, Li, & Strick, 2002; Koziol, Budding, & Chidekel, 2011; Sokolov, Miall, & Ivry, 2017; Thach, 1996). Beside the fact that these hypotheses are quite young and first evidence is still sparse, these parallels indicate apparently that thought and movement are related to each other.

Therefore, we can certainly conclude that understanding movement is an important research topic not only because studying movement is an important and interesting subject for its own, but also especially due to its relation to cognition. A deeper understanding of how movement is processed in the brain, or biological neural networks in general, could lead to deeper insights of cognition. This understanding is helpful for issues like healing mental disorders or improving education, and applications like brain computer interfaces or autonomous machines.

The objective of the thesis

When analyzing movement, we need to be aware that movement is a highly complex field. Our body consists of about 200 bones (Neumann & Gest, 2019), which are connected with even more joints and controlled by more than 600 skeletal muscles (Saladin, 2017). This spans a high-dimensional movement space which wants to be controlled. The control is performed by the peripheral, as well as the central nervous system. In particular, numerous brain regions are linked to body movements. This includes the prefrontal cortex, the supplementary motor area, the primary and pre-motor cortex, the primary somatosensory cortex, the posterior parietal cortex, but also the hippocampus, the cerebellum & the basal ganglia (Krakauer, Hadjiosif, Xu, Wong, & Haith, 2019). With this, movement is not only a high-dimensional task, but in addition underlies a highly complex control.

This work intends to add a small piece of understanding to the complex phenomenon of movement. The research is performed with methods of computational neuroscience, which includes theoretical analysis and computational simulations. Moreover, neuromorphic hardware is used to perform most of the computational simulations. The goal of this work is to address several issues about how movement could be encoded in the brain. An overview over all results of this thesis is provided in Figure 1.1. First, I introduce a theoretical model which makes hypotheses about movement related functions, shown in purple within the overview figure. The proposed model is based on conclusions from experimental findings. Three core parts of this theoretical model, and therefore of movement in general, are implemented in spiking neural networks on the neuromorphic hardware Loihi (Davies et al., 2018). This includes the generation of robust but variable neural activity with an anisotropic connectivity structure and the learning of movement trajectories with a reward-based synaptic plasticity rule, both part of the main results of this thesis and colored in green within Figure 1.1. In addition, an appendix chapter suggests a homeostatic plasticity mechanism for the formation of cell assemblies, potentially important for

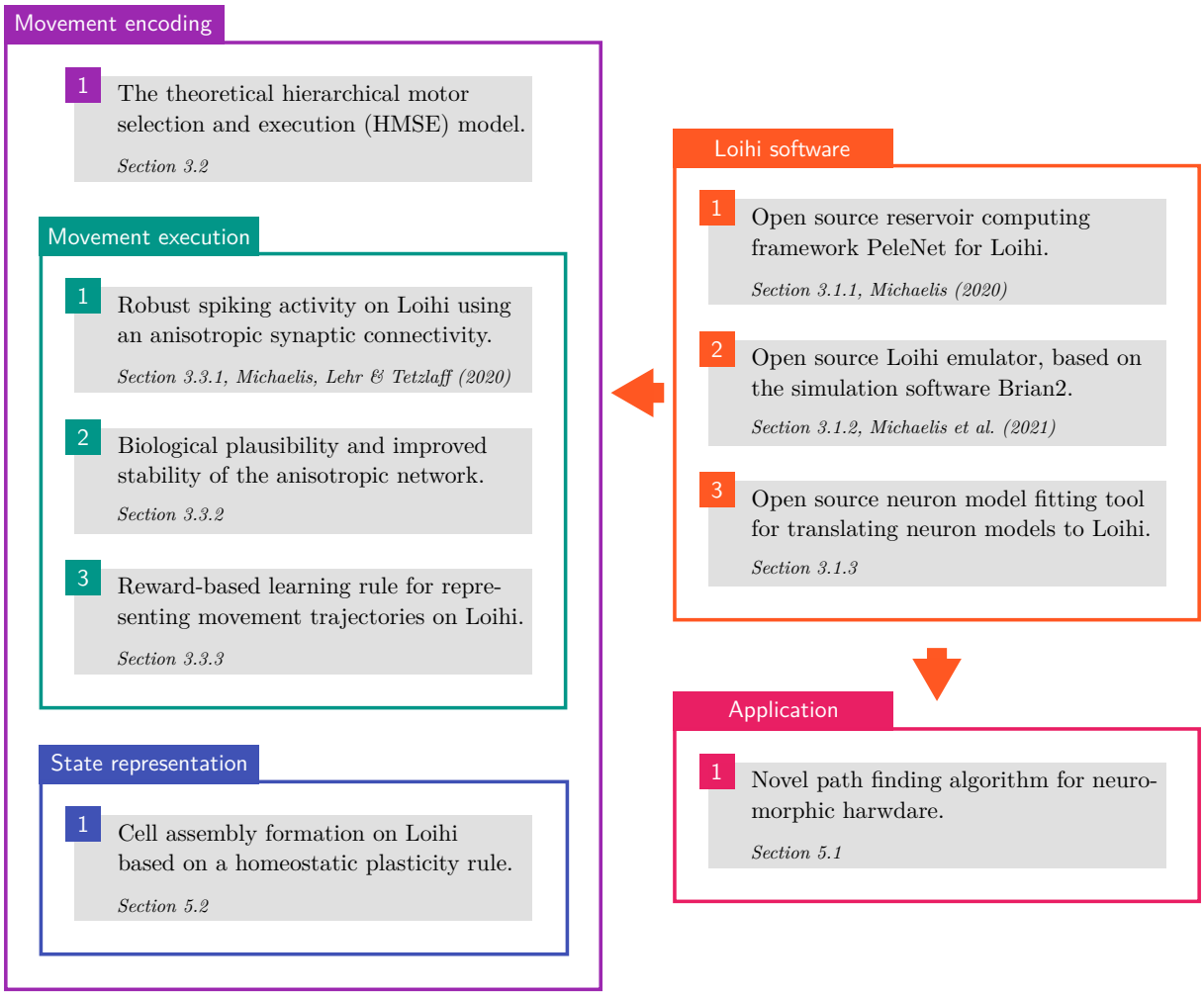


Figure 1.1: Overview of studies, related sections and publications in this thesis. The left side concerns the neuroscientifc research question of how movements are robustly encoded in the brain. The right side summarizes related results. **(orange)** Several frameworks and tools, developed to facilitate efficient neuroscientific research on Loihi. **(purple)** Movement encoding comprises all scientific studies which are based on the hierarchical motor selection and execution (HMSE) model. **(green)** The movement execution includes studies for a part of the HMSE model that contains the robust representation of trajectories, based on local mechanisms. **(blue)** The state representation is a general mechanism that is possibly important at many stages of the HMSE model. **(pink)** A novel single source shortest path (SSSP) algorithm that is designed for Loihi or similar neuromorphic hardware. This result is linked to Loihi and the frameworks, but not related to the movement encoding.

several issues of the theoretical model, highlighted in blue within Figure 1.1. An important feature of these solutions is that all of them base on purely local synaptic mechanisms and provide valuable insights for future research within neuroscience, which is addressed in the discussion. Note that the basic principles of how movement is performed plays on a more general level and is similar between different species (Bakken et al., 2020; Kaas, 2020). However, the human brain is the target model in this work.

A key problem for storing a highly dimensional trajectory is the necessity of variability within the data set. To provide sufficient information, the neural spiking activity cannot follow a repeated or even constant pattern. However, we still need a robust spiking activity, such that movements

can reliably be performed under noisy conditions. Specifically, the brain is in different states every time a movement is initiated. But the neural activity providing the information for the movement is still able to keep a sufficient level of stability. The requirement for high variation and stability at the same time results in a variability-stability trade-off, also referred to as robustness–flexibility in the literature (Pehlevan, Ali, & Ölveczky, 2018). Note that it is not required to keep this state forever, instead we only need this variable and stable activity for a sufficient period. A single movement action, part of a longer sequence, is assumed to be in a scale of hundreds of milliseconds to seconds (see e.g. S. A. Overduin, d'Avella, Roh, & Bizzi, 2008). But even with robust spiking activity at hand, it is still necessary to transform the high-dimensional spiking activity of the neurons into a meaningful output, describing the actual three dimensional movement trajectory. Since we are biologically restricted to mechanisms which are local-only with respect to the synapses, most traditional output training algorithms are not applicable. Note that this local-only restriction is also given by the neuromorphic hardware. Thus, using neuromorphic hardware forces us to consequently avoid non-local mechanisms at all costs.

As a tool to approach solutions for the described issues, I had the opportunity to make use of the digital neuromorphic research chip Loihi (Davies et al., 2018) provided by Intel. Neuromorphic hardware promises fast and energy-efficient simulations of large spiking networks (Blouw, Choo, Hunsberger, & Eliasmith, 2019; Tang, Shah, & Michmizos, 2019). Despite these advantages, the first generation of the Loihi chip has still limitations, considering a slow read-out, slow initialization times and limitations in the way synaptic connections can be established. However, I will demonstrate that computational models for neuroscientific research can successfully be implemented and algorithms for technological use cases, like robotic control and path finding algorithms, can successfully be developed.

In summary, this work aims to show that a biologically plausible approach with pure local mechanisms can be used for encoding movements in spiking neural networks. In addition, the research demonstrates that these mechanisms can be implemented on today's neuromorphic hardware by utilizing several tools and software libraries developed for Loihi. The here presented theoretical model provides a clear picture of the current understanding of processes underlying movement execution and selection and suggests in particular concrete mechanisms on a neural level for movement execution. Using these insights, experimenters can design specific studies that potentially reveal even deeper insights into the basic mechanisms and their interactions. Further, robotic research groups can make use of the neuroscientific results and the concrete open source software packages, both provided within this work, to build efficient algorithms based on neuromorphic hardware.

The structure of the thesis

The thesis starts with an outline about neurons and neural networks, from a biological and a computational perspective. I will first summarize what is already known about movement. In particular, which brain regions participate, how they interact and which part takes which role.

Further, neuromorphic hardware and specifically the neuromorphic research chip Loihi is introduced. In a first results chapter, I will present software libraries and tools which were developed to facilitate the work with Loihi. This includes a framework for reservoir computing (Michaelis, 2020), an emulator for Loihi, written in `Brian` (Michaelis, Lehr, Oed, & Tetzlaff, 2021) and a tuning tool for translating existing neuron models to Loihi. After the frameworks and tools are described, a results chapter introduces the so-called theoretical hierarchical motor selection and execution (HMSE) model that makes hypotheses about the generation of movements and provides a big picture for the following subsequent sections. The third part introduces a network with an anisotropic connectivity structure (Spreizer, Aertsen, & Kumar, 2019) which is used to overcome the variability-stability trade-off and provides an algorithm for robotic control (Michaelis, Lehr, & Tetzlaff, 2020). The anisotropic network is further explored and optimized and a surprisingly well match between the simulation results and experimental data is demonstrated. Finally, the reward-based remote supervised method (re-ReSuMe) is introduced, based on the original ReSuMe rule from (Ponulak, 2006; Ponulak & Kasiński, 2010). This reward-based plasticity rule allows learning output functions, based on random network activity. The re-ReSuMe learning rule provides a local-only mechanism, which is not only more biologically plausible than most existing approaches, but also efficient for neuromorphic hardware. In addition to the main results chapter, two further research results are described in the appendix. First, a novel algorithm for the single source shortest path (SSSP) problem is provided that is designed for spiking neural networks in general and neuromorphic hardware in particular. Second, a synaptic plasticity mechanism is described which allows the self-organized formation of cell assemblies. Finally, the anisotropic network and the re-ReSuMe approach are discussed and embedded in the context of the HMSE model. Alternative approaches are evaluated and open questions for future work are addressed, which mainly includes the selectional level of the HMSE model and therefore the ability to arbitrarily concatenate movement actions to longer sequences. The discussion also contains an overview over advantages and currently existing limitation of neuromorphic hardware, especially Loihi.

Chapter 2

Background

In this chapter, the ingredients of neural networks from a biological as well as from a computational perspective are introduced. I further review the current understanding of movement generation and which brain areas participate in this process. Finally, the main principles of neuromorphic hardware are explained, current chips are compared and an overview over the neuromorphic research chip Loihi is provided.

2.1 Neural networks

> "It was as though the scales had fallen from my eyes. I asked myself the question: But why do we always look for anastomoses? Could not the mere intimate contact of the protoplasmic processes of the nerve-cells effect the functional connection of nervous conduction just as well as absolute continuity? [...] All the data supported the theory of simple contact." (August Forel, 1937, as cited in Finger, 2001)

A *neuron* is a cell type which occurs in all eumetazoa and therefore in most multicellular organisms. Neurons can process and transmit electrical and chemical signals. They are interconnected by so-called *synapses*. Being connected, they form a neural network which is then used by the organisms to process information. Here, the biological background of neurons and neural networks are introduced and theoretical models are defined[1].

[1] Some few parts within this chapter base on the "Neurons and synapses" and "Self-organizing recurrent neural network" sections of my master thesis from 2018, entitled "Does intrinsic plasticity prefer regularity? A Markov chain Monte Carlo perspective on a self-organizing recurrent neural network".

2.1.1 Neurons & synapses

The Nobel Prize in 1906 was given to Camillo Golgi and Santiago Ramón y Cajal. The two researchers performed histological studies about the nervous system and could demonstrate two main properties of the nervous system: the functional localization and the so-called neuron doctrine (Glickstein, 2006). The functional localization describes the idea that different parts of the nervous system are responsible for different tasks. In addition, the neuron doctrine assumes that the nervous system is build out of "atomic" elements, called neurons. The idea of independent neural elements was firstly hypothesised by August Forel (see quote above). Some years later, Wilhelm von Waldeyer-Hartz coined the term *neuron* for these "atomic" neural cells (Waldeyer, 1891). In 1897, Charles Scott Sherrington suggested the term *synapse* for the connections between the neurons (Sherrington, 1897; Tansey, 1997). At that time the fundamental building blocks of today's neuroscience were developed. It took about half a century until the first comprehensive mathematical description of a neuron was developed by Alan L. Hodgkin and Andrew F. Huxley (Hodgkin & Huxley, 1952), honored with a Nobel Prize in 1963. It took some further decades until computational neuroscience was established, with first large-scale simulations of neural networks in the 1980s (reviewed by Bower, 2013; Fan & Markram, 2019). Over time the understanding of the nervous system was refined and became more precise, but still many aspects of neurons and synapses are unknown.

Structure of a typical neuron

A typical neuron consists of:

- *Soma* (cell body): The main part of the cell, containing the organelles. It contains in particular the *nucleus* which incorporates the *genome* and causes the majority of the protein synthesis of the cell.
- *Dendrites*: Branched projection from the soma. These projections receive information from other cells via their *dendritic spines*.
- *Axon*: Long slender projection from the soma. The junction between soma and axon is called *axon hillock*. The terminations of the axon branches are denoted as *axon terminals*.

The axon terminals of one neuron are connected to another neuron's dendritic spines. Between the axon terminal and the dendritic spine is a small gap, called *synaptic cleft*. All three parts together form the *synapse*. Figure 2.1 contains a schematic representation of three neurons, where soma, dendrites and axon are indicated. The synapses are highlighted by circles.

Action potential of a typical neuron

According to Deetjen, Speckmann, and Hescheler (2005), a typical cell has a membrane potential of about $-70\,mV$ at its axon hillock, it is called the *resting potential*. In principle, if the cell gets some electrical stimulation, the permeability of voltage-gated ion channels in the membrane of the neuron are changed. If the potential at the membrane reaches the *threshold potential* of at least $-55\,mV$, the *depolarisation* of the membrane potential starts. Within this phase, Na^+ ions pass from the outside of the axon membrane to the inside. This self-enforcing process brings the potential up to about $40\,mV$, which is then called the *action potential* or *spike*. Afterwards K^+ ions move from the inside of the axon membrane to the outside, which causes the potential to drop again within the so-called *repolarization*. Finally, the original equilibrium is reached again within the *refractory period*. Directly after an action potential was fired from the neuron, there is a short period, where the membrane is not excitable at all, the so-called *absolute refractory period*, which equals a threshold of infinity. After that, there is a period where the neuron is excitable again, but where the threshold is higher than normal, denoted as *relative refractory period*.

When an action potential is triggered within a neuron, the action potential spreads over the whole neuron, mainly along the axon. The propagation can either be a *saltatory conduction*, when the axon is covered with *myelin sheaths*, or *continuous conduction* for unmyelinated axons. It then reaches the axon terminals and induces a postsynaptic potential at the post-synaptic neurons via the synapses. The threshold potential decides if the postsynaptic activation is enough to elicit a spike at the post-synaptic neuron or not. With this, signals are transmitted binary and the underlying principle is called *all-or-none law*.

The first complex mathematical model of a neuron's spiking behavior was formulated by Hodgkin and Huxley (1952). The idea was to model the cell membrane of a neuron as an electrical circuit. The cell membrane acts as a capacitor, whereas the channels behave like a resistor. If an action

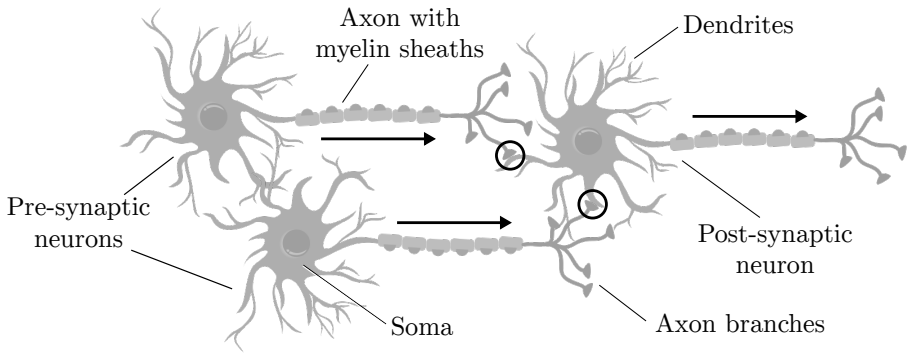


Figure 2.1: Three neurons showing the typical structure with soma, axon and dendrites. The synapses, containing the axon terminal of the pre-synaptic neuron, the dentritic spine of the post-synaptic neuron and the synaptic cleft, are highlighted with circles. The direction of the conduction is shown by arrows. Adapted from: *Set Of Neuron Vector.* (2021, August 05). Retrieved from https://www.vecteezy.com/vector-art/115973-set-of-neuron-vector

potential is arriving at the membrane, it increases the current. The process can be described by

$$C \cdot \frac{dv}{dt} = I - (I^{\mathrm{K}} + I^{\mathrm{Na}} + I^{\mathrm{L}}), \tag{2.1}$$

where I is an external current (e.g. an incoming action potential). v is the potential of the membrane and C the membrane's capacity. I^{K} & I^{Na} are leaky currents from voltage-gated ion channels for potassium & sodium. I^{L} is a leaky current from an unspecific channel. The channels are modeled by

$$I_i = g_i(v - v_i), \tag{2.2}$$

where $i \in \{\mathrm{K}, \mathrm{Na}, \mathrm{L}\}$. g_i is the conductance and v_i the *reversal potential* of the channel.

In the following years and decades, several different neuron models were developed (reviewed e.g. by Long & Fang, 2010). The goal was either to model more details of a neuron to be more biological plausible or to simplify the neuron model in order to decrease computational complexity. In this work, I will focus on the *leaky integrate-and-fire* (LIF) model, since it is an established candidate for conducting larger network simulations of several hundred or thousand neurons being largely plausible at the same time. It is very similar to the Hodgkin–Huxley model, but does not distinguish between different ion channels. According to Gerstner, Kistler, Naud, and Paninski (2014), we can define the voltage behavior of a leaky integrate-and-fire (LIF) neuron as

$$\tau_v \frac{dv}{dt} = -\big(v(t) - v_{\mathrm{rest}}\big) + RI(t), \tag{2.3}$$

where $\tau_v = R \cdot C$ is a time constant, defined by the ion channel resistance R and the membrane capacity C. The time constant causes an exponential decay of the membrane voltage v to the *resting potential* v_{rest}. $I(t)$ is an external current, which can for example be caused by an action potential.

In addition to the description of the voltage behavior, it is important to also model the spike behavior. For this we define a spike time t_k as an event at which the voltage reaches a threshold v^{th}, viz.

$$t_k : \quad v(t_k) = v^{\mathrm{th}}. \tag{2.4}$$

With this, we can define a spike train of a neuron by

$$\sigma(t) = \sum_k \delta(t - t_k). \tag{2.5}$$

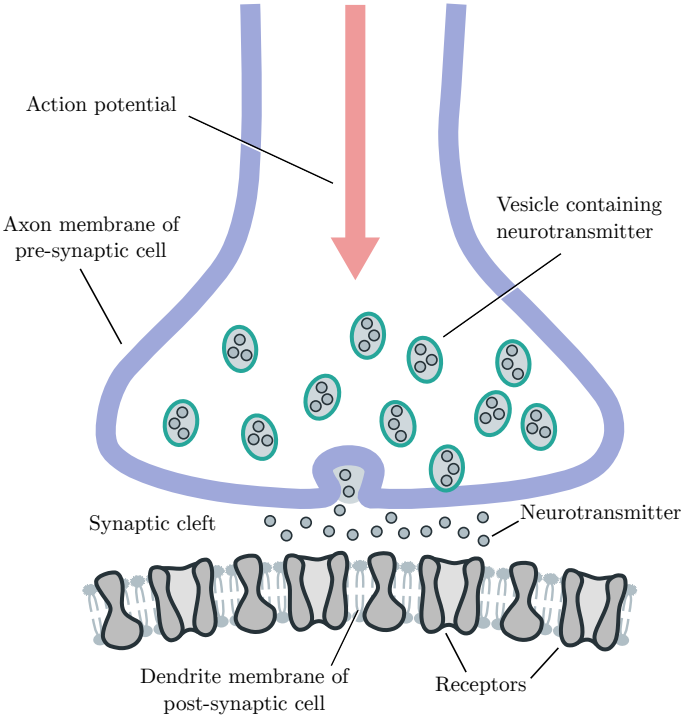


Figure 2.2: Structure of a typical chemical synapse. An incoming action potential causes vesicles to move to the synaptic cleft and release the containing neurotransmitter. The neurotransmitter connects to the receptors in the membrane of the post-synaptic neuron. This causes a flow of ions through the membrane of the dendritic spine, which changes its potential and may trigger a spike within the post-synaptic neuron.

Structure and behavior of a typical synapse

As described above, a synapse consists of the axon terminal of the pre-synaptic neuron, the dendritic spine of the post-synaptic neuron and the synaptic cleft between those two. Electrical signals are transferred from the *pre-synaptic membrane* to the *post-synaptic membrane*. A sketch is shown in Figure 2.2.

Synapses can occur as a *chemical synapse* or as an *electrical synapse*. Chemical synapses transmit signals via chemical substances, called *neurotransmitters*. Neurotransmitters are necessary since the gap between the pre-synaptic and post-synaptic cell is about $12\,nm$ to $20\,nm$ (Savtchenko & Rusakov, 2007) and cannot be bridged by electrical activation. The neurotransmitters need some time to reach the post-synaptic cell. Simulations show that it takes about $0.5\,ms$ until 50% of the distributed transmitters receive the post-synaptic neuron, assuming a synaptic gap of $20\,nm$ (Clements, 1996). Whereas electrical synapses are nearly directly connected with a gap of just $2\,nm - 3\,nm$ (Peracchia, 1977). They are connected by gap junction proteins, called *connexins*, and can transmit electrical signals directly. In those synapses nearly no delay occurs in transmission compared to chemical synapses.

In chemical synapses the axon terminal contains the neurotransmitter, collected in *vesicles*. When an action potential arrives at the synapse, the vesicles move to the pre-synaptic mem-

brane. The vesicles merge with the membrane and the neurotransmitters are released into the *synaptic cleft*. At the post-synaptic membrane are receptors and ion channels, often joined to a *receptor-channel-complex* and called *ionotropic receptor*. In this case, the neurotransmitter binds to the ion channel and changes its structure which allows an exchange of ions. Since the neurotransmitter operates like a key opening a lock, where the lock is the ion channel, this mechanism is called *key-lock principle*. A consequence of the joined receptor-channel-complex is a relatively fast reaction time (Deetjen et al., 2005, p. 40). In the other case, receptor and ion channels are separated and the receptor is called *metabotropic receptor*. The neurotransmitter connects to a receptor, but this time the receptor releases a messenger substance called *second messenger*. This second messenger finally opens an ion channel. This process is slower, since it takes some time until the reaction inside the post-synaptic cell is ready and the ion channels are opened.

Inhibitory and excitatory neurons

The potentials, reaching the post-synaptic neuron via the synapses, can cause *inhibitory* or *excitatory* effects on the post-synaptic side. Inhibitory synapses reduce the potential and decrease the chance to reach the threshold potential that leads to an action potential. They mainly operate with GABA (γ-aminobutyric acid), glycine, serotonin or dopamine as neurotransmitter. Excitatory synapses raise the potential and therefore increase the chance to trigger an action potential. Those synapses mainly contain glutamate, acetylcholine or noradrenalin. Both types of synapses have different chemical mechanisms to change the potential. Within this thesis, we will not consider a more detailed process, but it is, however, important to be aware of those two types.

According to *Dale's principle*, a neuron has either only inhibitory or only excitatory axonal synapses (Dale, 1935; Eccles, 1976). There are exceptions from this rule, but the principle holds for most neurons (Sossin, Sweet-Cordero, & Scheller, 1990). We denote a neuron with inhibitory outgoing synapses as *inhibitory neuron* and a neuron with excitatory synapses as *excitatory neuron*.

2.1.2 Plasticity & reward

William James coined the term *plasticity* in 1890 in his book *Principles of Psychology*, despite the fact that he was not aware of the neural doctrine or the concept of synapses (Berlucchi & Buchtel, 2008). Donald O. Hebb was the first describing a mechanism underlying neural plasticity in *The Organization of Behavior* (Hebb, 1949): the correlation of pre- and post-synaptic firing causes a change in the efficacy of the synapse. The term neural plasticity became widely accepted in the 1970s, a first clear definition was given by Jacques Paillard (Paillard, 1976; Will, Dalrymple-Alford, Wolff, & Cassel, 2008). While it was long believed that most structures in the adult brain are "fixed" and only young brains are plastic to a larger extent (see

e.g. Sylva, 1997), today we know that a variety of plasticity mechanisms exist in every stage of age (Fuchs & Flügge, 2014; Huttenlocher, 2009).

Plasticity

Plasticity or more specific *neuroplasticity*, is the ability of the brain to change its neuronal structure depending on experience. It can broadly be separated intos *functional plasticity* and *structural plasticity*.

- Functional plasticity: Changes in synaptic transmission due to changes in amount of neurotransmitter or receptors.
- Structural plasticity: Changes in structure of the neuron, like growing or shrinking of axons, dendrites, synapses or even whole neurons.

Furthermore, *synaptic plasticity* describes all processes related to synapses, like changes in transmission characteristics and growing or shrinking of synapses. Regarding synaptic plasticity one can distinguish between *pre-synaptic plasticity* and *post-synaptic plasticity*. Pre-synaptic plasticity describes a process in which the amount of released neurotransmitters or the time for readmission of neurotransmitters changes. Post-synaptic plasticity changes the sensibility for a given amount of neurotransmitters. For example the amount of receptors and ion channels can be adapted to achieve post-synaptic plasticity. Furthermore, plasticity can vary in duration. *Short-term plasticity* changes the transmission characteristic of the synapses in a scale of milliseconds to minutes, while *long-term plasticity* has an effect for hours, or even years. Both can be separated in so-called depression – short-term depression (STD) and long-term depression (LTD) – which means a decrease in connectivity, and potentiation – short-term potentiation (STP) and long-term potentiation (LTP) – which means an increase in connectivity between the synapses.

A rule that first postulated a functional plasticity mechanism is the *Hebb rule*. Biologically, synapses strengthen their connection if one neuron j causes action potentials in another neuron i (Bliss & Lømo, 1973; Hebb, 1949). In that case the weight between those neurons, modelled with w_{ij}, will increase. This can be expressed by

$$\Delta w_{ij} = \eta \cdot x_j \cdot y_i, \tag{2.6}$$

where η is a learning rate. x_j is the activity (e.g. spike rate or eligibility trace) of the pre-synaptic neuron and y_i the activity of the post-synaptic neuron.

A causal type of Hebb's rule is the so-called *spike-timing-dependent plasticity* (STDP), which is widely used in models of spiking neural networks (SNN). This rule depends on temporal correlations between the spikes of pre-synaptic and post-synaptic neurons (Sjöström & Gerstner, 2010). If pre-synaptic spikes arrive shortly before an action potential of the post-synaptic neuron,

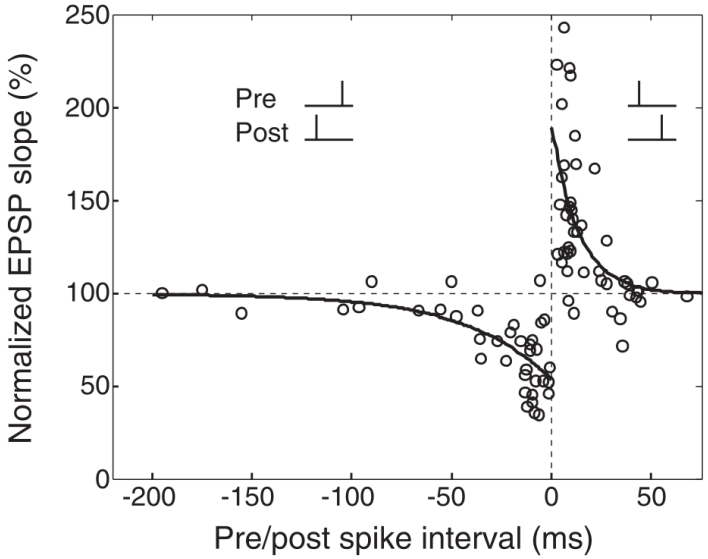


Figure 2.3: Spike-timing-dependent plasticity (taken from Dan & Poo, 2006, Figure 1). The closer the post-synaptic neuron spikes before the pre-synaptic neuron spikes, the higher will be the *decrease* in weight and vice versa.

it leads to long-term potentiation. In case that pre-synaptic spikes arrive shortly after an action potential of the post-synaptic neuron, it leads to long-term depression. The pre- and post-synaptic spikes have only a specific time window in which effects occur. This time window depends on the type of synapses. In most cases it is of the order of milliseconds (Dan & Poo, 2006). An illustration of STDP from experimental data is shown in Figure 2.3.

To model STDP we assume spikes at the post-synaptic neuron, collected in a set $l \in \{1, ..., n_l\}$, where n_l is the total number of spikes. Let then t_i^l indicate the time for every spike at neuron i. Furthermore, the spikes of the pre-synaptic neurons are in the set $k \in \{1, ..., n_k\}$, where n_k is the total number of pre-synaptic spikes. t_j^k indicates the points in time at which the pre-synaptic neuron j is firing. The distance between a pre- and a post-synaptic spike can be defined as $\Delta t = t_j^k - t_i^l$. With this and according to Sjöström and Gerstner (2010), we can model STDP as

$$\Delta w_{ij} = \sum_{k=1}^{n_k} \sum_{l=1}^{n_l} f(\Delta t) = \sum_{k=1}^{n_k} \sum_{l=1}^{n_l} f(t_j^k - t_i^l), \tag{2.7}$$

where the function $f(\Delta t)$ is a *learning window*. In general, the learning window can be defined in various ways, some are shown in Figure 2.4. Since an exponential decay seems to fit experimental data best in most cases (L. I. Zhang, Tao, Holt, Harris, & Poo, 1998), it is often chosen as an exponential function (Sjöström & Gerstner, 2010; Song, Miller, & Abbott, 2000) of the form

$$f(\Delta t) = \begin{cases} A_+ \exp(\Delta t/\tau_+), & \Delta t < 0 \\ -A_- \exp(-\Delta t/\tau_-), & \Delta t \geq 0 \end{cases}, \tag{2.8}$$

where A_+ and A_- are the amplitudes and τ_+ and τ_- are time constants. This mathematical

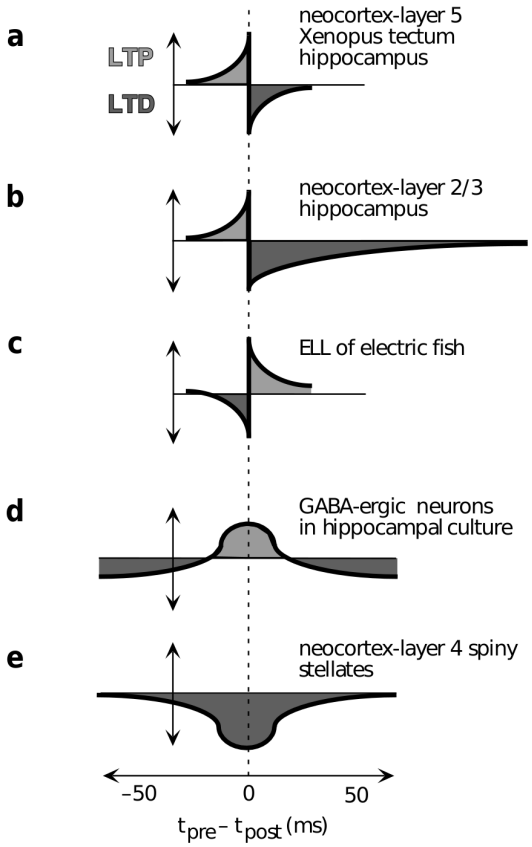


Figure 2.4: Spike-timing-dependent plasticity (STDP) learning windows (taken from Abbott & Nelson, 2000, Figure 2).

description forms the foundation for implementations of STDP applied throughout simulations described within this thesis.

Reward

Pavlov (1927) first described that a reinforcing stimulus can be used to strengthen a link between a stimulus and a behavior. Dogs were trained to connect an auditory stimulus with food. While this was a typical experiment for the black box approach (Chen, 2005) of the behaviorist, in the following decades with improved methods, scientists found pathways in the brain related to reinforcement[2]. In particular, dopamine was identified as the most important neurotransmitter in this process. The most popular hypothesis assumes that neurons in the midbrain encode a reward prediction error which guides learning in the frontal cortex and the basal ganglia (Glimcher, 2011).

[2]We need to distinguish two meanings of reinforcement, since it can cause confusions depending on the readers background. In a context of behavioral psychology, it generally describes the strengthening of an animal's behavior. In the context of machine learning or neural network simulations it describes an adaptation process that is based on maximizing reward. Certainly, both meanings are conceptually related, but it is still important to be aware that two meanings without explicit indication are used.

However, we still need to get an understanding of what happens on a cellular level to be able to model reward processes. First, it is important to distinguish between a so-called *wired transmission* (sometimes also referred to as point-to-point transmission) and a *volume transmission* of neurotransmitters (Agnati, Guidolin, Guescini, Genedani, & Fuxe, 2010). A wired transmission describes a transmission of neurotransmitters between synapses, whereas a volume transmission means that neurotransmitters are emitted into the extracellular space of the neurons. The dopamine triggering reward mechanism is emitted primarily to the extracellular space and therefore follows a volume transmission of neurotransmitters (Bamford, Wightman, & Sulzer, 2018; Rice, Patel, & Cragg, 2011). Finally, dopamine is emitted by specialized dopamine neurons that perform the prediction error, mentioned above, and facilitate reinforcement learning strategies (Pignatelli & Bonci, 2015).

Considering two neurons connected by a synapse, a reward signal modulates the plasticity at the synapse and can be described as a third factor, in addition to the pre- and post-synaptic spike behavior. Since dopamine is transmitted via the extracellular space, it influences the plasticity process from outside of the synapse and does neither belong to the axon terminal, nor to the dendritic spine. It was experimentally shown that neuromodulators modulate the shape of the STDP learning window (Pawlak, 2010). These ingredients can be combined in a three-factor rule, which is in the following adapted from Gerstner et al. (2014). First the window function $f(\Delta t)$, defined above in Equation 2.8, is used for an *eligibility trace* which enables a possible weight change if the eligibility is greater than zero. The eligibility trace is increased when pre- and post-synaptic spikes are close together. As in other processes before, also this trace decays exponentially. It can be defined as

$$\frac{d}{dt}e_{ij} = \frac{1}{\tau_e} \cdot \big(- e_{ij} + f(\Delta t)\big), \tag{2.9}$$

where τ_e is the time constant for the decay of the eligibility trace. The weight change then depends on the value of the eligibility trace e_{ij}, the spike times and the window function $f(\Delta t)$ and a reward r_{ij} according to

$$\frac{d}{dt}w_{ij} = r_{ij} \cdot f(\Delta t) \cdot e_{ij}. \tag{2.10}$$

Note that the reward r_{ij} is not necessarily directly a reward, but can also be more general a neuromodulator, that in turn depends on a reward. A common choice is that r_{ij} depends on the actual reward minus the expected reward (Gerstner et al., 2014).

Reward modulated STDP rules were applied in different variants (Baras & Meir, 2007; Răzvan V. Florian, 2007; E. M. Izhikevich, 2007; Legenstein, Pecevski, & Maass, 2008) and are candidates for machine learning approaches with spiking neural networks (Bing et al., 2019; Mozafari, Ganjtabesh, Nowzari-Dalini, Thorpe, & Masquelier, 2019; Mozafari, Kheradpisheh, Masquelier, Nowzari-Dalini, & Ganjtabesh, 2018; T. Zhang, Jia, Cheng, & Xu, 2021).

2.1.3 Spiking neural networks

In 1957, Frank Rosenblatt introduced the *perceptron* (Rosenblatt, 1958), which is a simplified neural network based on the simplified McCulloch-Pitts neuron model, introduced some years before (McCulloch & Pitts, 1943). In the early years, there was a rapid development of artificial networks, used for technological purposes as well as for neuroscientific modelling of brain processes (Haenlein & Kaplan, 2019). But in the 1970s the development got stuck in theoretical and computational challenges, which was especially the case for technological applications. While the simulation of neuroscientific models started to emerge with computational neuroscience in the 1980s and 1990s (Bower, 2013; Fan & Markram, 2019), the technological application of artificial neural networks started its rise as late as in the late 2000s (Haenlein & Kaplan, 2019).

Due to computational simplicity it was long common to simulate biological neural networks with a rate code, but improvements in hardware systems and software frameworks allowed more complex network simulations. Moreover, from a biological standpoint came growing evidence that a code that takes temporal order into account is important for many processes in the brain (Gautrais & Thorpe, 1998; Rullen & Thorpe, 2001). Wolgang Maass coined the term of "The third generation of neural network models" for spiking neural networks, considering temporal order (Ghosh-Dastidar & Adeli, 2009; Maass, 1997). These spiking neural networks evolve today to a state-of-the-art model for simulating large biological neural networks (Khadeer Ahmed, 2021) and are considered as a potential replacement for classical artificial neural networks (see e.g. Lee, Delbruck, & Pfeiffer, 2016; Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2019).

Spatial and temporal summation

The incoming signals from all pre-synaptic neurons – which can be several thousands (Schandry, 2006, p. 89) – sum up to an overall potential at the soma of the post-synaptic neuron. The summation can be spatial and temporal. If many synapses are active at the same time from different pre-synaptic neurons, the potentials sum up spatially, called *spatial summation. Temporal summation* describes a process where at one synapse many signals arrive in a row and sum up to a higher overall potential. The degree of temporal summation depends on the time constants of the synapses and neurons. In some synapses it is necessary that the signals must be closer than $20\,ms$ to have an additive effect, in some synapses a gap of $1\,s$ can still influence the potential (Schandry, 2006, p. 92).

We can model the summation of inhibitory and excitatory signals from source neurons j to a target neuron i as a sum of incoming spikes by

$$I_i(t) = \sum_j J_{ij} \sum_k s(t) H(t - t_{j,k}), \tag{2.11}$$

where $H(t - t_{j,k})$ is the unit step function which is 1 if a source neuron j fires a spike k at time t, the step function is 0 otherwise. Spikes from different source neurons j represent spatial summation, spikes from different spike times k represent temporal summation. $s(t)$ is a function that models a decaying behavior of the current, e.g. an exponential decay. Finally, J_{ij} is the *weight* of the synapse, which defines how strong the connection is established. The weight combines many different biological factors, like the amount of neurotransmitters in the axon terminal, the number of available channels in the dendritic spine and many more. The weight can either be positive or negative, representing excitatory or inhibitory synapses respectively.

Note that the summation bases on a so-called *current-based synapse*. It is also common to model synapses *conductance-based*, a comparison between these two types is given by Cavallari, Panzeri, and Mazzoni (2014). Within this thesis, only current-based synapses are applied.

Network architectures

Neurons can be connected to networks in different ways. We can broadly separate between a *feed-forward* and a *recurrent* architecture. In a feed-forward structure, neural activity is propagated in one direction only. Usually neurons are grouped in layers which are connected sequentially. A *recurrent neural network* (RNN) has a layer of neurons that is connected back to the same layer. Therefore, time-coded information can be included in recurrent structures. Another possible classification refers to the number of realized connections. A *fully connected* network has all possible connections established between neurons or between the neurons within the layers. In a *sparsely connected* network only a small fraction of the potential connections are realized.

A particular type of networks with a recurrent structure are reservoir networks, also referred to as *reservoir computing* (Lukoševičius & Jaeger, 2009). Input signals are fed into a large and randomly, but sparsely connected group of neurons, called reservoir. The reservoir is basically a complex nonlinear dynamic filter that transforms the incoming signals using a high-dimensional temporal mapping (Schrauwen, Verstraeten, & Van Campenhout, 2007). Schrauwen et al. (2007) described similarities between reservoir computing and kernel methods. Kernel methods are often used in machine learning to transform the input into a multi-dimensional or even infinite-dimensional feature space, where linear separation or linear fitting performs better than in the original input space (Hofmann, Schölkopf, & Smola, 2008). Under this perspective, also a reservoir performs a transformation of the input into higher dimensions. The readout, the state of the network's neurons, acts as a kind of feature space. If the reservoir contains N neurons, the input is projected into an N-dimensional space. A state of the reservoir is mathematically just a point in this high-dimensional space. Temporal signals entering the reservoir can be described as a trajectory in that space. Lazar (2009) hypothesized in her dissertation that, theoretically, the brain computes with trajectories in a similar way, making reservoir computing a promising approach for research in computational neuroscience.

Two major basic types of reservoir networks are Echo State Networks (ESN) (Jaeger & Haas,

2004) and Liquid State Machines (LSM) (Maass, Natschläger, & Markram, 2002). Both types were independently developed (Goodfellow, Bengio, & Courville, 2017), but are quite similar. LSM uses neurons with binary outputs. They are designed to explain processes in the brain, whereas ESN uses continuous hidden units and focuses on machine learning tasks. Under the condition of the *separation property* (different inputs result in separable outputs) and the *fading memory property* (information about recent input can be maintained) the LSM and the ESN fulfill the Stone-Weierstrass theorem, namely that both can approximate any continuous function (Grigoryeva & Ortega, 2018; Maass & Markram, 2004). Therefore, especially the LSM with its spiking neurons is an interesting starting point for reservoir based neural simulation. The network architecture used in this thesis is mainly based on this approach.

2.2 Movement

> "Hierarchical encoding of motor skills endows the system with the ability to efficiently generate new combinations of motor primitives without the necessity of forming execution-related representations de novo. The behavioral advantage of this architecture may explain why evolution has not simply expanded the primary motor cortex with direct access to the spinal cord, but instead has resulted in the emergence of several premotor areas with predominantly indirect cortico-spinal projections via M1." (Diedrichsen & Kornysheva, 2015)

In this section, we will get an overview over aspects of movement. First, the complex field of motor learning and representation is dissected into the two most researched components, which makes the topic easier to grasp: adaptation and sequence learning. Second, functions of the involved brain regions are introduced. We will see that up to today many details of motor learning and representation as well as the composition of the whole movement system still raises significant issues. Especially separating between motor processes and many other processes in the brain, like memory and cognition, is often not as clear as one would expect intuitively.

From selection to execution

We can distinguish three stages of movement, starting with the goal selection for the movement (Wong, Haith, & Krakauer, 2014), depending on the stimulus. Afterwards, a particular motor action needs to be chosen and finally executed (Krakauer et al., 2019).

- *Goal selection*: Depending on the environment, it is chosen what wants to be achieved by the movement.
- *Action selection*: A specific movement trajectory is selected, depending on the goal.
- *Action execution*: The movement trajectory is performed with a certain precision.

The general separation between selection and execution is also introduced in a review from Diedrichsen and Kornysheva (2015). Within this work, I will concentrate mainly on the *action execution* and especially how actions can be learned and represented in neural structures. *Action selection* also plays a role for concatenating single actions to longer movement sequences. *Goal selection*, however, is not in the research focus of this thesis.

2.2.1 Motor adaptation & movement sequences

In the literature, four different research areas can be distinguished (Krakauer et al., 2019). The two traditional paradigm of *motor adaptation* and *sequence learning* on one hand. And the two

Implicit adaptation	Explicit adaptation
Learns slowly	Learns quickly
Well retained	Poorly retained
Expressible at low reaction times	Expressible at high reaction times
Temporally stable	Temporally labile
Driven by sensory prediction error	Driven by reward and task error

Table 2.1: Implicit vs. explicit process underlying a motor adaptation (according to Huberdeau, Krakauer, & Haith, 2015; Krakauer, Hadjiosif, Xu, Wong, & Haith, 2019).

emerging fields of *de novo learning* and *motor acuity* on the other hand. In the following, I will focus on adaptation and sequence learning, since for these topics many studies are available and they allow to give an overview over the learning and representation of movement. Insights from these two research fields will be used for developing a theoretical model of how movement actions can arbitrarily be connected to complex movement sequences (see Section 3.2). This model will then form the theoretical framework for the main results. However, I will first introduce existing results and hypotheses.

Motor adaptation

Motor adaptation is the process of learning new movements based on existing similar movements. Therefore, adaptation is mainly important for the stage of action selection, but also includes action execution. At least two processes with different timescales underlie motor adaptation, a fast mechanism with a poor retention and a slow process with a good retention (Smith, Ghazizadeh, & Shadmehr, 2006). We can also distinguish between explicit learning, driven by a target or task error, and implicit learning by a sensory prediction error (Taylor, Krakauer, & Ivry, 2014). The task error defines how much the movement result deviates from the movement goal and the prediction error defines how much the performed movement trajectory varies from the intended trajectory (Tseng, Diedrichsen, Krakauer, Shadmehr, & Bastian, 2007). It turned out that the implicit learning corresponds to the slow learning process, whereas the explicit learning corresponds to the fast learning process (Huberdeau, Krakauer, & Haith, 2015; McDougle, Bond, & Taylor, 2015). Note that also a reward prediction error, giving only a binary result, can drive the explicit adaptation process (Shmuelof et al., 2012). Huberdeau et al. (2015) and Krakauer et al. (2019) summed up the two processes as shown in Table 2.1.

It is important to note that some noise and therefore some error will always be present. Variation in movement execution is today seen as an important factor for an exploration behavior. In general, there is a trade-off between exploitation and exploration. This said, an organisms needs to be able to perform a precise movement, but requires to keep enough variability to be able

to adapt to new situations as well (Dhawale, Smith, & Ölveczky, 2017; Wu, Miyamoto, Castro, Ölveczky, & Smith, 2014).

Sequences, chunks & hierarchies

Complex movements are only possible if single movement actions can be concatenated to longer sequences. Walking is one example for such a complex movement. Specifically the ability two walk bipedal is one of the outstanding features of the humans, which is probably due to energy efficiency (Sockol, Raichlen, & Pontzer, 2007). It is highly integral for the human being, since it seems that this skill was developed in the beginning of the early hominid time (Zollikofer et al., 2005). Every step is an interplay between many muscles, but one step alone is not sufficient. Many footsteps can be chained and coordinated, just imagine a dancer performing a specific pattern of moves. Hence it is obvious to ask: how are single actions concatenated to fluent, but flexible complex movements?

Two different modelling approaches have been developed. The chaining model and the hierarchical model (Krakauer et al., 2019), whereby evidence argues more for the latter in recent literature. The chaining model assumes independent actions and planning includes first to chain actions and then to execute them. The hierarchical model (Botvinick, 2008; Koechlin & Jubault, 2006; Rosenbaum, Kenny, & Derr, 1983) considers that movements are learned and represented as chunks and planning includes only choosing the correct chunk. Chunks can of course also be rearranged, however, this requires once again learning. A hierarchical structure assures that different chunk combinations of the same underlying movement actions can be stored. In the following, much evidence is collected which unfolds the properties of the hierarchical approach. Note that I use the term *movement action* for an atomic movement part which is part of a sequence or chunk, other may call it motor primitive (Diedrichsen & Kornysheva, 2015) or muscle synergy (Singh, Iqbal, White, & Hutchinson, 2018).

If a sequence is stored as a whole chunk after its representation was learned and a manual planning is not necessary before every single execution, we would expect to see a rise in reaction time only for the first action, which is in fact the case (F. M. Henry & Rogers, 1960; Sternberg, Monsell, Knoll, & Wright, 1978). In addition, the response time of the subsequent actions stays constant (Sakai, Kitaguchi, & Hikosaka, 2003). This means that a chunk can be seen as an action for itself, just on a higher level. It is further assumed that information of a chunk are buffered to allow a high execution performance (Verwey, 1996). But the buffer and therefore also the chunk length is limited. Studies assume that chunks can contain between 3 and 7 elements in general or movement actions in particular (Broadbent, 1975; Miller, 1956; Nissen & Bullemer, 1987; Sakai et al., 2003). Interestingly, if information is chunked, it is possible to use memory beyond the capacity of actual working memory (Miller, 1956). An important question at that point is, if the chunk represents only the whole movement sequence or if it also represent the underlying transitions. Rand, Hikosaka, Miyachi, Lu, and Miyashita (1998) argues that transitions are probably available and separately stored in the supplementary motor area (SMA). This is supported more recently by Stainer, Carpenter, Brotchie, and Anderson (2016) who found that

sequences learned by finger movements transfer to saccadic eye movements, indicating that transitions between actions are differently stored than the movement actions. In another study it was shown that a transfer between hands is possible. If a sequence is learned with one hand, the other hand can perform this sequence better than compared to learning a random sequence (Grafton, Hazeltine, & Ivry, 2002) which again supports the theory of separate representation of movement actions itself and their transitions to sequences. Furthermore, primary motor cortex (M1) only encodes for movement itself, but receives input from areas providing sequence representations (Yokoi, Arbuckle, & Diedrichsen, 2018). Note that Diedrichsen and Kornysheva (2015) published a concise review about the hierarchical model, which includes some further evidence about chunking.

2.2.2 Neural representation of movement

Krakauer et al. (2019) provides a detailed overview over all brain regions participating in motor learning. Here I just give a list of the most important regions, which will be introduced more detailed below. These include the cerebellum, the cortex – most importantly the motor areas –, the basal ganglia and the hippocampus.

- Basal ganglia (BG)
- Hippocampus (HC)
- Cerebellum (CB)
- Posterior parietal cortex (PPC)
- Pre supplementary motor area (preSMA)
- Premotor cortex (PM)
- Primary motor cortex (M1)
- Supplementary motor area (SMA)

These essential brain regions for movement are depicted in Figure 2.5.

Cerebellum

The cerebellum (CB) estimates the current state during a movement (R. C. Miall, Christensen, Cain, & Stanley, 2007; Xu-Wilson, Chen-Harris, Zee, & Shadmehr, 2009). Furthermore, it updates an internal model of the motor system, used to predict consequences of efferent motor commands (Daniel M Wolpert, Miall, & Kawato, 1998). This internal model is based on sensory error information, carried by strong discharges from climbing fiber and encoded by synapses of the parallel fiber connections (Albus, 1971; Ito, 2000; Marr & Thach, 1991). The CB then

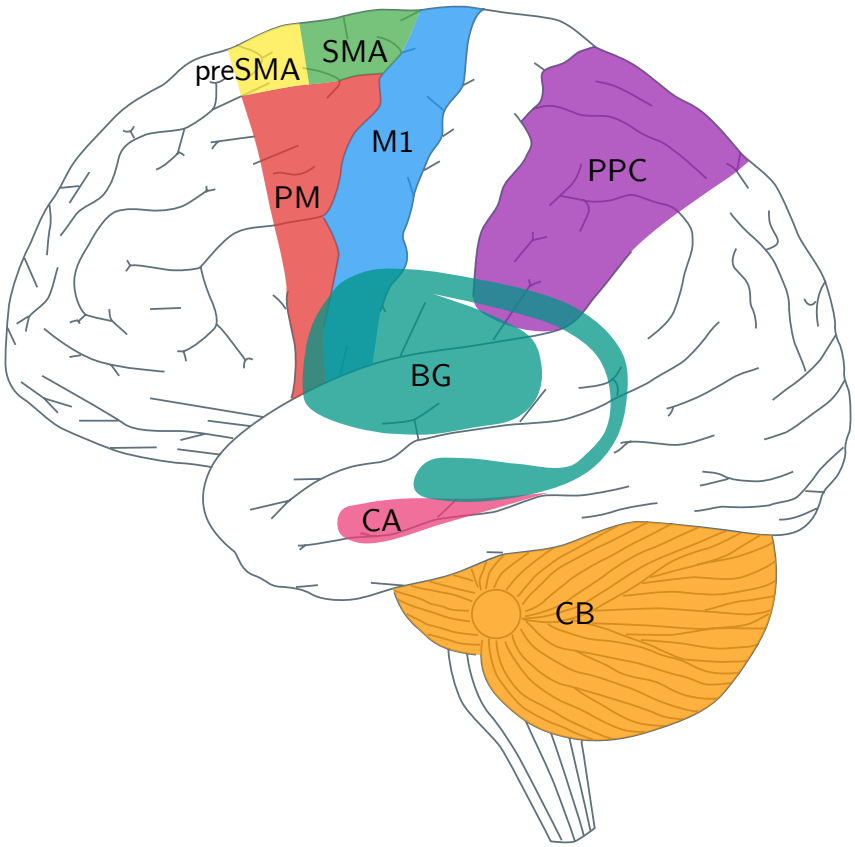


Figure 2.5: Shown are the brain areas related to movement which are introduced in the background chapter. Pre supplementary motor area (preSMA): yellow, Supplementary motor area (SMA): green, Premotor cortex (PM): red, Primary motor cortex (M1): blue, Posterior parietal cortex (PPC): purple, Basal ganglia (BG): teal, Hippocampus (HC): pink & Cerebellum (CB): orange. Note that more brain areas are related to movement (Krakauer, Hadjiosif, Xu, Wong, & Haith, 2019).
Brain template taken from: https://pixabay.com/images/id-150952 (2021, August 25)

performs the prediction of the movement by Purkinje cells that encode the movement trajectory or the movement outcome in general (Bastian, 2006; Ebner & Pasalar, 2008; Herzfeld, Kojima, Soetedjo, & Shadmehr, 2015). Nuclei within the CB transform the performed prediction into an adjusting motor command (Herzfeld et al., 2015, 2018; Medina, 2011). In summary, the CB predicts sensory consequences of movement. With this, it allow adjusting ongoing movements and coordinating movements between different body parts.

Basal ganglia

The basal ganglia (BG) are brain areas that receive input from many different brain regions like the cortex and the thalamus and they get especially dopaminergic modulation from the midbrain (Arber & Costa, 2018). They then convey incoming motor signals from these input regions to the brainstem (Thura & Cisek, 2017). The BG can be seen as a gate that controls if and how intensely motor commands are transmitted (Yttri & Dudman, 2016). Patients with Parkinson's disease or Huntington's disease, both diseases impairing the BG, have problems to execute sequences, but can still verbalize them (Doyon et al., 1997; Vakil, Kahan, Huberman, &

Osimani, 2000). These results indicate that explicit sequence learning seems to stay intact, only the execution is impaired. Evidence also comes from a study where the sensorimotor region in two monkeys was inactivated. Desmurget and Turner (2010) could demonstrate that the BG influences execution, but does not store either movement actions nor sequences. When the BG in monkeys is deactivated a gating behavior was not performed anymore (Bhutani et al., 2013).

In addition, the BG implement a competitive queuing which holds a second movement plan back, while the first movement plan is executed. Therefore, movement sequences are controlled or mediated, but not stored within the BG. Jin, Tecuapetla, and Costa (2014) could show that the BG is involved in chunking of movements, especially in parsing and concatenating sequences. This indicates an important role for the hierarchical model; the BG is part of the chunk selection and initiation (Diedrichsen & Kornysheva, 2015).

The modulation of movement executions seems to be mediated by dopamine (Arber & Costa, 2018; da Silva, Tecuapetla, Paixão, & Costa, 2018), which is reasonable, since BG is an important region for reward. The striatum, a part of the BG, provides a signal for an implicit "motor motivation" and therefore mediates the execution based on an intrinsic motivational state (Mazzoni, Hristova, & Krakauer, 2007). Explicit knowledge of the sequence order increases motivation and leads to sequence-specific improvement (Wong, Lindquist, Haith, & Krakauer, 2015). These results suggest that the BG mediate movement execution by motivational effects, which probably goes back to effects of the neurotransmitter dopamine in the BG.

Taken together, BG gate information from higher regions. This process is potentially related to learning from reward, carried by dopamine (Krakauer et al., 2019). The BG do not only gate single movement actions, but are significantly involved in parsing and concatenating chunks and movement sequence (Diedrichsen & Kornysheva, 2015). From our current understanding, we can imagine the BG as a buffer where incoming motor commands are queued. The BG prioritize these incoming instructions depending on the importance and intrinsic motivation and makes sure that the outgoing motor commands are executed orderly one after another.

Hippocampus

The hippocampus (HC) also has a role in motor learning (Schendan, Searl, Melrose, & Stern, 2003). An interplay between the striatum, a part of the BG, and the HC seems to improve the performance of motor sequences if subjects have slept after training (Albouy et al., 2008; Albouy et al., 2013). This indicates a clear role of the HC in movement sequence consolidation. Interestingly, the HC seems to be especially important for learning higher-order sequences (Curran, 1997; Schendan et al., 2003) and is, beyond explicit learning, also active while implicit learning (Fletcher et al., 2004; Gheysen, Opstal, Roggeman, Waelvelde, & Fias, 2010). In addition, it seems that the HC supports learning of sequences probably by associating nearby fragments of a sequence (Krakauer et al., 2019).

Cortex

This section covers several regions in the cortex. I will start with the motor cortex in general and will become more specific, especially introducing the role of the M1 and the SMA.

The motor cortex has two types of neurons with different specialization: one type has early preparatory activity, the other type is active in late preparatory activity and during motor commands (Economo et al., 2018). The first part connects to thalamus and back to motor cortex, the second part connects to motor centers within the medulla. The motor cortex is probably not too much involved in the actual execution of a movement. Lesions of the motor cortex did not impair movement execution in rats (Kawai et al., 2015). The authors hypothesised that subcortical areas play an important role in execution and that the motor cortex has perhaps more a tutoring role. This fits together with results from Yeom, Kim, and Chung (2020), showing that activity in motor areas increases during planning, but decrease during movement. Only the CB and the BG had increased activity during movement.

The primary motor cortex (M1) seems to be important for retention, but not for the acquisition of movements. If the excitability of the primary motor cortex (M1) is increased, it leads to an increase in retention, but an improved movement adaptation was only observed if the cerebellum was stimulated (Galea, Vazquez, Pasricha, de Xivry, & Celnik, 2010). Further, M1 stores only single movement actions and not whole sequences (Aumann & Prut, 2015). The representation of single spatio-temporal movement trajectories probably takes place in sub-networks of the M1 which can then be executed as a coordinated muscle activity as part of a sequence (Diedrichsen & Kornysheva, 2015; S. Overduin, d'Avella, Carmena, & Bizzi, 2012). While the M1 encodes short movement actions, it does not store information about sequence representation, which is provided by other areas (Yokoi et al., 2018).

In contrast to the premotor cortex (PM), the M1 has fewer connections to cortical areas. Before sensory information reaches the PM, it has to be processed by higher order corticies. The PM has therefore a coordinating function for the M1 (Chouinard & Paus, 2006).

The posterior parietal cortex (PPC) is important for movement adaptation, since an impairment of the PPC leads to a reduction of the level of adaptation at a late stage of the learning phase (Della-Maggiore, 2004). An activation of the PPC during the late learning phase or post consolidation recall phase was also shown by Shadmehr (1997) and Krakauer et al. (2004). In addition, the PPC is more important for the retrieval than for the acquisition of visuomotor sequences (Sakai et al., 1998).

Finally, also the supplementary motor area (SMA) has a role in motor learning. We can separate between the SMA and the preSMA, which is the anterior region of the SMA. Both seem to have different functions for sequence learning. The preSMA is more active during learning of new movement sequences (Hikosaka et al., 1996; Nakamura, Sakai, & Hikosaka, 1998), in particular in context of chunking (Sakai et al., 2003). If the preSMA is inactivated, the execution of existing sequences works still well, whereas new sequences can hardly be learned (Nakamura et al., 1998). The SMA, on the other hand, is in case of an inactivation still equally active

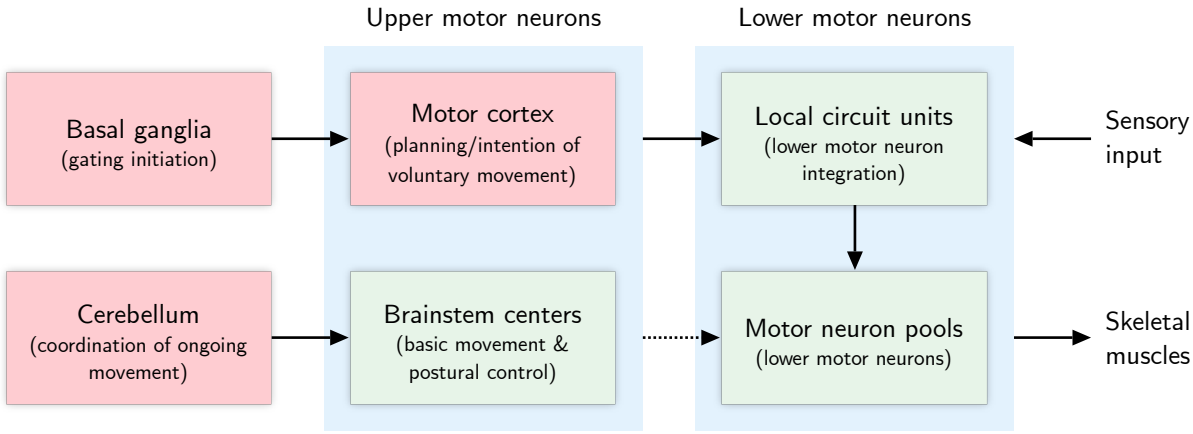


Figure 2.6: Interactions between movement related brain regions (see also Purves et al., 2013). Red boxes indicate higher-order regions, which are introduced in the text. Green boxes indicate lower-order regions, not mentioned in the text.

during learning and during performing sequences (Hikosaka et al., 1996; Nakamura et al., 1998). While the SMA seems to prepare the next movement, the preSMA prepares the second-next movement (Nakajima, Hosaka, Mushiake, & Tanji, 2009). In summary, the whole SMA region represents the order of the sequences and guides its learning as well as its execution (Krakauer et al., 2019).

Summary of movement related brain regions

In this last paragraph, I want to summarize the roles of the main important brain regions in order to leave the reader with a clear picture of the relations. Note that this is just an overview with the intention to outline the interplay between the regions. The M1 encodes single movement actions, whereas the PM helps to coordinate them, depending on inputs from higher-order regions. The SMA concatenates single movement actions to whole chunks or sequences. It helps to learn and finally represents these sequences. Actions or action sequences are then queued within and gated by the BG. The gating depends on an intrinsic motivation or "urgancy" for a particular movement and is mediated by dopamine. While the motor commands are then sent to the brainstem and are executed by the muscles, the CB coordinates the ongoing movement. Figure 2.6 outlines the interactions between the regions. Diedrichsen and Kornysheva (2015) concludes that these complex relations exist for a reason and may reveal some of their meaning when looking from the perspective of the hierarchical model. Within this thesis, I will use insights about movement related brain areas to develop the hierarchical motor selection and execution (HMSE) model within Section 3.2. The execution layer of this model is then used as a theoretical foundation for developing concrete solutions for SNN simulations in Section 3.3.

2.3 Neuromorphic hardware

> "We envision these systems scaling up to hundreds of racks, with billions of neurons and trillions of synapses, that run complex, integrated, large-scale neural applications with unprecedented energy-efficiency and throughput." (Sawada et al., 2016)

The developments in neuroscience, described in Section 2.1, and other scientific findings in computational neuroscience and material science lead to the idea of molding the state of the knowledge into hardware. At that point, the story of neuroscience continuous with the development of neuromorphic hardware, the "next phase of brain research" (Fan & Markram, 2019), implementing – in most cases – spiking neural networks. This third generation of neural networks (Maass, 1997) has the advantage that only sparse binary information, i.e. spikes, are transferred between the computational units. Applying this concept to specialized hardware systems, beyond the *von Neumann* architecture, comprises the opportunity to simulate neural networks in much larger scales and with lower energy consumption (Zhu, Zhang, Yang, & Huang, 2020).

In the 1980s many research developments facilitated the idea of neuromorphic hardware, first mentioned by Mead and Ismail (1989) and Mead (1980). Important research fields include materials science, production methods and advances in neuroscience (Zhu et al., 2020). From this point on it took until the 2010s when larger systems were developed. These analog and/or digital system include SpiNNaker (S. B. Furber, Galluppi, Temple, & Plana, 2014), BrainScaleS (Schemmel et al., 2010) and Loihi (Davies et al., 2018). Recent neuromorphic chips are listed in Table 2.2.

In this background section I will first introduce the functionality and types of today's neuromorphic hardware. In the second section, currently existing neuromorphic systems are briefly compared and foreseeable next development stages are introduced. Finally, the specific neuromorphic chip Loihi, used within this thesis, and its abilities as well as its limitations are introduced in detail.

2.3.1 What is neuromorphic hardware?

In 1945, John von Neumann introduced a possible computer architecture for realizing a Turing machine (Hodges, 2014; Turing, 1937), which is today known as the *von Neumann* architecture (von Neumann, 1993). This architecture contains a memory, that stores data *and* instructions, and a separated central processing unit (CPU). Both communicate via a bus. The system allows communication with mass storage and provides input & output mechanisms. A simplified sketch is given in Figure 2.7A. However, the von Neumann architecture comes with a drawback, known as the von Neumann bottleneck (Backus, 1978; Efnusheva, Cholakoska, & Tentov, 2017). This bottleneck persists since all data between the CPU and the memory has to be transmitted via

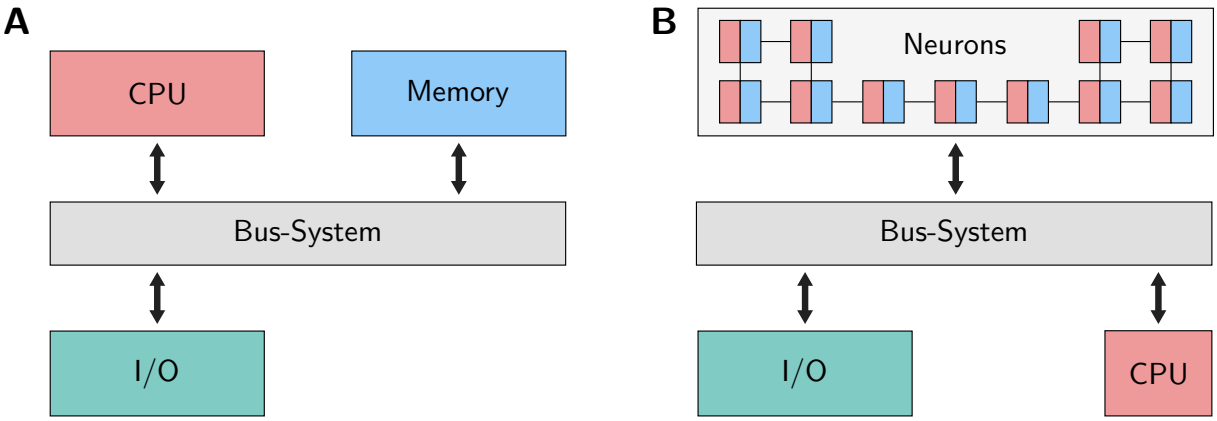


Figure 2.7: Comparing von Neumann with neuromorphic architecture. The main principles are sketched. **(A)** The von Neumamm architecture with a powerful CPU and a large memory, both connected by a bus system. The bus system allows to connect to input/output devices and storage as well. **(B)** Neuromorphic hardware consists of small entities, called neurons, that contain computing units with a very limited instruction set and small memory. The bus systems provides a link to input/output devices and often an additional CPU with a conventional von Neumann architecture.

the bus. Today's CPUs implement caching systems to reduce the bottleneck, but those solutions have other drawbacks (Efnusheva et al., 2017).

One possible solution to the problem is to colocate the processing units with the memory entities (Schuman, Birdwell, Dean, Plank, & Rose, 2016; Young, Dean, Plank, & Rose, 2019). Both, the computational unit and the memory component are designed simple in this architecture. This simplicity, however, enables not only a highly parallel layout with many computation-memory units, but also allows a high scalability of the system. A sketch of the principle of neuromorphic hardware architecture is shown in Figure 2.7B. Apparently, this architecture is similar to how the brain computes according to the neuron doctrine (see Section 2.1.1), small independent neurons connected to each other via synapses, communicating in a decentralized manner. An important perspective on neuromorphic hardware is that memory is represented locally in synaptic weights in neuromorphic hardware (Young et al., 2019), and therefore requires a new perspective on memory. Kuzum, Yu, and Wong (2013) defined 5 properties of the brain, compared to how standard computers are working. According to them, the brain is

1. massively parallel, organized in a three-dimensional structure and extremely compact,
2. power efficient,
3. combines storage and computation,
4. fault and variation tolerant and robust,
5. self-learning and adaptive to changing environments.

Neuromorphic hardware is massively parallel inherently by design. The same holds for the combination of storage and computation. It also provides energy efficiency due to its high

specialization, the spike-oriented, and therefore sparse, information transmission, the high parallelism and some more factors (D. Liu, Yu, & Chai, 2020). One of the promises of neuromorphic hardware is to come closer to the energy consumption of the brain, which is as low as 20 Watt[3]. Therefore the first 3 conditions are met by neuromorphic systems, just because of how they are build. The last two properties depend mostly on the algorithm and less on the system itself. Obviously, a particular system needs to support plasticity to be able to meet the self-learning and adaptivity property. But the sole provision of plasticity does not guarantee that learning is possible in a meaningful way. This evokes a challenge in the field of neuromorphic computing, which is probably the biggest challenge in this field currently: How to build local algorithms that have a global meaning? The property *local* means that computations are performed decentralized at the synapses. More specifically, a synapse can only access information from itself and the pre- and post-synaptic neuron, but cannot access or change other synapses' states, e.g. a normalization operation for all synaptic weights is not possible in the brain and highly inefficient in neuromorphic hardware. Nguyen, Tran, and Iacopi (2021), Schuman et al. (2017), Steve Furber (2016) and also Zhu et al. (2020) point out that finding such local algorithms is a challenge. Note that this problem does also exist for spiking neural networks in general, but it is more serious in the field of neuromorphic hardware. In simulations performed on computers with a standard architecture often non-local mechanisms, that require a centralized access, are implemented to solve a specific problem not solvable yet. Neuromorphic hardware forces the scientist to design the algorithm consequently local, at least considerably more than on standard computers. To put it in words from Schuman et al. (2017):

> "However, in order for this innovation to take place, algorithm developers will need to be willing to look beyond traditional algorithms such as back-propagation and to think outside the von Neumann box."

Also Zhu et al. (2020) state that:

> "Although currently there is a flourish in network level research concerning the application of crossbar arrays, huge knowledge from neuroscience studies is still urgently required [...]."

We can conclude that neuromorphic hardware is a promising approach to overcome the von Neumann bottleneck, which promises large network simulations, high scalability and power efficiency. On the other hand, we have to invent new algorithms and even need to develop a new decentralized "thinking".

[3]The 20 Watt claim is often repeated, but mostly unsatisfactorily cited in the literature. It can roughly be estimated by using the *Schofield equation* (Schofield, 1985) to, first, calculate the base metabolic rate of the human body (but see also C. Henry, 2005). The base metabolic rate depends heavily on age and weight of the body, but a mean would end up at roughly around 100 Watt. The energy consumption of the brain can then be estimated taking the oxygen consumption of the brain into account. Since the brain consumes about 20 % of the total oxygen (Jain, Langham, & Wehrli, 2010; Raichle & Gusnard, 2002), it is assumed that also the energy consumption is about 20 % of the body's energy consumption, which, finally, leads to the often referenced 20 Watt.

Properties of neuromorphic hardware

In this paragraph I will give an overview over properties of neuromorphic hardware and how systems can be compared. Of course, every chip is individual, but some general properties can be identified. I focus on those properties which are most relevant for science and application and will only consider few aspects of the chips' architecture or materials. Note that the understanding of these properties is less important for readers who are solely interested in understanding the thesis, but inevitable for researches who want use results from this thesis for further research on neuromorphic hardware.

In principle neuromorphic hardware can implement different kind of neurons, but most neuromorphic hardware systems provide at least the LIF neuron (Rajendran, Sebastian, Schmuker, Srinivasa, & Eleftheriou, 2019). Some systems even allow flexible neuron models. Furthermore, some chips provide plasticity and some not. But even if plastic synapses are provided, the implementation of synapses differs between chips, since plasticity is a wide field and many different mechanisms can possibly be included. Zenke and Gerstner (2017) give an overview over a great variety of synaptic learning mechanisms. If existing algorithms are ported to or between neuromorphic hardware systems, it is important to be aware of the provided synapse and neuron models, as well as the supported plasticity mechanism.

Another aspect is the manufacturing technology which determines the size of a transistor channel, today often in a scale of nanometers. The size of a transistor matters, since smaller silicon units transmit information faster and have a lower power consumption due to less resistance (Khaled Ahmed & Schuegraf, 2011). In science it is mostly not necessary to optimized power consumption of neuromorphic hardware to its limits. However, the manufacturing technology might be important when algorithms are designed for low-power applications, e.g. when the here provided robotic control algorithm in Section 3.3.1 shall be applied to an autonomous system.

One important distinction between neuromorphic system is the underlying transmission technology. Chips can either compute with analog signals or digital signals. Some systems are build as a hybrid in order to obtain advantages from both worlds (Rajendran et al., 2019; Thakur et al., 2018). Analog hardware can either store analog values directly as a voltage in a capacitor or it can represent an analog value as a digital multibit value. In addition, it can be separated between subthreshold and above-threshold (also denoted as superthreshold) circuits. Subthresold circuits (Vittoz, 1996) are very noise sensitive, but need lower noise energy and have a better energy-efficiency (S.-C. Liu, Delbruck, Indiveri, Whatley, & Douglas, 2014). In general, full analog chips suffer from leakage of the synaptic weights, which limits long-term learning (Saxena, 2021). The other type, namely digital hardware, is based on logical circuits, like adders or multipliers (Seo et al., 2011). This type of chip is deterministic and has good scaling abilities (Thakur et al., 2018). Finally, mixed analog/digital architectures try to combine the advantages from analog and digital chips. Some hardware designs apply for example analog circuits for the computation of the neural spike and digital circuits for routing spike transmissions (Thakur et al., 2018). In this thesis, a digital chip was used that provides a deterministic behavior, which was reasonable for this work, since the development of algorithms on non-deterministic systems

is more difficult.

Moreover, some chips provide time-multiplexing. In this case, a physical silicon neuron is actually used for multiple neurons which are evaluated one by another. This, of course, slows down the system, but increases the amount of available neurons drastically. The user has to deal with a trade-off between speed and size of the network. Chips implementing time-multiplexing are often still able to simulate in real-time (Davies et al., 2018). While in science, speed is important, in technical applications often a real-time behavior is sufficient. In our implementation, we tried to exploit as many cores as possible to reduce the number of neurons per core in order to increase the speed. This is performed by the `PeleNet` framework, introduced in Section 3.1.1.

A final property is the communication type, namely if the communication is synchronous or asynchronous. It is self-evident that the brain operates asynchronous, but chips can also be build in a synchronous way. However, Martin and Nystrom (2006) argues that system on a chip (SoC) single-clock synchronisation will not be feasible in the future. And in fact, many today's chips process information purely asynchronous (Rajendran et al., 2019). But a synchronous communication does not mean, that the computational cycles need to be fixed. It can be separated between fixed length cycle times and variable length cycle times (Young et al., 2019), where the variable length cycle times is also often referred to as asynchronous (e.g. in Davies et al., 2018). Note that this event-driven synchronisation has the drawback, that the algorithmic time of the chip is unrelated to the real-time (Young et al., 2019). For the implementation of our algorithms it is less important which property is supported by the chip, either synchronous or asynchronous. But it is perhaps important for specific applications, like robots in a factory that may require an equal performance time and therefore depend on synchronous systems. In most cases this reliable clocking is probably not required and the speed is most important, where asynchronous systems are the best choice.

As a summary, neuromorphic chips can be build in many different ways with a variety of advantages and disadvantages depending on the architectural choice. Here I have introduced differences in the neuron model, the plasticity, the manufacturing technology, the basic architecture and the communication method. In the next section, I will introduce specific state-of-the-art chips.

2.3.2 Comparison and future of neuromorphic hardware systems

Many different neuromorphic systems have been developed since the 2010s. I will give a brief overview over the currently available chips ODIN, DYNAP, Darwin, TrueNorth, Baindrop, SpiNNaker and BrainScaleS. Finally, a detailed introduction to Loihi follows in the next section.

2009: SpiNNaker

The SpiNNaker project was started in 2009 at the University of Manchester (Stephen Furber & Brown, 2009; S. B. Furber et al., 2014) and is part of the Human Brain Project (Markram et al., 2011). The goal is to facilitate large network simulations with more than a billion neurons and a realistic connectivity of about $1,000$ to $10,000$ synapses per neuron. The neuron model is flexible and provides many usual neuron models, like LIF and Izhikevich (E. Izhikevich, 2003, 2004). These large networks shall still provide plasticity and run in real-time. For this purpose ARM9 cores are combined with some small memory and finally tiled to large board. The chip is therefore digital, fabricated with $130\,nm$ technology and globally asynchronous and locally synchronous (Martin & Nystrom, 2006). Note that a global memory is not implemented (Rajendran et al., 2019). Programming is, inter alias, possible using the PyNN interface (Davison, 2008). SpiNNaker is foremost a supercomputer, but also small implementations exist, e.g. for robotic applications (e.g. Araújo, Waniek, & Conradt, 2014; Denk et al., 2013).

In 2018, the second generation of SpiNNaker was announced (Höppner & Mayr, 2018). It is intended to increase the possible network size for simulations from $1\,\%$ of the human brain to the whole brain. In addition, further features will be implemented to extend the range of possible network model implementations (Young et al., 2019).

2010: BrainScaleS

The first BrainScaleS system was introduced by Schemmel et al. (2010). It has a mixed architecture, where neurons are processed with analog circuits, but the communication is performed digitally (Meier, 2015). BrainScaleS implements the adaptive-exponential integrate and fire (AdEx) neuron. 512 neurons can be used on one chip, with up to $14,000$ synapses per chip. But several chips can be joined, such that a wafer allows $180,000$ neurons and 40 million synapses. An outstanding feature of the chip is the ability to simulate up to $10,000$ times faster than real-time. The next generation, BrainScaleS-2, supports plasticity utilizing general-purpose processors on the chip (Friedmann et al., 2017). The new generation also provides more additional features, like nonlinear dendrites and structured neurons (Kaiser et al., 2021; Young et al., 2019). Finally, BrainScaleS can, like SpiNNaker, also be programmed using the PyNN interface (Davison, 2008). The development of the BrainScaleS chip is, again like SpiNNaker, part of the Human Brain Project.

2014: TrueNorth

The TrueNorth chip was developed by the company IBM. It is a digital chip and its production is based on $28\,nm$ technology (Akopyan et al., 2015). Similar to SpiNNaker, it is globally asynchronous and locally synchronous. Each core time-multiplexes 256 neurons and 4096 cores are tiled to a chip. Every core provides 256 outgoing and incoming axons. It incorporates LIF neurons, but does not provide plasticity. In recent years, the TrueNorth chip was extended to

	SpiNNaker	**BrainScaleS-1**	**TrueNorth**	**Braindrop**
# neurons core/chip	$1,000$	512	256	$4,096$
# cores/chips	$1,000,000$	384	$250,000$	?
Manufacturing process	$130\,nm$	$180\,nm$	$28\,nm$	$28\,nm$
Neuron model	Variable	AdEx	LIF	NEF
Type	digital	mixed	digital	mixed
Plasticity	yes	yes	no	?
Communication	GALS	async	GALS	?
Simulation speed	real-time	accelerated	real-time	real-time
	Darwin	**DYNAP**	**ODIN**	**Loihi**
# neurons core/chip	$2,048$	256	256	1024
# cores/chips	?	4	?	$2,097,152$
Manufacturing process	$180\,nm$	$28\,nm$	$28\,nm$	$14\,nm$
Neuron model	LIF	AdEx	LIF	LIF
Type	digital	mixed	digital	digital
Plasticity	?	yes	yes	yes
Communication	?	async	sync	async
Simulation speed	?	real-time	?	real-time

Table 2.2: Comparison of active neuromorphic hardware projects. Some unknown values are marked with a question mark. GALS stands for *globally asynchronous locally synchronous*. Synchronous and asynchronous communication type is abbreviated by sync and async respectively.

larger boards, denoted as *NS16e* (DeBole et al., 2019). For example, the NS16e-4 board provides 100 million neurons.

2014: NeuroGrid & Braindrop

NeuroGrid and Braindrop were both developed at the Stanford University, where Braindrop is historically the second design. NeuroGrid was introduced in 2014 and aimed to provide simulating large-scale networks in real-time (Benjamin et al., 2014). It is an analog/digital mixed architecture with subthreshold circuits, fabricated with $180\,nm$ technology. Every chip provides $65,536$ two compartment neurons. Chips can be extended to large boards which increases the number of neurons significantly. Neurons consist out of a soma, a dendrite, gating-variables and synapses. NeuroGrid does not provide plasticity.

The second design, Braindrop, was published 5 years later (Neckar et al., 2019). The chip follows a novel approach: it is tailored to use the Neural Engineering Framework (NEF) (Eliasmith & Anderson, 2003), which adds a quite high abstraction layer on top of the chip. It allows defining coupled nonlinear dynamical systems which are then automatically synthesized into the hardware. A Braindrop chip provides $4,096$ neurons and is produced in a $28\,nm$ process. Since the introducing study does not report about plasticity, we can assume that also Braindrop does not provide on-chip learning.

2015: Darwin

Darwin, a chip developed at the Zhejiang University and Hangzhou Dianzi University in China (Ma et al., 2017; Shen et al., 2015), is another smaller neuromorphic hardware project. The chip is digital, fabricated with $180\,nm$ technology and provides 8 physical LIF neurons which are time-multiplexed. This increases the number of neurons to 2048. Theoretically $4,194,304$ synapses are possible. It was not possible to find information about plasticity, which leads to the assumption that on-chip learning is probably not supported.

2017: DYNAP

A whole architectural family of chips is developed by the Institute of Neuroinformatics at the University of Zurich. The architecture is called dynamic neuromorphic asynchronous processors (DYNAP). It was originally produced with a $180\,nm$ technology, but was improved to $28\,nm$ (Thakur et al., 2018). The chip Dynap-SEL is mixed analog/digital and communicates asynchronously (Moradi, Qiao, Stefanini, & Indiveri, 2018). Each core has 256 analog adaptive-exponential integrate and fire (AdEx) neurons and 64 plasic synapses for each neuron. Since the chip provides 4 cores, the whole system comprises 1024 neurons and $64,000$ synapses.

2018: ODIN

The online-learning digital spiking neuromorphic processor (ODIN) (Frenkel, Lefebvre, Legat, & Bol, 2018) is a $28\,nm$ chip and processes information digitally. Implemented is a first-order LIF neuron, but also a second-order Izhikevich (E. Izhikevich, 2003, 2004) behavior can be modelled. 256 neurons and $65,536$ synapses with a precision of 3 bits are available. Synaptic plasticity is possible and the transmission of spikes is performed synchronously.

2018: Loihi

In 2018 the digital chip Loihi was introduced by Intel (Davies et al., 2018). Since this chip was used in this work, it is described in detail below in the next section.

Future

Many different architectures were developed within the last decade. These systems are already considerably energy efficient and provide fairly large networks. Another main advantage, compared to the von Neumann architecture, is the scalability. For most neuromorphic systems, chips can be assembled to larger boards providing as many neurons as necessary for a specific application. This can be very small, e.g. for Internet of Things (IoT) applications, or very huge, like for science-related neuromorphic supercomputers.

Nevertheless, it is still in an early stage. In Section 4.1 in the discussion, I consider limitations of Loihi which will probably also affect most other state-of-the-art systems. For the future of neuromorphic computing, Rajendran et al. (2019) predicts two stages of innovations. Within the first stage, conventional neuromorphic hardware systems, as we know them today, will be improved year by year and will overcome today's limitations. In a second stage, novel materials will play an important role for future neuromorphic systems. For instance, memristors are a promising alternative to the complementary metal-oxide-semiconductor (CMOS) technology currently used in hardware systems. They allow even better scalability, small on-chip area, low-power dissipation, efficiency and adaptability (Chua, Sbitnev, & Kim, 2012; Krestinskaya, Ibrayev, & James, 2018), as recapped by Krestinskaya, James, and Chua (2020). The development of bioinspired memristors is currently a dynamic and fast-emerging research field (Chakma et al., 2018; Choi, Yang, & Wang, 2020; Fu et al., 2020).

This overview is meant to support the reader assessing the description of Loihi in the next section. In Table 2.2 the main properties of the above-introduced systems are summarized.

2.3.3 The neuromorphic research chip Loihi

Most results within this thesis were performed on the neuromorphic research chip Loihi (Davies et al., 2018). It is a digital chip, fabricated with $14\,nm$ technology that communicates event-driven. It incorporates 1024 LIF neurons per core and provides plastic synapses. Every core contains only one physical neuron, the 1024 neurons are time-multiplexed. One chip contains 128 cores and several chips can be combined to boards. In summary, one chip allows up to $131,072$ and $130,000,000$ synapses. With this, Loihi is to date the only chip that provides a notable amount of neurons and synapses without requiring a supercomputer infrastructure. Since it implements a variant of the LIF neuron, it is best suited for simulations on a network level. Its digital architecture allows to keep track of the dynamics of all neurons and synapses that facilitates both, the development of neural networks for movement encoding and an algorithm for robotic control.

Intel provides a `Python` software packages called `NxSDK`, which contains two APIs. The `NxCore` API supports lower-level operations with the chip, is only little documented and not recommended for a practical usage in most cases. The `NxSDK` API is a higher-level software development kit (SDK) that directly allows defining neuron groups per core and connecting neuron groups within and between cores. It also offers defining probes, learning rules and spike generators. The latter are used to predefine external input for neurons to given points in time. In Section 3.1.1 I describe a self-developed additional framework for reservoir computing on Loihi that is based on the `NxSDK` and enabled most of the simulations provided within this thesis.

Neuron model

While projects like e.g. SpiNNaker and BrainScaleS provide a lot of details about the hardware and the embedded spiking models, details about the mechanics of Loihi are more shallow. The available papers, introducing Loihi (Davies et al., 2018; Lin et al., 2018), provide very limited information about the functionality of the chip. We have therefore published a paper describing details of the chip's neuron and synapse model and refer the reader to the preprint in Section 6.3 for details. Here I provide a summary of the neuron and synapse model.

Loihi uses a variant of the LIF neuron model. The voltage of a LIF neuron i on Loihi can be described by the following dynamic

$$\frac{dv_i}{dt} = -\frac{1}{\tau_v} v_i(t) + I_i(t) - v_i^{th}\sigma_i(t). \tag{2.12}$$

The first term on the right side describes the voltage decay, the second term is the current input to the neuron, and the third term resets the voltage to zero after a spike by subtracting the threshold. A spike is caused if $v_i > v_i^{th}$ and transmitted to connected neurons. τ_v is the time constant for the voltage decay and v^{th} is the threshold voltage to spike. $\sigma(t)$ is the so-called

spike train which indicates whether the neuron spikes at time t and is defined as

$$\sigma_i(t) = \sum_k \delta(t - t_{i,k}). \tag{2.13}$$

$t_{i,k}$ denotes the time of the k-th spike of neuron i.

In our paper, available in Section 6.3, we could show that Loihi uses the forward Euler numerical integration method to solve the differential equation of the voltage. The voltage with discrete values and unitless time is finally computed according to

$$v_i[t] = v_i[t-1] \cdot (2^{12} - \delta^u) \cdot 2^{-12} + I_i[t] + I_i^{\text{bias}}. \tag{2.14}$$

Note that the decay value δ^v is proportional to the previously used time constant τ_v according to $\delta^v = 2^{12}/\tau_v$.

Synapse model

Davies et al. (2018) describe the behavior of the synaptic input $I(t)$ by

$$I_i(t) = \sum_j J_{ij}(\alpha_I * \sigma_j)(t) + I_i^{\text{bias}}, \tag{2.15}$$

where J_{ij} is the weight from unit j to i and I_i^{bias} is a constant bias input. The spike train σ_j of unit j is convolved with the synaptic filter impulse response α_I, given by

$$\alpha_I(t) = \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{2.16}$$

where τ_I is the time constant of the synaptic response and $H(t)$ the unit step function.

Again, the forward Euler method is used which results in the following computation of the synaptic input

$$I_i[t] = I_i[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + 2^{6+\Theta} \cdot \sum_j \left(\tilde{w}_{ij} \cdot s_j[t]\right), \tag{2.17}$$

where $s[t]$ is zero unless there is an incoming spike on the synapse, in which case it is one. Note that also for the synaptic input $\delta^I = 2^{12}/\tau_I$. Θ is an exponent that scales the weight mantissa $\tilde{w}_{ij}$.

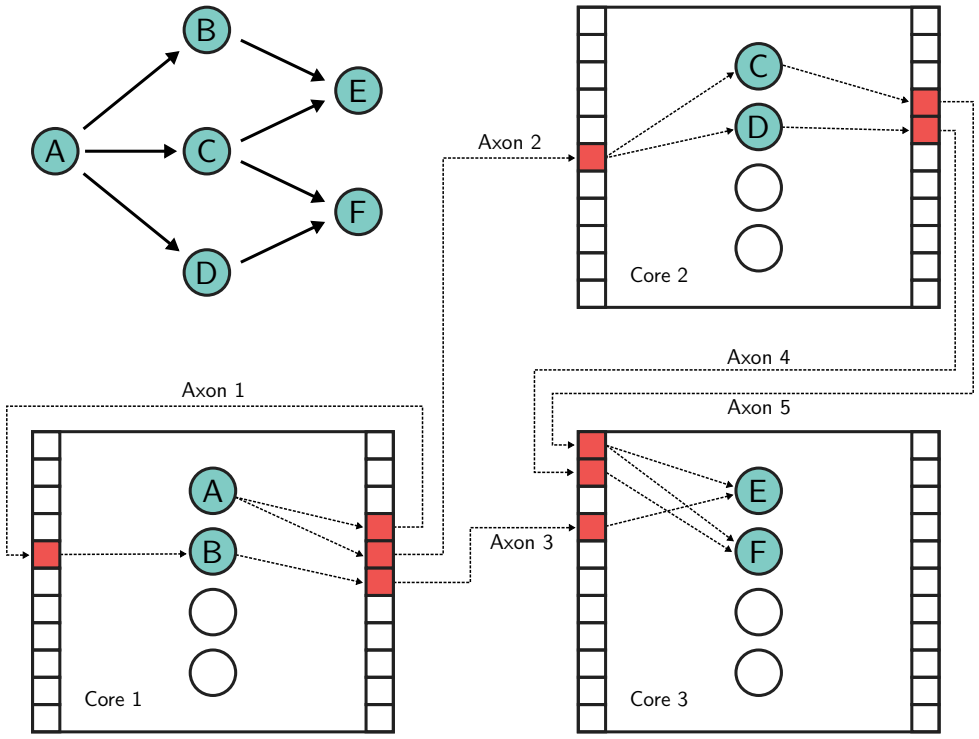


Figure 2.8: Shown is how neurons are wired over cores on the Loihi chip. The intended network structure is shown on top left. Right and bottom are 3 neuromorphic cores, comprising 6 neurons, 5 axons and 7 synapses. Used neurons within the cores are colored in teal, whereas used input and output routing slots are colored in red. Axon connections transmit information between cores. Every axon can carry information for several synapses. The figure is based on Figure 3 in Davies et al. (2018).

Plasticity

Synapses on Loihi provide on-chip learning. For this, several variables are available, comprising a pre-synaptic spike indicator, 2 pre-synaptic traces, a post-synaptic spike indicator, 3 post-synaptic traces, a reward spike and a reward trace, the weight itself and a tag. A full list is provided by Davies et al. (2018). The spike indicators are of value 0 or 1, depending on the spike state of the pre- or post-synaptic neuron or a given reward. A trace i decays exponentially according to

$$x_i[t] = \alpha \cdot x_i[t-1] + \hat{x}_i \cdot s[t]. \tag{2.18}$$

A spike, indicated by $s[t] \in \{0, 1\}$, increases the trace value by a constant parameter $\hat{x}_i$. It then decays according to the decay factor α. In practice, on Loihi one does not set α directly but instead a decay time constant τ_{x_i}. Again assuming a first order approximation for synaptic traces, akin to synaptic input and voltage, α can be defined as

$$\alpha(\tau_{x_i}) = 1 - \frac{1}{\tau_{x_i}}. \tag{2.19}$$

Note that this definition of α does not perfectly match the behavior on Loihi, the actual imple-

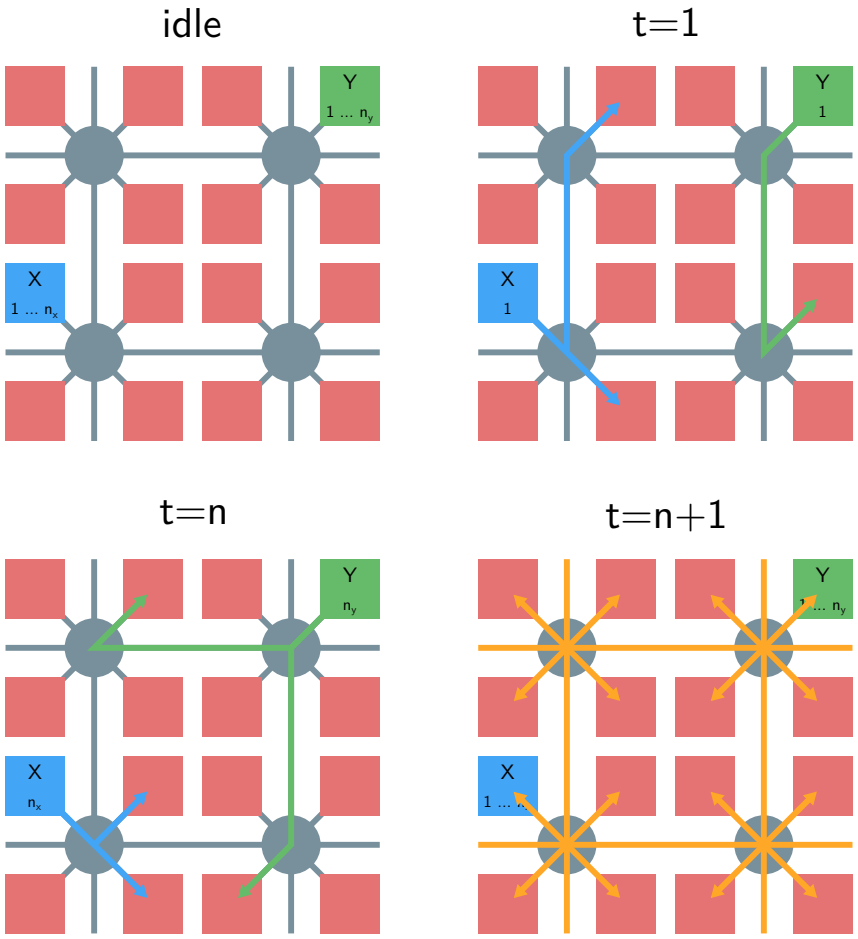


Figure 2.9: The principle of the Loihi mesh protocol. We assume two cores, X in blue and Y in green. For simplicity we choose $n = n_x = n_y$. **(idle)** The network is ready for transmitting spikes. **(t=1)** Spikes of the first neuron of both cores is distributed to their targets. **(t=n)** This continues step by step until all neurons in both cores have transmitted their spikes. **(t=n+1)** The protocol sends a barrier synchronization messages, informing the neighbors that all spikes are transferred. The figure is based on Figure 2 in Davies et al. (2018).

mentation is unknown. However, our definition of α is sufficiently close to simulation results, therefore we assumed that the real implementation should be at least very similar to this definition.

Finally, an example learning rule with an asymmetric learning window for the STDP rule is given to illusrate how the variables can be used. The learning window is similar to the one shown in Figure 2.3. The learning rule uses one pre-synaptic trace x_1 and one post-synaptic trace y_1. In addition, the dependency factors $x_0 \in 0, 1$ and $y_0 \in 0, 1$ are used, which indicate a pre- and post-synaptic spike respectively. Using these components, an STDP learning rule can be defined as

$$dw = 2^{-2} \cdot x_1 \cdot y_0 - 2^{-2} \cdot x_0 \cdot y_1. \tag{2.20}$$

Wiring & synchronisation

Every core contains one physical neuron which is time-multiplexed and allows to simulate up to 1024 neurons. Every core supports 4096 incoming and outgoing axons, where each axon can contain multiple synapses. Within the latest version of the `NxSDK` (09/2021), defining multiple synapses per axon is not supported. Therefore, it is necessary to define one axon per synapse. This reduces the number of possible incoming and outgoing synapses to 4 per neuron, if all 1024 neurons are exploited. The wiring within and between cores is shown in Figure 2.8.

After defining the neurons and synapses with their parameters using the `NxSDK`, the Loihi compiler maps the defined network to the hardware. Depending on the size and complexity of the network, this process may take some time. Finally, the chip starts simulating the network. The communication type of Loihi is asynchronous, performed by a mesh protocol (Davies et al., 2018). The mesh protocol is illustrated in an example in Figure 2.9. In this example, source neurons are in two cores, X and Y. Each core has $n = n_x = n_y$ neurons. In time step $t = 1$ the spikes of the first neuron of both cores is distributed to their targets. This continues step by step until all neurons in both cores have transmitted their spikes. In a final step at $t = n + 1$, the protocol sends a barrier synchronization messages, informing the neighbors that all spikes are transferred. After this, the process can start from the beginning.

The next generation: Loihi2

In late 2021, shortly before the submission of this thesis, Intel released the second generation of Loihi. Loihi2 is fabricated with $7\,nm$ technology. The new chip again provides 128 cores, but now allows 8192 neurons per core, instead of 1024, which increases the total amount of neurons per chip to about 1 million. The neuron model is now programmable and synapses can carry information up to a 32 bit size, instead of just a binary spike. The latter feature is called *graded spikes*. In addition, many processes are optimized, especially axon routing is improved, which probably provides larger recurrent networks. How Loihi2 could improve many aspects of this work is discussed in Section 4.2.

Chapter 3

Results

Within the background chapter I introduced the basics of spiking neural networks and the principles of reservoir computing, what experimental neuroscience knows about how movement is processed in the brain and the fundamental basics of neuromorphic hardware. The goal of my research was to find a theoretical model that can describe how voluntary movements could be learned and represented in neural networks. It was intended to simulate the theoretical solution on neuromorphic hardware, robustly and in real-time. Therefore, a secondary goal was to show that the neuromorphic hardware Loihi can successfully support neuroscientific research.

The following result sections comprise three consecutive parts. The first part describes challenges with and solutions for the neuromorphic hardware Loihi. Since the research chip is still in an early stage, a lot of work went into developing frameworks and tools to provide an effective workflow for the provided results. In the second part, I introduce the hierarchical motor selection and execution (HMSE) model that forms the theoretical foundation for the development of concrete simulation models. Spiking neural network (SNN) implementations of the lower layer of this model, the execution layer, are then described in the third part. First, I report promising properties of a so-called anisotropic network, which was then used to train and robustly represent trajectories. The biological plausibility of the anisotropic network is further explored and finally, a reward-based output learning mechanism is presented. With this, the last result section provides solutions that allow the encoding of a movement trajectory in a SNN, based on local mechanisms.

3.1 Making Loihi ready for neuroscientific use

In this work the first generation of the research chip Loihi was used (Davies et al., 2018). The digital chip promises fast simulations of large networks. Both is required for the results provided in the next sections. Despite the fact that these promises are not fully satisfied, as addressed in the discussion, the neuromorphic architecture of the hardware forces the researcher to focus on biologically plausible mechanisms within the network simulations and simplifies the transfer of neuroscientific solutions to applications.

For programming Loihi, Intel provides a `Python`-based SDK (Lin et al., 2018) that was in an early pre-release stage when I started my work and was in a final release stage at the end of my work. While working with the chip and the provided `NxSDK`, I figured out four main issues:

1. The wiring of neurons between cores was not implemented within the SDK and required manual connecting large networks, in the case of this work of up to 256 cores (Section 3.1.1).
2. Prototyping of network models was a difficult task, due to some limitations, e.g. limited input/output capacity or slow initialization time (Section 3.1.2).
3. The theoretical functionality of the chip was insufficiently described, which made developing networks and fixing bugs in the models' code often a difficult task (Section 3.1.2).
4. Loihi is not based on physical parameters (e.g. time in milliseconds or membrane potential in voltage), which makes a translation of existing spiking neural network models to Loihi a difficult task and often requires much tuning effort (Section 3.1.3).

In the following three sections, solutions to all four problems are provided and described. The related sections are mentioned in brackets. Some solutions are only summarized and details can be found in the related preprints (Michaelis, 2020; Michaelis et al., 2021). The developed frameworks were finally used for Loihi implementations described in Section 3.3 and in Section 5.2. In addition, all frameworks are open source and publicly available and therefore provide other researchers with tools for developing on neuromorphic hardware, especially Loihi.

3.1.1 Reservoir computing on Loihi

Contribution statement

This section is a summary of the preprint

> Michaelis, C. (2020). PeleNet: A reservoir computing framework for Loihi. arXiv:2011.12338.

which is attached as Section 6.2. I developed the software and wrote the article. My contribution to this article was 100%. The preprint is licensed under CC BY 4.0.

Summary of Preprint 1

The first problem described above concerns implementing larger networks on the Loihi chip and was solved with a framework called `PeleNet`[1]. I will explain the issue as well as the solution with a concrete example. The system available to us and mainly used, is the so-called Kapoho Bay. It is in a format of a USB-Stick with two Loihi chips, comprising 256 neuron cores. In an application described in Section 3.3.1 we used for example 4500 neurons for a reservoir network and 72 neuron for two pooling layers. These neurons need to be distributed to the neuron cores. A possible solution is to split these 4572 neurons to 20 neurons per core, which would use 228 cores with 20 neurons each and one core with the remaining 12 neurons. Note that it makes sense to exploit as many cores as possible and to reduce the number of neurons per core to increase simulation speed, due to the time-multiplexing on the cores. It is also necessary to split the connection matrix in $229 \cdot 229 = 52,441$ parts, taking into account that one core has less neurons than the others. If it is necessary to monitor the neurons or synapses, which is especially the case while prototyping, then these probes are defined per core or per part of the weight matrix. It is therefore necessary to store the mapping between the originally defined network and units realized on the chip. After a simulation, it is necessary to put all parts together again in order to get probes of the network's behavior that match the expected form.

Note that this issue of distributing neurons and synapses over the chip is a challenging problem for recurrent networks, especially reservoir networks. Networks with a feed-forward structure are much easier to map, since we can for example simply distribute each layer to a separate core. I developed a framework called `PeleNet` to solve this task. It is an object-oriented framework that abstracts away the distribution of networks to Loihi. The user can define a number of neurons with a connection matrix and is not forced to care about the hardware. The network

[1] The name `PeleNet` is inspired from *Pele*, the goddess of volcanoes and fire in the Hawaiian religion (Emerson, 2013; Nimmo, 1986). She is in control over lava and volcanoes and therefore also in control over the Hawaiian volcano Loihi.

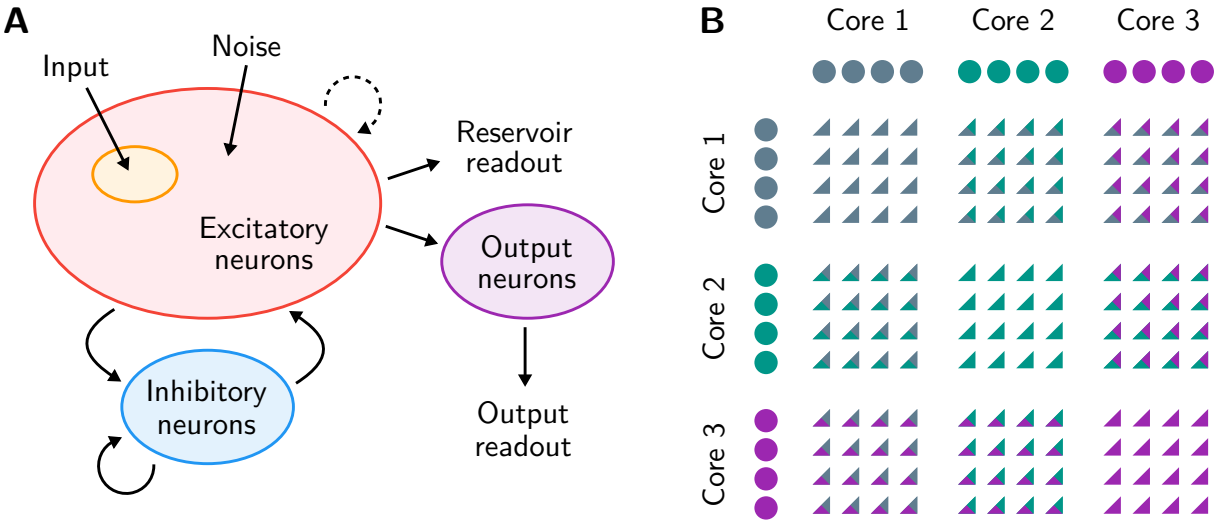


Figure 3.1: `PeleNet` framework target network and distribution problem. **(A)** The target network for the framework. A reservoir network with a pool of excitatory and inhibitory neurons. Input and noise can be given to the pool of excitatory neurons. In addition, it is possible to define a pool of output neurons, connected to the excitatory neurons. **(B)** An example of a connection matrix for 12 neurons. The neurons are distributed to 3 cores and the connection matrix is split in 9 sections.
The figure is slightly adapted from Figure 1 in Michaelis (2020).

is also independent from the board and allows defining networks for arbitrary Loihi boards. `PeleNet` is fully parameter driven and offers detailed logging of the simulations. In addition, many default plots and analysis procedures are included. Finally, it directly allows defining custom experiments such that users can compress their code and concentrate on analysing their results. The framework is introduced in a concise preprint, available on Arxiv (Michaelis, 2020) and appended in Section 6.2. Figure 3.1A gives an overview for which case the framework was designed and Figure 3.1B visually sketches the distribution problem.

3.1.2 Understanding and emulating Loihi

Contribution statement

This section is a summary of the preprint

> Michaelis, C., Lehr, A. B., Oed, W., & Tetzlaff, C. (2021). Brian2Loihi: An emulator for the neuromorphic chip Loihi using the spiking neural network simulator Brian. arXiv:2109.12308

which is attached as Section 6.3 and has been submitted for peer review. I developed the software implementation, Andrew B. Lehr contributed major parts of the mathematical analysis, Winfried Oed performed simulations and provided the figures. Andrew B. Lehr and I drafted and wrote the article, which was jointly reviewed with Winfried Oed and Christian Tetzlaff. My contribution to this article was about 30%. The preprint is licensed under CC BY 4.0.

Summary of Preprint 2

Two further problems for the workflow with Loihi were solved in a second study (Michaelis et al., 2021). This includes a comprehensive analysis of the functional mechanics (issue 3) and providing a Loihi emulator to allow faster prototyping (issue 2). The emulator was used for the development of a novel shortest path algorithm, described in Section 5.1 and the insights from the analysis were important throughout all Loihi-related results within this thesis. The analysis of the neuron model, the synapse model and the traces of the learning variables was already provided in Section 2.3.3. In this section, I briefly describe the motivation behind the `Brian2Loihi` emulator and summarize its functionality.

While neuromorphic chips overcome the von Neumann bottleneck (see Section 2.3.1), they introduce an input/output bottleneck. All information from neurons and synapses have to be transmitted via the bus, which is shown in Figure 2.7B. This limits the input/output capacity by the size of the bus. While theoretically $131,072$ can be simulated per Loihi chip, the capability to read out spikes from neurons is limited to 2912 neurons per chip, which is the limit of the spike counter on the embedded CPUs within Loihi. In situations were new network models are developed it is in most cases vital to know the state of all neurons. This reduces the potential size of the applicable network models drastically. In addition, the more information are monitored and therefore read out from the chip, the slower the simulation becomes. In Michaelis et al. (2020) we showed that reading out spikes from 3600 neurons compared to 72 neurons from the two-chip Kapoho Bay system slows down the simulation by more than 10-fold (from $1.49\,s$ to $15.73\,s$). If more parameters are probed, like e.g. membrane voltages or weights, this slowdown becomes even more significant. Another problem for developing new network models is the initialisation time. The compiler needs to map the intended network to physical cores and needs to provide a routing for all synapses. If all 256 cores of the Kapoho Bay system are

used, this would require the system to handle $65,536$ interconnections between cores. While the simulation time of the network does not depend on the size of the network and scales greatly, the initialization time heavily depends on the size of the applied network. Large networks can easily cause several minutes of initialization time. Note that this issue occurs less in networks with a feed-forward structure, due to a much simplified connectivity structure. Together, this makes developing reservoir networks, especially tuning parameters of those networks, a tedious task. In particular when we compare it with traditional spiking neural network simulators like `Brian`, where the initialization of such networks takes some few seconds and monitoring variables is mostly free, since variables are stored in the memory directly during runtime.

We therefore developed an open source Loihi emulator based on the spiking neural network simulator `Brian`, called `Brian2Loihi` (Michaelis et al., 2021). It was developed based on our mathematical analysis of the functional mechanics of the Loihi chip, summarized in Section 2.3.3. `Brian2Loihi` is able to emulate Loihi neurons and synapses exactly and provides plasticity with small and reasonable errors due to stochastic rounding. The emulator was successfully applied in Section 5.1, as mentioned above, and used for the fitting and tuning dashboard, described in the next section. We hope that with this emulator algorithms can be developed more efficiently and that our emulator approach can be used as a blueprint for future neuromorphic hardware projects. All details are given in the attached preprint in Section 6.3.

3.1.3 Translating parameters from existing models

Contribution statement

The tool was developed in collaboration with Winfried Oed and Andrew B. Lehr. Andrew B. Lehr contributed the idea and a first draft implementation of the manual tuning mechanism. I contributed the idea of a fitting mechanism to translate parameters from `Brian` models to Loihi parameters which was developed by Winfried Oed. This work was initiated and supervised by me and I finally programmed a graphical user interface using a *Dash*[2] app where all tools are merged and finalized within the web app. My contribution to this section was about 40%.

Introduction

The final issue, problem number 4, relates to the translation of parameters from existing neuron models to Loihi. In the past years, models were often defined and used in simulators like for example `Brian` (Stimberg, Brette, & Goodman, 2019) or `NEST` (Gewaltig & Diesmann, 2007). These biological neural network simulators base on physical values like mV for the membrane potential, ms for the time, etc. However, on Loihi parameters are integer values, not related to any physical units. In particular the time is given in discrete time steps instead of a physical time in seconds. This necessitates a lot of tuning if a scientist wants to base his or her research with Loihi on an existing neuron model from the literature.

The anisotropic network, originally developed from Spreizer et al. (2019) and used in Section 3.3.1, required a translation of a neuron model defined in `NEST` to Loihi. Based on the experiences of this translation process, we developed an approach how neuron models can be translated to Loihi systematically. The so-called *Loihi dashboard* aims to solve the issue by comprising two functionalities. First, it contains a method to translate existing physically related models to Loihi parameters. And second, the dashboard allows a manual parameter tuning with graphical feedback.

Manual neuron model tuning

One part of the dashboard provides manual tuning of neuron models. This is shown in Figure 3.2A on the right side. Loihi parameters can be chosen and two traces, current and voltage, are generated with nearly instant speed using the `Brian2Loihi` emulator. Parameters available for tuning are voltage decay, current decay, weight mantissa, weight exponent, weight bits, threshold mantissa and refractory period. Our paper introducing the emulator (Michaelis et al., 2021) contains more detailed information about the background of the parameters. With the

[2] https://github.com/plotly/dash

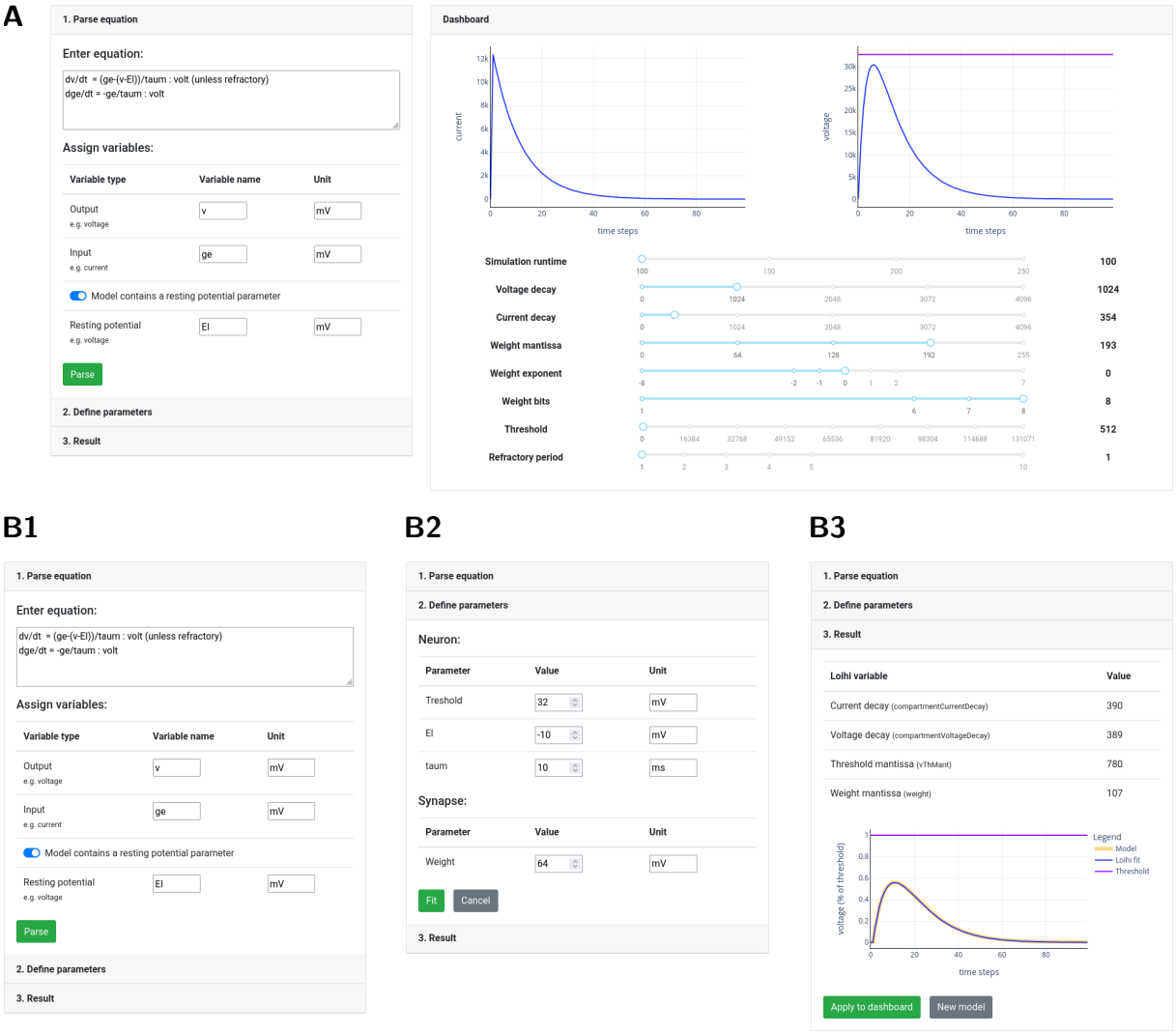


Figure 3.2: A screenshot of the Loihi fitting and tuning dashboard. **(A)** The whole dashboard provides two main functionalities. (left) A neuron model can be defined as an ordinary differential equation. This equation is parsed and current & voltage traces are generated. These traces are used to fit Loihi parameters. (right) The `Brian2Loihi` emulator is used to quickly emulate a neuron behavior depending on Loihi parameters, chosen below of the figures. **(B)** The three parameter fitting steps are shown. Equations and essential variables are defined in (B1). In (B2) the user assigns values to variables parsed from the equation. (B3) shows the results of the fitting. Obtained Loihi parameter are reported and the original model is compared with the approximated Loihi model. The obtained parameters from the fitting can be transferred to the right side of the dashboard for further manual tuning.

manual tuning option for the Loihi parameters, a scientist can easily adjust a neuron model to his or her needs. Obtained parameter are then directly available for an application on the Loihi hardware.

Automatic parameter fitting of neuron models

The other part of the dashboard allows to fit arbitrary neuron models to Loihi. Note that Loihi has a fixed LIF neuron implemented and therefore all other neuron models, different from the LIF model, will be approximated by the fitting algorithm. In addition, some specific parameters

are not available on Loihi, like for example a resting potential which is implicitly zero. Therefore fitting Loihi parameters to existing models is an optimization problem with the goal to maintain the main properties of the neuron model dynamics.

The graphical user interface of the parameter fitting tool is shown in Figure 3.2A on the left side. The process is subdivided into two steps, shown in Figure 3.2B1 and Figure 3.2B2. Within the first step, the user enters a set of differential equations. Moreover, two obligatory and one optional parameter need to be assigned, such that the fitting tool is able to parse the equations correctly. Within the second step, the equation is parsed and parameter values need to be entered by the user. In addition to the parsed parameters, also the firing threshold and the synaptic weight require a value. These values are finally used to generate traces for the current and voltage and to fit Loihi parameters to the given model.

Before explaining our fitting approach in detail, I want to mention that fitting neuron models is not a new field. Different approaches have been developed to fit neuron models based on spike trains (Ladenbauer, McKenzie, English, Hagens, & Ostojic, 2019; Rossant et al., 2011). While these approaches can possibly be applied to fit Loihi parameters, it is not optimal in a sense that the fitting procesure is only based on spike trains, which omits a lot of information. These approaches are better suited to fit theoretical neuron models based on experimental spike data. A more theoretical optimization approach is performed from the `brian2modelfitting`[3] toolbox. This toolbox follows a general approach without specific assumptions, which includes a lot of calculation time for optimizing models with many parameters. While this approach indeed solves the fitting problem also for Loihi, it is relatively slow due to its general approach. Our approach allows much faster optimizations, since we can include constraints given by Loihi that reduces the complexity drastically.

The parameter fitting comprises four main Loihi parameters: the voltage decay δ^v (inverse of the time constant τ_v), the current decay δ^I (inverse of the time constant τ_I), the threshold mantissa v^{th} and the weight mantissa $\tilde{w}$. The Loihi fitter fits those parameters by using three components, a *variable interpreter*, a *trace generator* and the actual *fitter*. The *variable interpreter* parses variables from a differential equation, describing a neuron model. Whereas the *trace generator* uses these extracted variables to simulate traces for the synaptic input (e.g. a current) and the membrane potential (e.g. a voltage). The *fitter* itself first normalizes the traces at the maximum, such that the maximum value of the synaptic input trace and the maximum value of the membrane potential trace reaches 1. In a next step, all 4096 possible current decay values on Loihi are emulated, given a single input spike. The run time is chosen as $100\,ms$ with $dt = 1\,ms$, which results in $n = 100$ simulation steps. Since only a small amount of neurons need to be simulated for a short run time, it performs relatively quick, in less than a second. A mean squared error (MSE) is used to compare the obtained 4096 current traces with the original trace from the given model and the decay value with the lowest error is chosen. The MSE is defined

[3] https://github.com/brian-team/brian2modelfitting

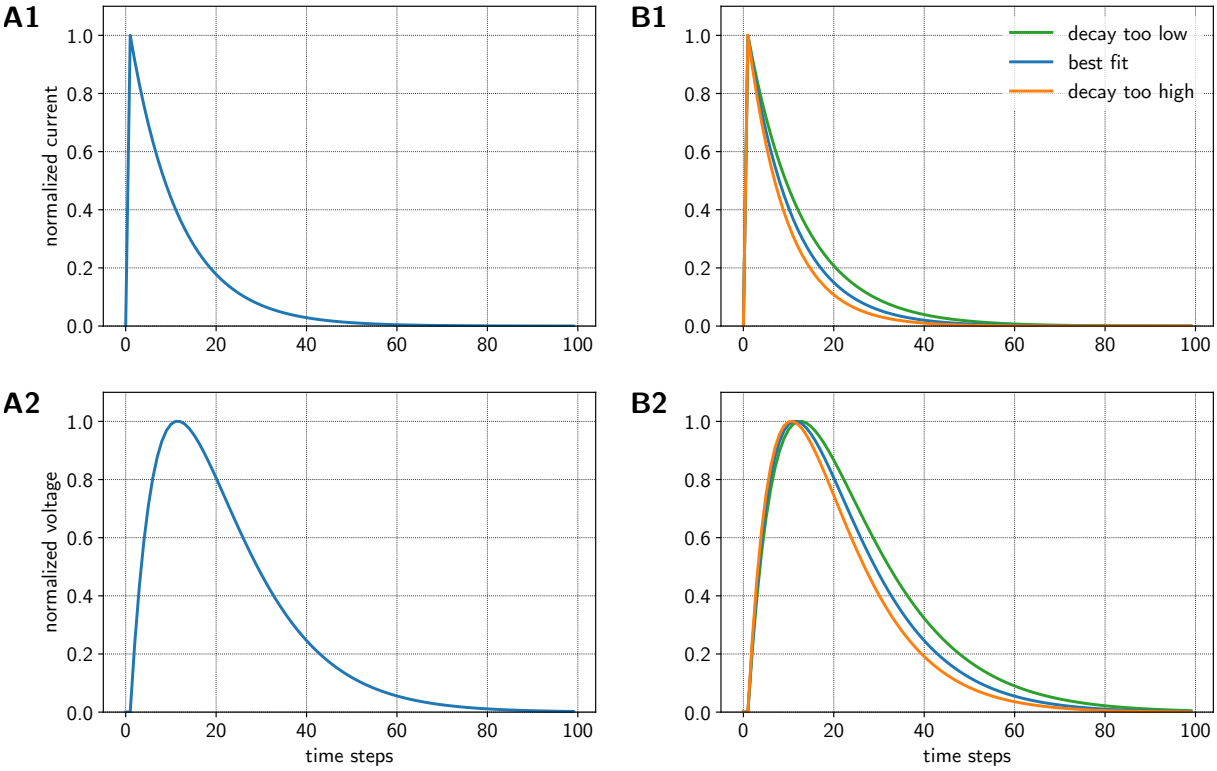


Figure 3.3: Fitting the Loihi decay parameter for the synaptic input and the membrane potential of a LIF neuron. **(A)** The normalized synaptic input (A1) and membrane potential (A2) of the given model, induced from a single spike. **(B)** The fitting procedure on Loihi, again for the decay parameter of the synaptic input (B1) and the membrane potential (B2). Shown is the best fit (blue) and two non-optimal fits (green and orange) out of 4096 possible values for each decay parameter. Note that the dynamics of the membrane potential depend on the dynamics of the synaptic input. Therefore, first, the synaptic input decay parameter is optimized and, afterwards, the decay parameter of the membrane potential is obtained.

as

$$\varepsilon_{\text{mse}} = \frac{1}{n} \sum_{t=1}^{n} \left(x_t^m - x_t^L \right)^2, \tag{3.1}$$

where n is the run time. x_t^m and x_t^L are the values of the model and the Loihi fit at time step t. In this first step, x corresponds to the current I. The current decay fitting, based on a given current trace, is depicted in Figure 3.3A1 and 3.3B1. In a next step, the same procedure is performed with the membrane potential trace, where the current decay value is already given. In case of the voltage trace, all 4096 decay values are simulated and compared with the voltage values from the original model using the MSE from Equation 3.1 again. This time, x equals v. Figure 3.3A2 and 3.3B2 shows the fitting of the voltage decay, based on a voltage trace from a given model.

In a final step, the threshold mantissa v^{th} and weight mantissa $\tilde{w}$ is obtained. First, it is calculated how much percent of the threshold of the original model was reached, when the voltage reaches its maximum. This measure is essential to allow choosing an optimal weight, since the

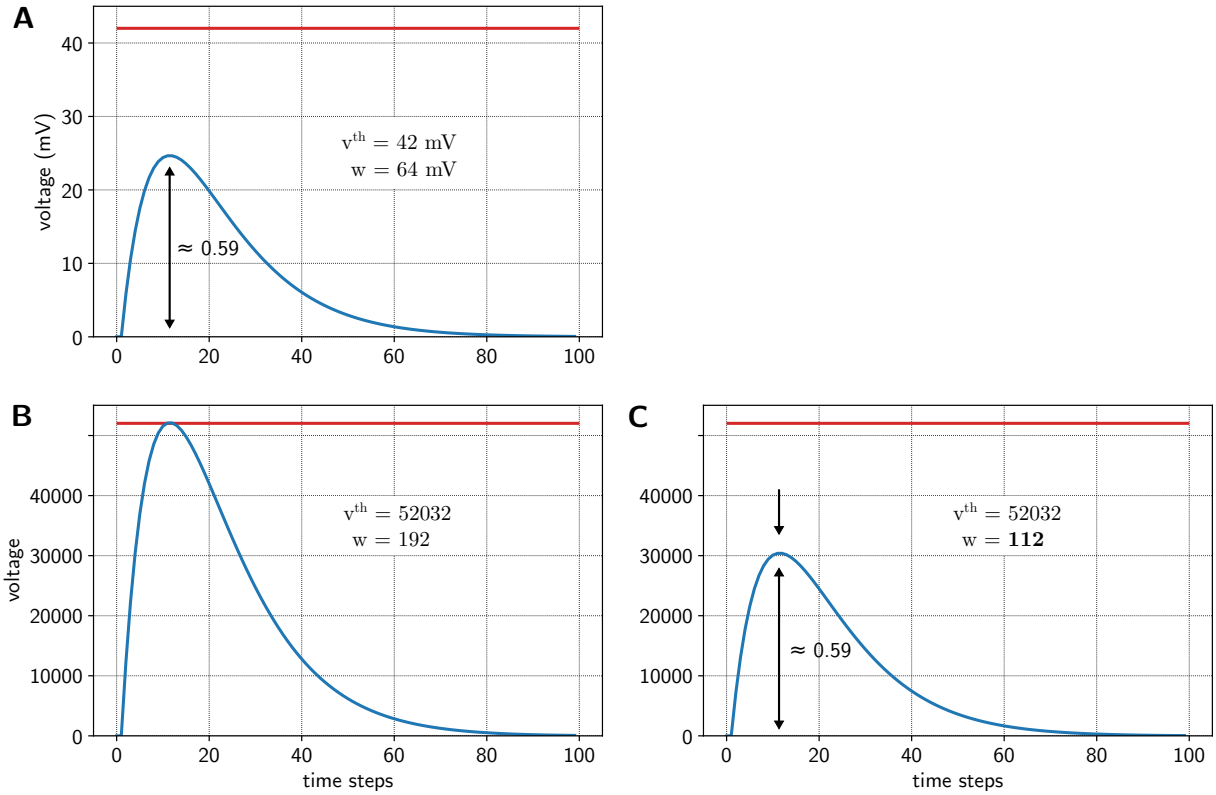


Figure 3.4: Fitting the Loihi threshold and weight parameter. **(A)** The non-normalized voltage trace of the given LIF model, induced by a single input with a given weight. The threshold is marked in red. The ratio between the voltage maximum and the threshold of about 0.59 is kept for the fitting. **(B)** The voltage of the fitted model on Loihi is shown. The weight is manually chosen as $\tilde{w} = 192$ (see details in text) and the threshold is defined as the maximum of the voltage course. **(C)** The weight, inducing the voltage trace, is reduced by the ratio obtained in (A). With this, the dynamics of the Loihi model follows the given LIF model.

spiking behavior of the neuron model depends on both, the threshold and the weight. The threshold ratio therefore sets a reference point for the optimization algorithm. The maximum voltage to threshold ratio is defined as

$$r^{th} = \frac{1}{v^{th}_{\text{model}}} \cdot \max\left(\{v^m_1, \ldots, v^m_n\}\right), \tag{3.2}$$

where v^{th}_{model} is the threshold and $v^m_1, \ldots, v^m_n$ the voltage values from the original model. Note that r^{th} is dimensionless. A visual example is given in Figure 3.4A. After the threshold ratio is obtained, a single simulation is performed that uses the current and voltage decay values from the previous step. Now the threshold mantissa for Loihi is defined as the maximum of the voltage trace of the new simulation

$$\tilde{v}^{th}_{\text{Loihi}} = \left\lfloor \frac{1}{2^6} \cdot \max\left(\{v^L_1, \ldots, v^L_n\}\right) \right\rceil, \tag{3.3}$$

where $\lfloor \cdot \rceil$ denotes an integer operation and $v^L_1, \ldots, v^L_n$ defines all voltage values within the simu-

lation based on the Loihi decay values. Figure 3.4B visualizes how a threshold is obtained from the voltage trace. The weight mantissa $\tilde{w}$ is then defined as the threshold ratio r^{th} multiplied by a fixed weight of 192

$$\tilde{w} = \lfloor \tilde{r}^{th} \cdot 192 \rceil. \tag{3.4}$$

Since the weight mantissa on Loihi is in a range between 0 and 255, the value 192 is chosen such that 75 % of the weight values are available for sub-threshold cases, which is more likely for most model definitions, given a single input. If the given model has above-threshold dynamics, 25 % of the weight values are available to provide a sufficient resolution for this case, which we assume is more uncommon. Given the weight mantissa $\tilde{w} = 192$, the maximum voltage to threshold ratio r^{th} is maintained, which is shown in Figure 3.4C. With this, the fitted Loihi neuron model has a dynamic behavior which is as close as possible to the originally defined model.

In general, this dashboard allows users to consciously choose a neuron model for their needs. Either because it is intended to use an existing model as good as possible or because parameters of a LIF model shall be tuned to fit an intended behavior. In addition, it helps to understand the interplay between the Loihi parameters and can also serve as a helper tool for beginners in the field of computational neuroscience, like students, to quickly get an intuition for relations between essential neuron model parameters.

3.1.4 Summary

This section provided solutions for four issues that emerged while working with the Loihi chip. The `PeleNet` framework is a significant prerequisite for the implementation of the anisotropic network on Loihi, presented in Section 3.3.1. It was also vital for developing the cell assembly formation experiment in Section 5.1. The theoretical understanding of the functionality of the neuromorphic hardware Loihi was important for all experiments performed on Loihi. The emulator, derived from the mathematical description of the Loihi functionality, could successfully be used for the path finding algorithm, introduced in Section 5.1, and accelerated the development time significantly. The fitting and tuning dashboard was developed as a result of my experience with translating parameters from the anisotropic network's neuron model defined in `NEST` to Loihi, described in Section 3.3.1. The dashboard now provides a systematic approach that possibly helps other researchers to translate their models faster and easier. Since all developed software is open source, other researchers can easily benefit and contribute, which hopefully facilitates the utilization of the neuromorphic hardware Loihi.

3.2 The hierarchical motor selection and execution model

Contribution statement

This model was developed together with Andrew B. Lehr. We both conceived the main principles of a hierarchical structure that describes as many features of movement encoding as possible. I did a comprehensive literature research to back the model with existing theoretical and experimental findings. My contribution to this section was about 65%.

Introduction

> "First, what are the guiding principles of the functions of the motor cortex or the motor system in general that can be incorporated into computational models, like feature extraction and spatial pooling in the visual system? Second, how could the three different models, optimality models, recurrent neural network models, and spatial dynamics models, explain the characteristics of motor cortex, and is there any unifying model which embraces the previous models?" Tanaka (2016)

In this section, I introduce the hierarchical motor selection and execution (HMSE) model. This theoretical model is a hypothesis about how movement sequences could be learned and represented in spiking neural networks. The lower layers of this model are further elaborated in the next results section, the higher levels remain open for future research. I will first summarize some theoretical studies providing models for movements and sequences. Based on these theoretical studies and based on the biological findings introduced in Section 2.2, the principles of the HMSE model are described.

3.2.1 Present theoretical frameworks

In order to get an overview over existing computational frameworks related to movement, I start with a review from Tanaka (2016) that suggests to distinguish between three existing approaches. All of them were developed recently within the last 20 years.

- Optimality models, like the optimal feedback control (Todorov & Jordan, 2002)
- Recurrent neural network models, especially reservoir networks, like the LSM (Maass et al., 2002) or the ESN (Jaeger & Haas, 2004)
- Spatial dynamics, which is more related to robotic applications (Graziano, Yap, & Gross, 1994; Pesaran, Nelson, & Andersen, 2006)

Optimality models

Optimality models assume that movement is learned by an optimization process supported by feedback using the sensory system (Tanaka, 2016; Todorov, 2004). There are different definitions of "optimal", but it is mostly understood in terms of a cost function or just a minimized error, depending on the task goal. Note that the theory assumes that an optimization only takes place if the error influences the movement adversely (Scott, 2004). The optimization process itself can be modeled with reinforcement learning (Tanaka, 2016; Todorov & Jordan, 2002). In addition to the cost function, body constraints are considered. For example, muscles can move body parts only in a specific direction, determined by the skeletal structure of the body (Tanaka, 2016). These constraints reduce the degree of freedom and can support optimizing movements. Therefore, the optimal feedback control theory assumes that a high-dimensional movement can actually be simplified when the task and the body are taken into account (Diedrichsen, Shadmehr, & Ivry, 2010). In general, a sensorimotor loop is proposed in which a state estimate is performed using visual and proprioceptive sensory input as well as previous predictions from an internal model (R. Miall & Wolpert, 1996). A new motor command then depends on the state estimate and the given task goal (Benyamini & Zacksenhouse, 2015; Daniel M. Wolpert & Ghahramani, 2000). Diedrichsen et al. (2010) suggests a higher-level state estimation for explaining a variety of more complex movements. This is a valuable addition, since it adds findings from experimental neuroscience to the more behaviorally motivated theory of optimal control. And such additions seem to be important, according to Tanaka (2016) who states:

> "There appears no single model that explains behavioral and neural findings within a unified framework. In addition, most optimality models argue consequences of optimality but do not describe mechanisms of how the motor system 'learns' such optimal control in a biological plausible manner. Finally, there have been no reports of neural correlates of key model components in optimal control."

Recurrent neural networks

The class of recurrent neural networks (RNN), especially reservoir networks, build the second field of computational approaches. The principles of those networks were already introduced in Section 2.1.3. RNN models do not provide any direct hypothesis about how movement is processed, but rather provide a tool for research. The important property of reservoir networks is the ability to perform computational research on a neuronal level rather than on a behavioral level. Researchers aim to achieve a high level of biological plausibility with these networks which often includes the usage of spiking neurons and local learning rules. Some recurrent network simulations with considerable contribution to understanding the motor system were performed for example by Yamazaki and Igarashi (2013), Sussillo, Churchland, Kaufman, and Shenoy (2015) or Pyle and Rosenbaum (2019). Note that in many, if not most, cases the underlying mechanisms are non-local or other less biologically plausible constraints are used to be able to control these highly chaotic systems. This problem of lacking algorithms based on

local mechanisms for spiking neural networks was already mentioned within Section 2.3.1, as a current challenge for the development of neuromorphic hardware algorithms.

Spatial dynamics

The last approach is the field of spatial dynamics. This approach is more related to robotic applications and less to neuroscience (Tanaka, 2016). It is very unlikely that processes suggested by this framework also happen in the brain (Kalaska, 2009; Kandel, Schwartz, & Jessell, 2012). A biologically-related discussion, coming from this approach, is the question if movement trajectories are represented in a Cartesian space or in a joint angle space (Tanaka & Sejnowski, 2013). This comparison is probably too simple and interestingly the dimensionality of a movement representation is probably lower than the joint angle space (Saleh, Takahashi, & Hatsopoulos, 2012). In addition, it seems that movement is rather encoded as velocities than as positions (Saleh et al., 2012), while still both seems to be part of the encoding scheme (Paninski, 2004; Truccolo, Friehs, Donoghue, & Hochberg, 2008).

3.2.2 The hierarchical model

All three existing research domains provide theoretical frameworks for understanding movement processes. In the following, the RNN approach will be the foundation, but aspects of the optimal feedback control theory are also considered.

The here proposed hierarchical motor selection and execution (HMSE) model does not aim for a complete explanation of the neural movement mechanics nor to replace existing hypotheses. The goal is to suggest a connection between the behavioral properties and neural processes which is inspired from and connected to existing theories. The HMSE model should be seen as a starting point that is open for future extensions, ongoing modifications and further specifications.

Note that I use the term "trajectory" in two different settings. A *movement trajectory* describes the path of a body part, like a hand, performing an arbitrary motion. This is also referred to as *output* in the following. A *trajectory* in the context of a neural network describes neural activity in a high-dimensional neural space. For this, we can imagine one dimension for each neuron. If we assume n neurons, activity in the network can be described by a trajectory within an n-dimensional space.

Overview

The basic building block of the HMSE model is the hierarchical understanding of movement representation that was distilled within the last years and decades, summarized by Diedrichsen and Kornysheva (2015). Further experimental evidence for a hierarchical encoding of movements was recently published by Yokoi and Diedrichsen (2019). The biological background was

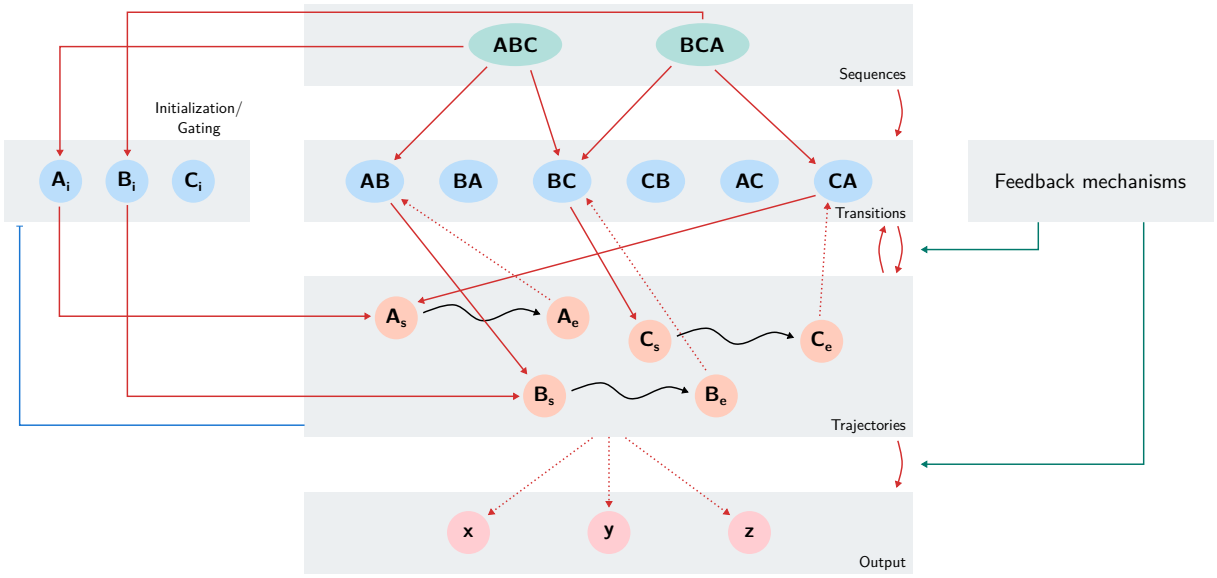


Figure 3.5: The hierarchical motor selection and execution (HMSE) model uses interconnected recurrent spiking neural networks to explain some core functionalities of the motor system. It consists of 5 layers. Four of them are stacked in a hierarchy, one additional layer helps to initialize movements. The output layer stores output functions in a multidimensional space (e.g. 3 dimensions in Cartesian space), while the trajectory layer produces robust, but variable neural activity as basis for the output functions. The transitions layer stores links between actions and the sequence layer selects related transitions. Activating the sequence layer also initializes the movement sequence. Reward based feedback mechanisms, indicated by solid green lines, are used to train the output functions and to connect the ends of the neural activity streams with the link assemblies. Reward based learning rules are marked with a dotted red line, more simple correlation based learning rules (like STDP) are marked with a solid red line. A solid blue line marks an inhibitory connection.

comprehensively introduced in Section 2.2. The HMSE model consists of four hierarchical layers and one additional network for initialization and gating, shown in Figure 3.5. Every layer is a RNN of spiking neurons and the network layers are meaningfully interconnected, either from top to bottom or bidirectional. In nearly all levels a cell assembly formation is necessary and relations between cell assemblies need to be learned. A local learning rule for providing this feature in spiking neural networks is provided in Section 5.2 in the appendix. In this appendix section a learning rule, combining STDP and synaptic scaling, is described that enables a cell assembly formation.

The output layer

The lowest level of the model encodes the movement action, where x, y, z are arbitrary dimensions. This could represent coordinates in a Cartesian space, but could also represent joint angles or muscle velocities. However, the main functionality is to output actions, represented within the weights coming from the layer above. These actions can for example be motor primitives or muscle synergies, as mentioned in Section 2.2.1. We hypothesize that these connections are learned through a reward based feedback mechanism, similar to principles known from the optimal feedback control theory. A possible reward-based output learning mechanism is proposed

in Section 3.3.3. The output layer obviously acts on an executional level.

The trajectory generator

The second last layer produces high dimensional trajectories in the neural space. This layer requires to solve the variability-stability trade-off. Neural activity needs to be sufficiently stable for some hundreds of milliseconds to provide noise-robustness for motor actions. The trajectory needs to stay relatively similar in different contexts, like slight variations in the input and background activity of the network. Concurrently, a high variability of the activity is required to allow learning complex movement functions in the output layer. This trade-off is solved by a spiking neural network with an anisotropic connectivity structure, which is introduced in Section 3.3.1 in detail. With respect to Section 2.2.2, from this layer down, the functionality corresponds biologically best to the primary motor cortex (M1) and brainstem regions. The premotor cortex (PM) probably helps to coordinate the learning process of single actions.

The transitions layer

The second layer from top allows representing links between two actions within cell assemblies (Abeles, 2011). A cell assembly (in Figure 3.5 e.g. CA) is connected to an arbitrary area in the trajectory network which will start a sequence (e.g. A_s) when activated. A feedback mechanism, again possibly similar to optimal feedback control theory, helps to learn a connection between the endpoint of a movement (e.g. A_e) to a second cell assembly within the link layer (e.g. AB). with this, transitions between single actions can be represented and arbitrarily combined.

The sequences layer

Finally, the top layer contains cell assemblies again, encoding a whole movement sequence or chunks. This layer is on a selectional level. It not only starts a sequence through the initialization layer, but also pre-activates associated links. Note that an activation of the cell assemblies within the link layer (e.g. AB) is not strong enough to elicit a starting signal within the trajectory network. Here, the ability of cell assemblies to provide a threshold for a full activation is used (Braitenberg, 1978; Buzsáki, 2010; Guzowski, Knierim, & Moser, 2004). Hence, the activity level is raised below the threshold such that those link assemblies become candidates. Additional activity from the trajectory network is necessary to activate the cell assembly strong enough for inducing an input for the next trajectory. With respect to Section 2.2.2, connections between the first three layers could best associated with the supplementary motor area (SMA). In addition, the hippocampus (HC) could play a role for higher level sequences and consolidation of movement sequences.

The gating and initializing network

Aside of the hierarchical layer is another network, responsible for initializing and possibly also gating activity. Cell assemblies within this network are activated directly from the sequence and are connected to the initial trajectory within the trajectory network. This network is strongly inhibited if any sequence is currently performed to avoid that other sequences can be started during an ongoing execution of a sequence. Referred to Section 2.2.2, this part could functionally meet the gating property of the basal ganglia (BG).

3.2.3 Conclusions

The HMSE model has the ability to explain some core features of the motor system. Compared to a feed forward structure e.g. in synfire chains (Abeles, 2009), the HMSE model can store multiple sequences in parallel (e.g. ABC and BCA). These action sequences can even contain the same motor transitions (e.g. BC). To the best of my knowledge, this is a first proposal for providing this flexibility. Moreover, all parts of the model can be associated with the function of specific brain regions. At least the division of tasks in sequence learning, trajectory representation, initialisation and gating are separated parts within the network as it is in the brain.

Cell assemblies within the network are used for binary information only. In Section 5.2 and 4.2 I argue that cell assemblies are highly appropriate for storing the status of a process, but are less suited for complex computations. For the latter, we provide an approach based on an inhomogeneous local connectivity structure that solves the variability-stability problem, described in the introduction. The combination of those two concepts, the so-called anisotropic network and the classical cell assembly theory, build the neural foundation of the HMSE model. It allows not only to represent sequences of high dimensional movement trajectories, but also to control and gate those sequences avoiding an interference between movement executions.

Beside the flexibility, some restrictions exist. At first, repetitions are problematic within this model. A link between A and A can easily be learned, but it would result in an endless loop. The only possible way around this problem is to define a distinct trajectory for every repetition, which is not efficient. Another possible solution is to define the repetition as a single action, which may only be reasonable up to two or three repetitions. It could also be that repetitions play out on a higher level, on top of the sequence layer, which is currently just not considered. However, a more elegant solution for this problem is required and would be an improvement to the model.

In addition, single motor actions need to be in a specific order. This problem is also mentioned by Krakauer et al. (2019). In the case of the HMSE model, actions have to end in a reasonable position. If action A ends in a position where action B cannot start, the sequence makes no sense. Considering for example playing the piano and assuming a finger down *and* up as one action, all possible sequences for a piano piece are possible. But defining a finger down as a single action and a finger up as another action, actions cannot arbitrarily be connected. For

example, it is obviously not possible to put a finger down twice. Since it is only a matter of definition, we define reasonable actions as *canonical actions* and assume their existence.

As already mentioned in the introduction of this section, this model should be seen as a basic framework which is meant to be extended by future research. Three parts could possibly be further developed. On a higher level, functionalities from other brain regions needs to be included, like coordination from the CB or sensory feedback from the somatosensory cortex. On a lower level, specific functionalities are not modeled in detail yet, which includes for example the feedback mechanisms. But currently available local feedback learning mechanisms for spiking neural networks are unfortunately rare (Taherkhani et al., 2020). Finally, it remains open to test interactions between the network layers, which requires much larger simulation capacities. This final goal of building a large simulation with all parts together is further elaborated in the discussion, in Section 4.2.

The HMSE model is an abstract concept that requires further specification. This work aims to suggest concrete solutions to three major parts of it. First, a solution to the variability-stability trade-off is presented, which allows to produce robust but sufficiently long neural activity, described in Section 3.3.1 and Section 3.3.2. In addition, I introduce an algorithm for the output learning problem, using a reward based mechanism in Section 3.3.3. As a further result, I provide an approach of how assemblies can emerge within recurrent neural networks with local learning rules in Section 5.2, which could be a supporting feature on several layers of the model. All provided mechanisms are purely local and are implemented on the neuromorphic hardware Loihi.

3.3 Encoding motor execution

The previous section introduced a big picture of how movement sequences could be encoded in the nervous system. Here, I want to focus on the lower part of the HMSE model that relates to the trajectory generation and output learning of single actions. For this, two problems need to be solved. First, a variable but robust activity is necessary within the trajectory layer. Second, an output learning mechanism is required that uses the high-dimensional spiking activity from the trajectory layer as input and stores low dimensional output functions in the weights between the trajectory and the output layer.

In the first section, a paper summary shows how a so-called anisotropic network can be used to store three dimensional movement trajectories, which is useful for understanding the brain, but may also have applications in robotic control. Further, I demonstrate the biological significance of the anisotropic network and present how the properties of this network type can even be improved. In a last section an output learning mechanism is introduced. A reward based learning rule is used to train output weights that translate spiking network activity to actual movement trajectories.

Since Loihi, and the brain as well, requires local-only mechanisms, the following solutions do not use any global information from the network. Note that this is a very strict approach which is rarely applied in other studies. The usage of local-only mechanisms hardens the task significantly, but also contains the chance to gain new insights into the dynamics of neural networks and may provide inspiration for future experimental studies.

3.3.1 Storing movement trajectories using the anisotropic network

Contribution statement

This section is a summary of the paper

> Michaelis, C., Lehr, A. B., & Tetzlaff, C. (2020). Robust trajectory generation for robotic control on the neuromorphic research chip Loihi. *Frontiers in Neurorobotics, 14*. doi:10.3389/fnbot.2020.589532

which is attached in Section 6.1. Andrew B. Lehr provided simulations based on the neural network simulator `NEST`. I developed the implementation on Loihi, performed simulations, the data analysis and produced the figures. I drafted and wrote substantial parts of the article with support from Andrew B. Lehr. The article was jointly reviewed with Andrew B. Lehr and Christian Tetzlaff. My contribution to this article was about 65%. The paper is licensed under CC BY 4.0.

Summary of Paper 1

A central challenge for storing spatio-temporal information, like movements, in spiking neural networks is to find a solution to the variability-stability trade-off (Pehlevan et al., 2018). The solution to this problem requires two properties: sufficient variability within the spiking activity and sufficient robustness against noise. The simulated spike trains can be considered as a data set that can be used to learn functions based on that data. These spiking data needs to provide sufficient information to be able to adapt parameters, e.g. output weights, such that a function can be represented. If these data do not provide enough variability, functions cannot be represented based on that data. At the same time, the spiking behavior within the network needs to robust to noise, e.g. background noise or noisy inputs. In a context of movement, the neural network needs to provide sufficient variability to allow the representation of a movement trajectory, but also stability to be able to reliably reproduce this trajectory, independent of the current state of the network. A chain of single neurons, for example, would provide a perfectly reliable spiking pattern, but only very little information. A traditional cell assembly with hundreds of neurons would provide sufficient information to learn anything, but would follow a highly chaotic behavior in presence of noise that would not provide any reliable replay of a movement trajectory.

While robust and stable activity in a neural network can be obtained for a few spikes, and therefore a few milliseconds, it is much harder for a timescale of hundreds of milliseconds, considering recurrently connected spiking neural networks as highly chaotic systems (Brunel, 2000; London, Roth, Beeren, Häusser, & Latham, 2010; Sompolinsky, Crisanti, & Sommers, 1988;

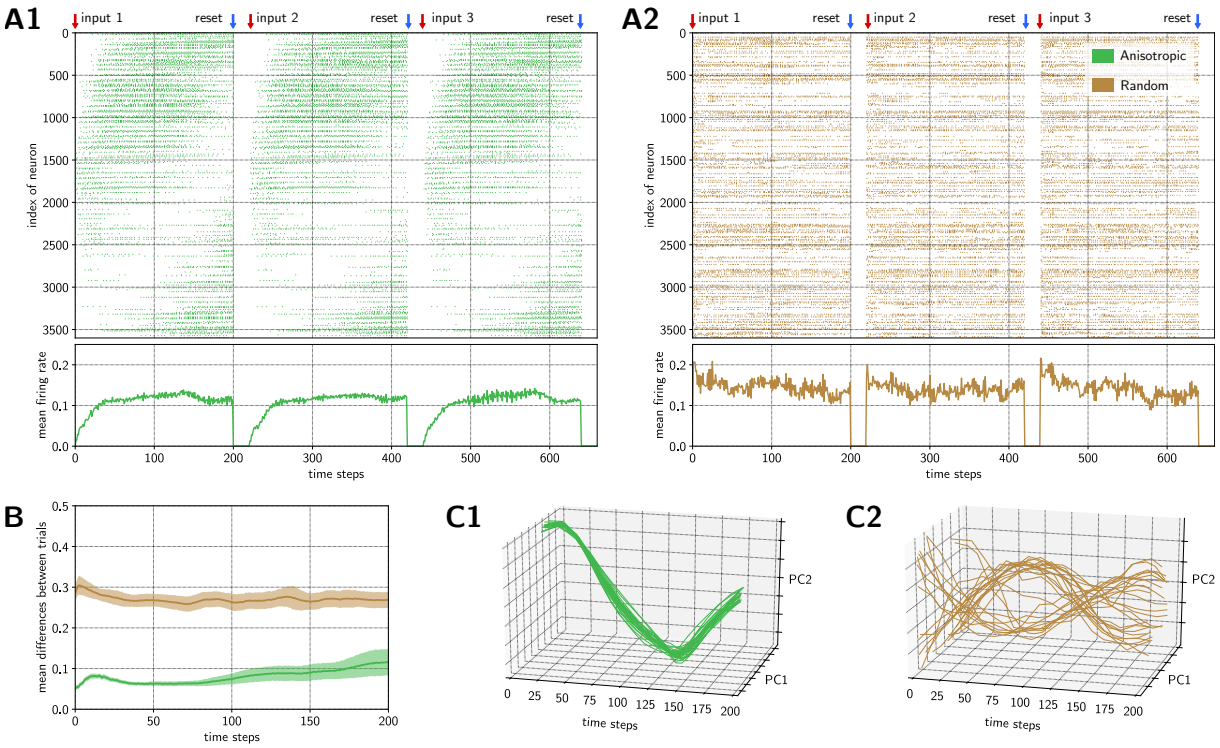


Figure 3.6: The Loihi implementation of the anisotropic network is robust to varying input conditions, while a randomly connected network is not. **(A)** Three exemplary trials out of 25 trials are shown for a simulation based on an anisotropic connectivity structure (green) and with randomly initialized weights (brown). The anisotropic structure clearly has a stream-like spiking pattern, where the firing rate starts slowly until it reaches a constant rate. The randomly connected network shows a Poisson-like spiking pattern, where the firing rate starts directly at a high level. **(B)** The solid lines show the mean difference between all trial combinations and the shaded area indicates the standard deviation of the trial-to-trial differences for the anisotropic (green) and random (brown) network. **(C)** In the reduced space of the first two principal components of the activity of all 25 trials over time for both networks, we can clearly see that the spiking pattern of the anisotropic network are very similar between trials, while the activity in the randomly connected network differ much more between trials.
Figure taken from Michaelis, Lehr, and Tetzlaff (2020).

van Vreeswijk & Sompolinsky, 1996). A recently developed biologically-inspired spiking neural network model (Spreizer et al., 2019), the so-called anisotropic network, is a promising candidate to solve this task in a biologically plausible way. We identified the underlying inhomogeneous connectivity structure of the synapses as the central principle for maintaining stability. While Spreizer et al. (2019) stimulated the network in different settings and obtained several wandering bumps of spiking activity within the topology, they did not test the network's ability to produce stable spiking pattern. We stimulated the network with a noisy input burst in several trails and tested the stability with different measures. We hypothesized that the single input will initiate a spatio-temporal activity which is reliable enough to finally learn output functions.

We successfully implemented the anisotropic network on Loihi, using the `PeleNet` framework introduced in Section 3.1.1. With this implementation, we could show that the variability is high enough to learn a set of different three-dimensional movement trajectories and at the same time sufficiently robust to noisy inputs. Results compared to a traditional randomly connected network revealed significantly better stability on a timescale of two seconds. We then developed

an architecture, including an additional spiking pooling layer, that allowed a fast read out of the spiking activity. By reducing the number of output neurons but maintaining the spatial structure, the pooling layer regularized the parameter space of the output learning and helped to avoid overfitting. Conventional linear regression was then used to train three dimensional movement trajectories, which could be used for robotic control.

Taken together, we show that the anisotropic network on Loihi reliably encodes sequential patterns of neural activity, each representing a robotic action. The spiking patterns allow the generation of multidimensional trajectories on a control-relevant timescales and the pooling layer enables a real-time simulation of the movement. The study not only presents a new algorithm that allows the generation of complex robotic movements on state of the art neuromorphic hardware, but also presents a possible encoding scheme for movements in the brain.

While the anisotropic connectivity structure is a promising approach, the biological plausibility for this kind of network is to date not clearly shown. In the next section we will profoundly analyse the properties of the anisotropic network. We present strong indication for its biological plausibility and confirm that the stability is maintained under a biologically plausible parameter setup. Using STDP strengthens the ability to solve the variability-stability trade-off to a remarkable extent.

3.3.2 The anisotropic network is biologically plausible and can be improved using STDP

Contribution statement

Results presented within this section were performed together with Finn Schünemann, Benjamin Schulz and Andrew B. Lehr. In this section we used a `Brian` implementation of the ansiotropic network, provided from Leo Hiselius[4]. Benjamin Schulz worked on analyzing motifs in the anisotropic network and Finn Schünemann applied STDP to the ansiotropic network, both performed analyses. Andrew B. Lehr and I supervised the project. I took preliminary results from the collaborative work and performed further comprehensive analyses in order to provide a coherent result for this section. All figures in this section are created by me. My contribution to this section was about 35%.

Introduction

For the understanding of higher processes of the nervous system, especially movement, it is vital to understand how information is encoded in spiking neural networks. In particular, the issue is how activity in a noisy and high dimensional space can be reduced to a meaningful and reliable low dimensional trajectory. In other words, it is still an open question how a recurrently connected neural network is able to reliably generate movements or recall memories, especially in a noisy environment. In the literature this problem has been referred to as the variability-stability (Michaelis et al., 2020) or the robustness-flexibility problem (Pehlevan et al., 2018), as mentioned in the previous section. Recent studies have suggested a number of ways in which recurrent networks may produce rich trajectories that are stable to noise (Hennequin, Vogels, & Gerstner, 2014; Laje & Buonomano, 2013; Pehlevan et al., 2018; Spreizer et al., 2019; Vincent-Lamarre, Calderini, & Thivierge, 2020).

One recent and promising candidate, the anisotropic network (Spreizer et al., 2019), was used by Michaelis et al. (2020) to demonstrate that it can solve the variability-stability problem. Nevertheless, a match between the behavior of the network and experimental data is still sparse. The biological motivation for the anisotropic network structure comes from studies that found asymmetric dendritic and axonal arbors of single neurons (Jiang et al., 2015; Mohan et al., 2015; Xu et al., 2016). In addition, nearby neurons seem to share a similar orientation (Li et al., 2012). These results for single neurons clearly motivate initializing network connections in a way that neurons have a shift and nearby neurons share the same direction. On a network level, Patriarchi et al. (2018) found heterogeneous dopamine signals during a visuomotor learning task, which may cause a wave like expansion of activity (Hamid, Frank, & Moore, 2021). These

[4] https://github.com/leohiselius/spreizer-net

results support the notion of an underlying anisotropic structure with similar effects observed in the anisotropic network. However, it is still mostly unclear to which extent this inhomogeneous connectivity structure agrees with data, especially on a network level.

Here we demonstrate that the anisotropic network matches experimental connectivity structures remarkably well. First, we compare motifs of neuron connections between experimental findings (Song, Sjöström, Reigl, Nelson, & Chklovskii, 2005) and a simulated anisotropic network and demonstrate a region in parameter space where data and simulation agree. Second, we show that the ability of the anisotropic network to produce stable but variable activity is not optimal under biologically plausible parameters, but can be regained and even significantly improved by applying synaptic plasticity to the network. While adapting synapses influences the distribution of the weights and improves the ability of the network to produce such activity, it leaves the anisotropic connectivity structure untouched.

The anisotropic network

The principle of the anisotropic network is introduced in Michaelis et al. (2020) and depicted in Figure 2 of the publication, attached in Section 6.1. Therefore, I provide only a brief summary in order to recapitulate the main principles. The anisotropic network connects neurons locally, based on a Gaussian distribution around a particular neuron. The standard deviation σ defines how far synaptic branches can reach from the particular neuron. In addition, the center of the Gaussian distribution is shifted, such that a neuron connects with a tendency in a chosen direction. How the direction of this shift is chosen is the crucial aspect of the anisotropic network. The direction is chosen by Perlin noise (Perlin, 1985), which results in an anisotropic landscape where nearby neurons share similar directions. Such a landscape is shown in Figure 3.8A.

The landscape of the directions for the shift of the Gaussian distribution determines the connectivity structure and defines which synapses are realized, but is independent from the synaptic weights. The synaptic weights have a constant value in the original implementation of the anisotropic network (Spreizer et al., 2019). The connectivity structure provides a spiking activity that forces activity to evolve in a stream like manner. Note that the network is folded at the edges to a torus-shaped manifold. Activity can therefore evolve without boundary effects. Examples based on a single short input are shown in Figure 3.9A. In addition, Figure 7 in Spreizer et al. (2019) provides examples for this behavior based on background noise where several bumps of spiking activity wander within the manifold at the same time.

Comparing biological and computational motif statistics

In a first study we wanted to find out if the anisotropic connectivity structure matches connectivity motifs from experimental studies. Song et al. (2005) counted such motifs for connections between two and three neurons. Motifs define pattern of possible connections. For example, two neurons A and B can have no connection, one connection from A to B and/or one con-

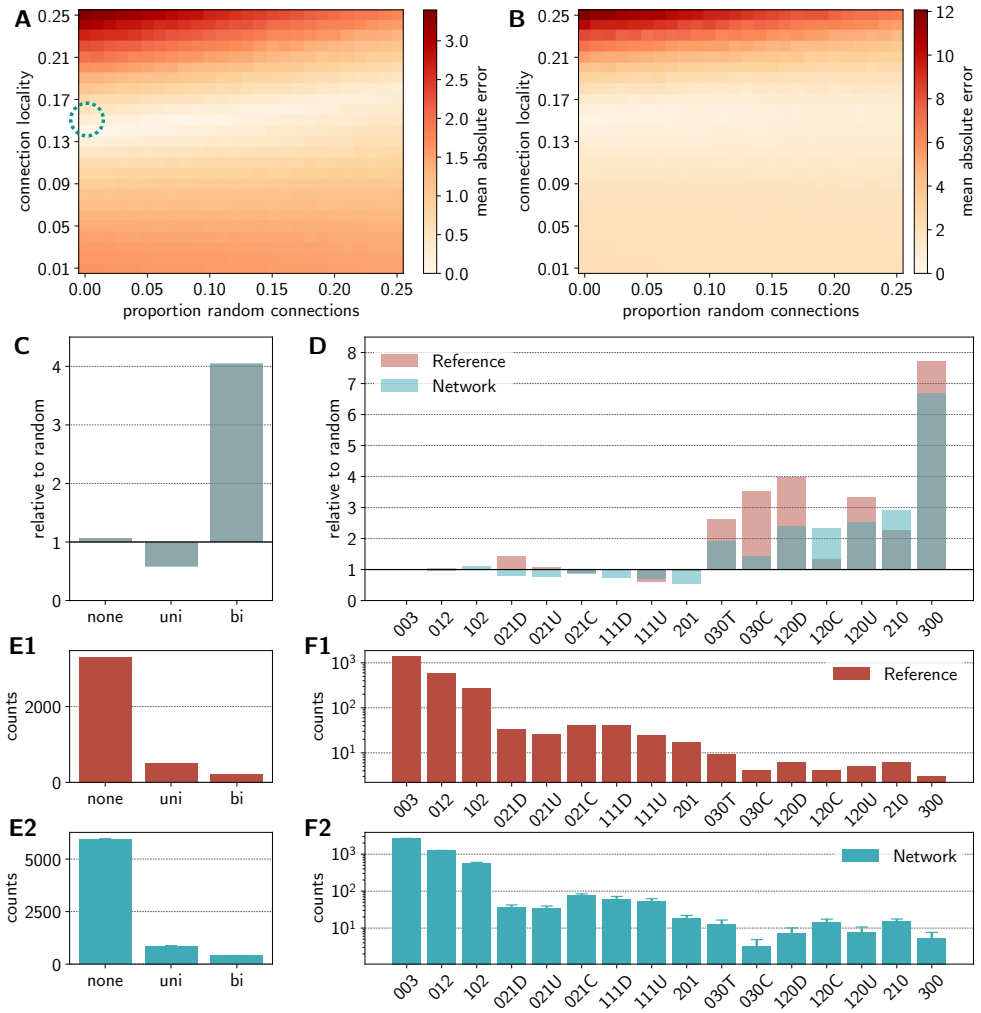


Figure 3.7: Parameter scan to match motifs between experimental data from Song, Sjöström, Reigl, Nelson, and Chklovskii (2005) and simulated data from the anisotropic network. **(A)** Mean absolute error between motif counts in the network's connectivity structure relative to random motifs for all *two* neuron motifs. The error depends on the proportion of random connections, replacing local connections, and the locality of the synaptic connections. The parameters providing the lowest error are marked with a green circle. **(B)** Mean absolute error between motif counts in the network's connectivity structure relative to random motifs for all *three* neuron motifs. **(C)** Relative motifs for all *two* neuron motifs in the network for the parameter set with the lowest error. **(D)** Relative motifs for all *three* neuron motifs in the network for the parameter set with the lowest error based on the two neuron motifs. **(E)** Absolute motif counts for the experiment (red) and the network simulation (blue) for the *two* neuron motifs. Note that the total amount of samples differs between the network and the biological reference. **(F)** Absolute motif counts for the experiment (red) and the network simulation (blue) for the *three* neuron motifs. Note that for readability reasons, error bars are only shown for absolute counts.

nection from B to A. Therefore three cases are possible: no connection between two neurons, an unidirectional connection or a bidirectional connection. The same can be applied to a three neuron system. The number of patterns in a neural sample can be counted and relative values can be obtained. In a final step, the number of occurrences can be put in relation to the number of expected motifs, based on the assumption that neurons are connected in a purely random fashion. If, for instance, the case *no connection* has a value of 1 in relation to the expected occurrence probability, it would mean that in the neural sample the amount of *no connection* is

exactly randomly often. If the value is higher than 1, it would occur more often than expected and if the value is lower than 1, the amount is lower than expected from random connections. From the experimental study from Song et al. (2005) we know that many connection types differ significantly from the expected occurrences. For a two neuron system, unidirectional connections are less frequent and bidirectional connections are more frequent than expected from a random connectivity assumption.

We varied parameters of the anisotropic network, obtained motifs from the connectivity structure for each parameter and compared them with the results from the biological experiment in order to examine the ability of the anisotropic network to match connectivity structures from biological neural networks. We expected that, in addition to the local connections of the anisotropic network, some space-independent random connections could perhaps improve the match between the computational model and the experimental results. Therefore, we systematically replaced a specific proportion p_r of the locally connected neurons by random connections. In addition, we varied the standard deviation σ of the Gaussian distribution and therefore varied the locality of the connectivity structure. We hypothesized that this would influence the type of connections. A scan over several σ and p_r was performed for 10 different random seeds for each pair. For all these cases an anisotropic network was initialized and the motifs of the two and three neuron motifs were counted. These counts from the anisotropic network were put in relation to the expected counts. We then calculated the absolute mean deviation between the relative values from the experimental study and the relative values from our computational model. Note that we chose all other parameters of the anisotropic network besides σ and p_r according to Spreizer et al. (2019). The only exception is the probability for a connection that was chosen as $p = 0.116$ according to Song et al. (2005), compared to the original value of $p = 0.05$ and the ratio of recurrent inhibition and excitation that was chosen according to the `Brian` implementation[5] with $g = 4$.

Figure 3.7A shows the mean absolute deviation of this scan for the two neuron system and Figure 3.7B for the three neuron system. A green circle marks the best value for the two neuron system with $p_r = 0.0$ and $\sigma = 0.14$. Interestingly, the anisotropic network provided the best fit for the two neuron system if no local connections are replaced by random connections. But still low error values can be obtained with random connections replacing local connections. Figure 5.8 in the appendix shows several examples for cases where a random proportion is applied.

Figure 3.7C compares the best parameter pair for the theoretical and experimental results for the two neuron system and Figure 3.7D shows the comparison for the three neuron system, also based on the best pair for the two neuron system. The comparison of the two neuron system clearly indicates an almost perfect match, whereas comparing the three neuron system reveals some deviations. But it is important to mention that the error for three neuron motifs with many connections (right side, `030T` to `300`) is high in the experimental data (Figure 4 in Song et al., 2005) and a good match for those motifs therefore not expected. However, in tendency those

[5] https://github.com/leohiselius/spreizer-net

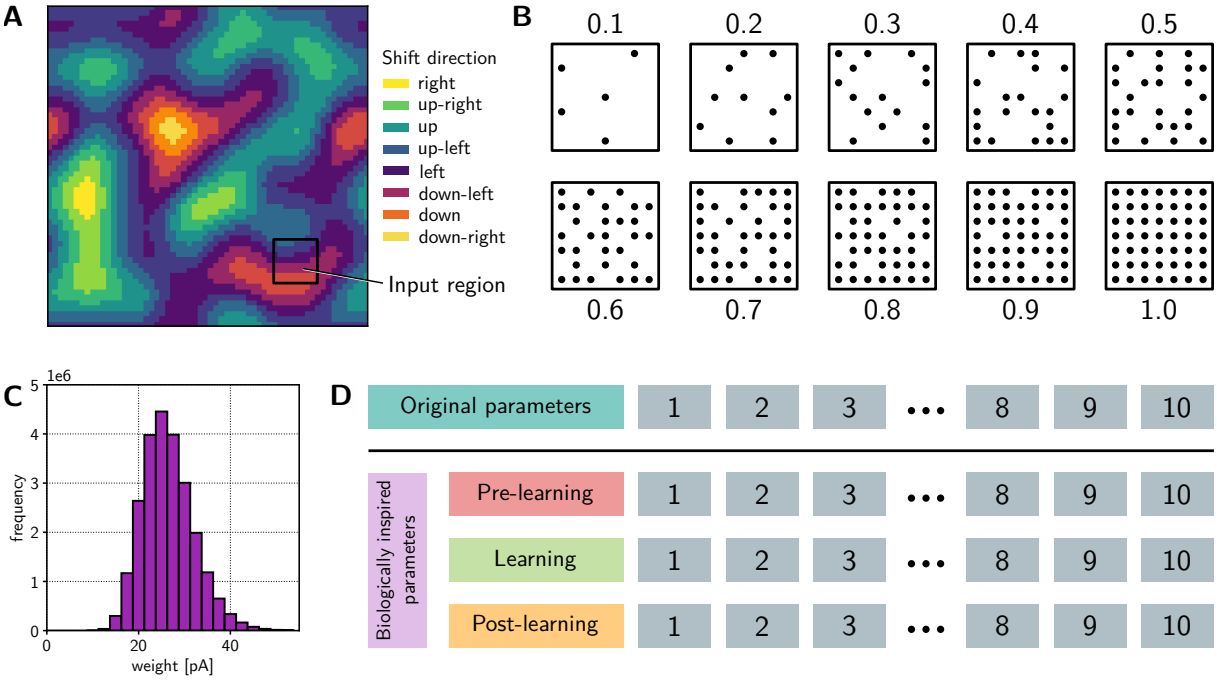


Figure 3.8: Experimental setup for comparing experimental and biological parameter sets for the anisotropic network. **(A)** The anisotropic landscape based on Perlin noise. Each color indicates a direction for the shift of the Gaussian distribution that connects neurons. An exemplary input region for network stimulation is illustrated with a black box. **(B)** The input region is stimulated with a different share of neurons, from 10% to 100%. Neurons which are actually stimulated within the input region are randomly drawn. **(C)** Example of a lognormal weight distribution for initial weights of the network. **(D)** The protocol for comparing the original and the biological parameter set for the anisotropic network. For each case (original, biological pre-learning, biological learning, biological post-learning) 10 trials are applied. Each experiment is performed with all 10 different input shares from (B). While the input share is fixed for an experiment, the actually selected neurons within the input region differ between trials.

motifs with many connections are over-represented, as it is the case in the experimental study. Figure 3.7E shows a comparison between absolute values of the two neuron system. Note that these values are not in relation to the number of samples and can therefore only be compared based on their distribution. Figure 3.7F shows the absolute counts for the three neuron system. Despite some differences between relative values of the three neuron system (as in Figure 3.7D), a Kolmogorov–Smirnov test revealed no significant difference between the distributions of the absolute count data ($D_n = 0.25, p = 0.716 > 0.05$), indicating a similar result also for the three neuron system.

With this we could show that the anisotropic network is indeed able to reproduce experimental data to a remarkable extent. Interestingly, it was sufficient to tune only one parameter, the locality parameter σ, in order to obtain a match with experimental data. With this we have found an additional strong indication that the anisotropic network provides a biologically plausible connectivity structure.

All following experiments denoted as *original* are performed with parameters from Spreizer et al. (2019) and experiments denoted as *biological* are performed with $\sigma = 0.14$. In the next part, we will improve the network even further by choosing a more biologically plausible weight

distribution and apply synaptic plasticity. We will perform a variety of analyses in order to compare the original and biological parameter set with focus on its ability to provide robust spiking activity.

Lognormal weight distribution and the stimulation protocol

The parameters that enabled the anisotropic network to match the experimental motifs from Song et al. (2005) were chosen as the *biological* parameter set for analysing the stability of the network. In addition to the adaptation of the connection probability and locality, the synaptic weights were initialized based on a lognormal distribution, instead of equal weights. Experiments have shown that the excitatory postsynaptic potentials (EPSP) of several different brain areas follow a lognormal distribution (Fukai, Klinshov, & Teramae, 2014; Lefort, Tomm, Sarria, & Petersen, 2009). The lognormal distribution for the weight w is defined as

$$p(w) = \frac{1}{\sigma_w w \sqrt{2\pi}} \cdot \exp\left(-\frac{(\ln(w) - \mu_w)^2}{2\sigma_w^2}\right), \tag{3.5}$$

where μ_w is the mean and σ_w is the standard deviation of the distribution. The mean equals the previous constant weight value $\mu_w = 10 \cdot e$ and the standard deviation is $\sigma_w = 0.2 \cdot \mu_w$. An example for the initial lognormal distribution of weights is shown in Figure 3.8C.

To verify the stability of the network, we developed a protocol that is similar to the protocol applied in Michaelis et al. (2020). Within the network, an input region is chosen. From this input region a share of $r_I = 0.1$ up to $r_I = 1.0$ neurons was randomly chosen and stimulated, examples are shown in Figure 3.8B. We hypothesized that the stability will increase, the larger the share of activated neurons in each trial. In Michaelis et al. (2020) we applied about 4% noise in the input, while in this study, the noise level is between 0% and 90% and additionally accompanied by moderate background noise ($\mu_{\text{noise}} = 150\,pA$ and $\sigma_{\text{noise}} = 50\,pA$). The protocol is again similar to our previous study and is based on 10 trials per experiment and a simulation time of $200\,ms$ per trial. Every experiment has a fixed input share, but the actual stimulated neurons are drawn individually between trials. The experiment for the biological parameter set was extended to a learning phase. Therefore, 10 trials were applied to a static network, a further 10 trials were performed with spike-timing-dependent plasticity (STDP) and finally, an additional 10 trials were simulated based on the trained weights, but without plasticity again. An overview is given in Figure 3.8D. All experiments were performed five times with different random seed values to be able to average over different network initializations. We hypothesized that STDP could improve the stability of the network. While the connectivity structure stays untouched by the learning rule, the weights adapt to perhaps support the connectivity structure and improve the stability of the spiking activity.

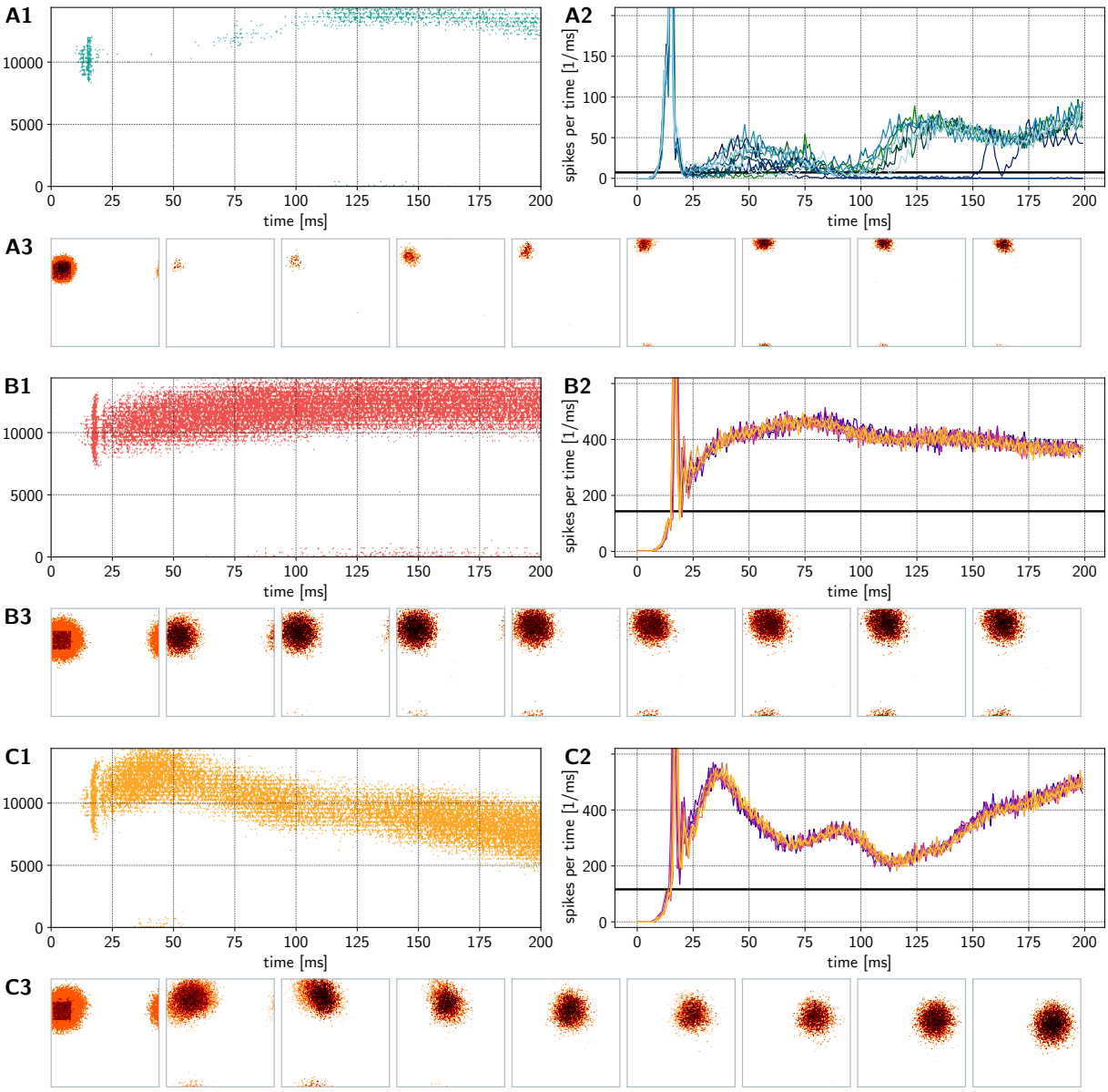


Figure 3.9: Spike trains, firing rates & topological propagation of activity in the anisotropic network. The activity is based on a trial with an input share of $r_I = 0.9$ for **(A)** the original parameter set, **(B)** the biological parameter set before learning & **(C)** the biological parameter set after learning.

Biological parameter set provides stable spiking activity

In a first step, we will have a look into the spiking behavior of both the original parameter set and the biological parameter set obtained from the motif study extended by the lognormal distribution of synaptic weights. The spiking activity of the anisotropic network with both parameter sets is compared in Figure 3.9. A single input burst with an input share of $r_I = 0.9$ initiates spiking activity that results in a stream like propagation in all setups. Part A shows the spike trains, rates and the topological distribution of binned spiking activity over time for the original parameter set. The biological setting, shown in part B of the figure, provides a broader and stronger activity, determined by the larger σ value. After learning, in part C, the activity is similar to the pre-learning case, but has more dynamics in the spike rates and the bump seems to

move faster. Note that we quantified the velocities of the bumps below. In addition, Figure 5.10 in the appendix shows the biological case, but without the lognormal distribution. With equal weights, the activity is less stable. This shows that the lognormal distribution of the weights, in addition to the locality value, is an important factor for a strong and stable propagation of spikes.

The lognormal weight also have an effect on the STDP learning. If the same weight is chosen for every synapse, STDP adapts weights only slightly, as depicted in Figure 5.11C. Some lower and higher weights emerge, but only in close proximity to the initial weight value. If weights are initialized with a lognormal distribution, changes emerge more broadly, as shown in Figure 3.10C. This indicates that a higher variety in the initial weights facilitates the weight adaptation performed by STDP.

We can conclude that the biological parameter set performs well. The spiking activity does not fade out too fast, nor does it increase without bound. The lognormal distribution has a significant effect, in addition to the biologically chosen parameters σ and p.

STDP increases velocity and keeps reliability

In this section, we want to focus on two more specific measures, the velocity of the bump of spiking activity wandering through the network and the so-called *ignition rate* that defines how many trials are actually successfully "started" without dying out after a short time. We expect that both measures depend on the input share. The more neurons are activated, the higher is the chance for a successful ignition and the higher is the velocity.

I defined an ignition within a trial as successful if the spike rate stays above 60% of the median spiking activity of the whole experiment for at least $50\,ms$. In addition to this relative threshold, I defined an absolute threshold of 10 spikes per ms for the case that activity in the trial dies out directly. The additional absolute threshold prevents the threshold to drop to zero, or close to zero, which would possibly define a trial as successfully ignited, even though it provides nearly no activity. Taken together, the threshold is defined as 60% of the median or, if lower, at least 10 spikes per ms within the network. Note that this measure is widely used as a poverty threshold (Foster, Seth, Lokshin, & Sajaia, 2013; Spicker, 2012). We therefore somehow assume that "poor" trials do not ignite. For several trials this measure was manually checked by watching videos generated from the topological spiking activity and the ignition measure correctly captured the actual behavior for all checked samples. In Figure 5.9 in the appendix spike rates for the original and biological parameter set are shown for several different input shares. Black horizontal lines indicate the respective threshold. If the spiking rate does not stay above that line for at least $50\,ms$, the corresponding trial is considered as not successfully ignited. The examples show that in almost all cases it can clearly be distinguished between trials which are silent right from the beginning or keep a spiking rate above the threshold for a sufficient time.

Figure 3.10A shows the mean ignition over all trials and network initializations for the original

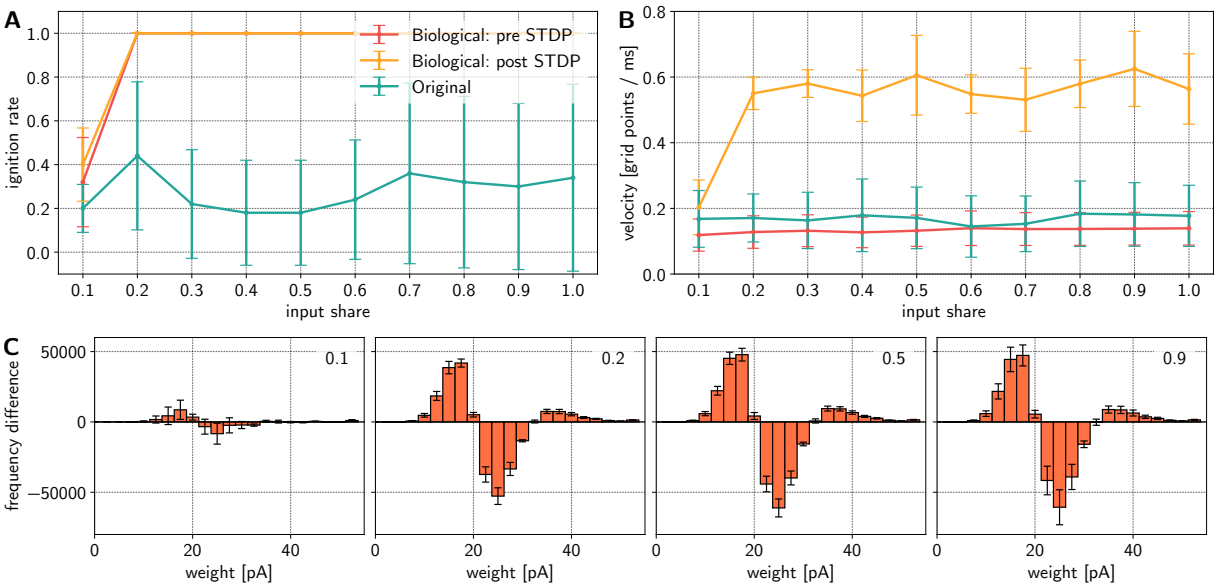


Figure 3.10: Ignition rates, velocities & weight changes of the anisotropic network. **(A)** Ignition rate depending on the input share. A rate of 1 indicates that the activity in all trials was sustainably started by the input. A value of 0 means that the activity in all trials died out shortly after the input was given. **(B)** Velocity of the activity bump moving within the network depending on the input share. How velocities are obtained is described in the text. **(C)** Synaptic weight differences between initial weights and weights after STDP was applied. The weight differences are shown for four input shares: 0.1, 0.2, 0.5 and 0.9 (from left to right).

and the biological parameter set. While the ignition rate is relatively low with a high standard deviation for the original parameter set, the biological parameter set provides perfect ignitions at least if the input share is 0.2 or larger. Interestingly, the biological parameter set provides a perfect ignition rate for pre- *and* post-learning trials. In Figure 5.11A in the appendix we can see that if the synapses are initialized with equal weights, instead of lognormally distributed weights, the biological parameter set does not provide perfect ignitions. They are, however, always on a very high level at about 90% but do not provide a perfect ignition in every case. Interestingly, learning has no influence on the ignition rate again.

In addition to the ignition rate, we obtained the velocity of the bump of activity that is wandering through the topology of the network. We used DBSCAN (Ester, Kriegel, Sander, Xu, et al., 1996; Schubert, Sander, Ester, Kriegel, & Xu, 2017) to detect clusters within the spiking activity. We binned the activity and classified the neurons belonging to the cluster of activity. The mean of the neurons' coordinates revealed the center of the cluster. From bin to bin, we measured the distance between the clusters' centers and calculated the velocity of the bump.

Figure 3.10B shows that the velocities of the bumps do not change when biological parameters are chosen instead of the the original parameters. Therefore, the speed of the activity within the network is not determined by the connectivity structure but rather by the synaptic weights. STDP accelerates the activity streams considerably. This was already qualitatively observed earlier within the topological activity plots of Figure 3.9. With the velocity measure, we are now able to quantitatively support this impression. This is further supported by the fact that the acceleration does not depend on the initial distribution of the weights. Moreover, when all

weights are initially equal, STDP is able to increase the velocity, even though this happens to a lower extent, as shown in Figure 5.11B.

Therefore, our initial hypotheses are not fully correct. It is true that for a very low input share of 0.1, both ignition rate and velocity are low, but there is no proportional dependency between the input share and the ignition rate nor the velocity. The effect of STDP on the velocity is a surprising yet valuable finding. We believe that a low velocity may negatively influence trajectories which are trained on the spiking activity of the network. A slowly moving activity bump reduces the resolution in the time dimension for a targeted output function. The ability to increase the speed simply by applying a simple learning rule to the weights of the network provides us with a powerful tool to tune such networks for our needs. Finally, the perfect ignition rates for the biological parameter set, even independent from learning, are a highly appreciated cornerstone for further experiments.

Biological parameters improve stability

In a further step of the analysis we tested the robustness of the anisotropic network within both, the original and the biological setting. The goal is to compare the stability between trials, i.e. for different samples from the same input share. Two evaluations were performed: we calculated differences between trials and performed a principle component analysis (PCA). For obtaining the differences between trials, we first binned the spiking activity with a sliding window of 25 ms. We further calculated the differences for all bins between all pairs of the 10 trials, resulting in 45 pairs. Within each bin, we divided the pair difference by the sum of the bins in order to normalize the differences.

Results for one experiment per input share are shown in Figure 3.11B. The differences between trials are much higher for the original parameter set, compared to the biological parameter set. The learning has no considerable effect on the differences. The differences in the beginning of the trials are higher for lower input shares. Lower input shares vary inherently more and, therefore, higher differences in the beginning are determined by the input itself and not necessarily by the moving bump. After the input has started the network stimulation, the differences end up at the same low level, independent of the input share. In Figure 3.11A we calculated the total mean difference over all bins and initializations. We can observe that the input share has no influence on the differences for the original parameter set. The total mean differences for the biological parameter set are significantly lower and slightly decrease with higher input shares, which is probably due to the input itself as argued before. This behavior can be observed for trials before as well as after the STDP learning. The differences after learning are in general slightly lower than before learning, but again, learning has no clear influence on the differences. Figure 5.12A in the appendix compares pre- and post-learning trials with equal weights. The differences are much higher and a lot closer to the differences based on the original parameters. We can therefore conclude that the lognormal distribution primarily influences the similarity between trials, which is reasonable, since presumably some few strong synaptic connections guide the spiking activity, which is a known consequence of lognormal weights (Teramae &

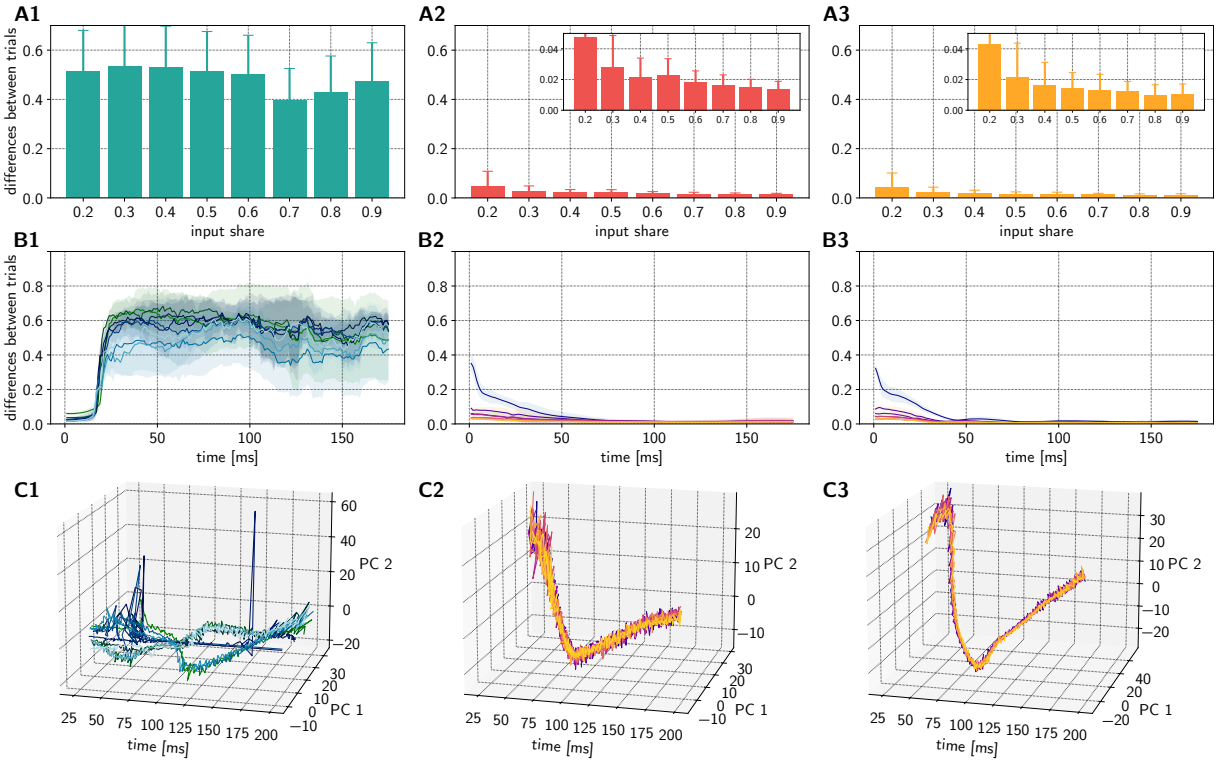


Figure 3.11: Trial-to-trial differences & PCA for activity in the anisotropic network. Results are shown for the original parameter set (left), the biological parameter set before learning (middle) and the biological parameter set after learning (right). **(A)** Mean differences between trials depending on input shares. **(B)** Differences between trials for one experiment per input share. **(C)** The first two principle components of the activity over time within a PCA space for an experiment with an input share of 0.9.

Fukai, 2014).

In a second analysis, we computed a PCA of the trials. The first two components of an experiment with an input share of 0.9 are shown in Figure 3.11C. The biological parameter set provides a common trajectory in the space spanned by the the first two principle components, which is not the case for the original parameter set. Comparing these results with Figure 5.12C reveals that the lognormal distribution again supports the stability.

In conclusion, the biological parameter set, especially the lognormal distribution of the synaptic weights, provides a high degree of reliability between trials when compared to the original parameter setting. We will use this stability to train an output function within the next and final part.

Biological parameters and STDP provide almost perfect generalization

In a final step, I applied a linear ridge regression with a L2-penalty of 0.005, to regularize the high amount of parameters, since each of the 14, 400 excitatory neurons in the network causes a parameter. The spike trains were binned with a sliding window of 25 ms width. These binned spikes were used as input variables for the linear regression. A sine wave was used as output function, but other functions should perform similar. The output function was trained on 9 out

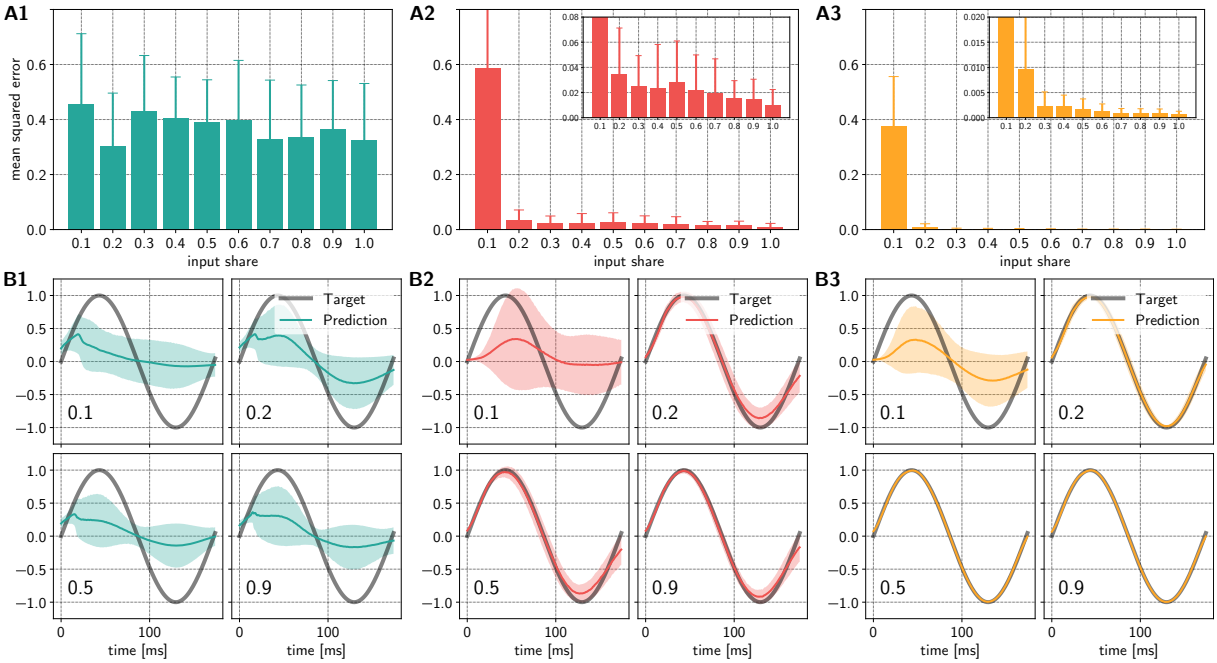


Figure 3.12: Learning a sine wave function based on the activity of the anisotropic network. Results are shown for the original parameter set (left), the biological parameter set before learning (middle) and the biological parameter set after learning (right). **(A)** Mean squared errors between predicted and target function depending on input shares. Inlays show errors on a smaller scale. **(B)** Predicted and target functions for one experiment and four input shares: 0.1, 0.2, 0.5 and 0.9 (from left top to right bottom).

of the 10 trials and tested on the remaining one, resulting in a generalization task which is equal to the generalization task applied in Michaelis et al. (2020). Therefore, the deviation between the predicted output function and the target function indicates the degree of robustness of the underlying spiking activity.

Figure 3.12B shows examples for experiments with input shares of 0.1, 0.2, 0.5 and 0.9. With the original parameter set it was not possible to reproduce the output function. In contrast, the biological parameter set is able to provide sufficient robustness to noise in order to estimate the output function relatively well, at least if the input share is larger than or equal to 0.2. Qualitatively comparing the estimated sine functions indicates that the application of STDP improved the results even more. Figure 3.12A compares mean squared errors and therefore quantifies the performances over several network initializations. While the errors for the predictions based on the original parameter sets are constantly high, the predictions for the biological parameter set reveal a much better performance, at least if the input share is larger than 0.1. Inlays show mean squared errors on a smaller scale to illustrate differences between input shares for the biological parameter set. For the pre- and post-learning case, the errors decline slightly with increased input share. Compared to the overall low errors, however, this effect is very small, especially for the post-learning case.

The output learning results show incredible stability and robustness. At the same time, the variability in the data is sufficient to learn an output functions within a time scale of 200 ms. In this last analysis, the biological parameters, obtained from matching the experimental motifs, as

well as the lognormal distribution of synaptic weights play a vital role. Figure 5.13 shows that without the lognormal weight distribution, the error is still lower than in the experiment with the original parameters. Nevertheless, the lognormal distribution improves the performances drastically. Finally, in this last output learning experiment, the weight adaptation using STDP also takes effect and complements the task to a nearly perfect result, even under considerably noisy conditions.

Conclusion

The anisotropic synaptic connectivity structure enables streams of spiking activity that were further facilitated by STDP. The existence of such streams was demonstrated by analysing the trial-to-trial differences and performing PCA. The strengthening property of the synaptic weight adaptation is reflected by the increased velocities and indicated by the PCA as well. If a certain threshold is reached, the activity propagates through the prepared stream. The ignition rate analysis revealed that an input share of 0.2 was sufficient to reliably start a sustaining activity in the network. The application of the output learning task yields a high performance, nearly independent from the input share after a threshold of at least 20% stimulated neurons within the input region was reached. The additional improvement of the output learning after the weight adaptation by STDP is likely due to the increased velocity after learning, but further experiments are required to get more evidence to claim a causal relation.

Taken together, we can understand this behavior akin to a kind of spatio-temporal cell assembly or continuous attractor (Spalla, Cornacchia, & Treves, 2021). Cell assemblies have the property of becoming active when a certain share of neurons is activated, but store binary information as a state memory (see Section 5.2 of the appendix for an introduction to cell assemblies). Cell assemblies can be switched on or off (see e.g. Levy, Horn, Meilijson, & Ruppin, 2001), depending on excitatory or inhibitory stimulation, and can therefore be controlled. In the case of the anisotropic network, combined with the biologically motivated parameters, a small proportion of input is required to activate a stream, similar to the threshold in cell assemblies. In contrast, streams within the anisotropic network are in tendency transient, provide spatio-temporal activity for movement trajectories and cannot be controlled in a sense that manually switching between an on or off state is not possible. In the discussion section I will extend this perspective and combine it with the other results from this thesis, especially the HMSE model.

3.3.3 A local reward-based plasticity rule learns output functions

Contribution statement

All results within this section were performed by me. I did the experiments, the analysis and created the figures. My contribution to this section was 100%.

Introduction

In the previous two sections, a solution to the variability-stability problem was suggested. In our publication (Michaelis et al., 2020) we used linear regression to learn output weights from the anisotropic network that represented the three dimensional movement trajectory. However, the method of least squares, used to estimate parameters of the linear regression, is obviously non-local. A new alternative local learning mechanism is required that would finally enable a full-functioning solution for the execution layer of HMSE model. A recent comprehensive review from Taherkhani et al. (2020) summarizes many state-of-the-art plasticity methods for spiking neural networks. According to this review, only one local-only plasticity rule is available that is applicable to the here required output learning task. This solution is called remote supervised method (ReSuMe) and was introduced by Ponulak (2006). Originally it was used to learn precise spike times of neurons. I used this rule to learn an output function, based on random spikes and based on input from the execution model of Section 3.3.1. First, I show that the ReSuMe learning rule can be modified to successfully run on Loihi. Furthermore, I introduce a specific sampling technique to use the learning rule to be able to learn a function. Analysing the approach based on random spiking input reveals conditions for a successful application of the learning rule. Finally, it is determined what is required to use the spiking activity from the anisotropic network as input for the learning rule.

The ReSuMe learning rule

The ReSuMe learning rule was introduced in a thesis from Ponulak (2006) that includes an extensive analysis of the parameters. A later publication contains a variety of applications (Ponulak & Kasiński, 2010). The basic principle assumes an input neuron with input spikes $S_i(t)$ and an output neuron with actual spikes $S_o(t)$ and target spikes $S_d(t)$, input and output neuron are connected with a synapse. Figure 3.13A shows the neuron setup. Note that I use the terms *input neuron* and *pre-synaptic neuron* as well as *output neuron* and *post-synaptic neuron* interchangeably. The rule states: increase the synaptic weight between the input and output neuron if a spike should occur, but does not take place. If a spike occurs, but in a time slot where it should not be, reduce the synaptic weight. In both cases, the increase or reduction is weighed by the synaptic trace of the input spike. If a spike occurs at the correct position, both

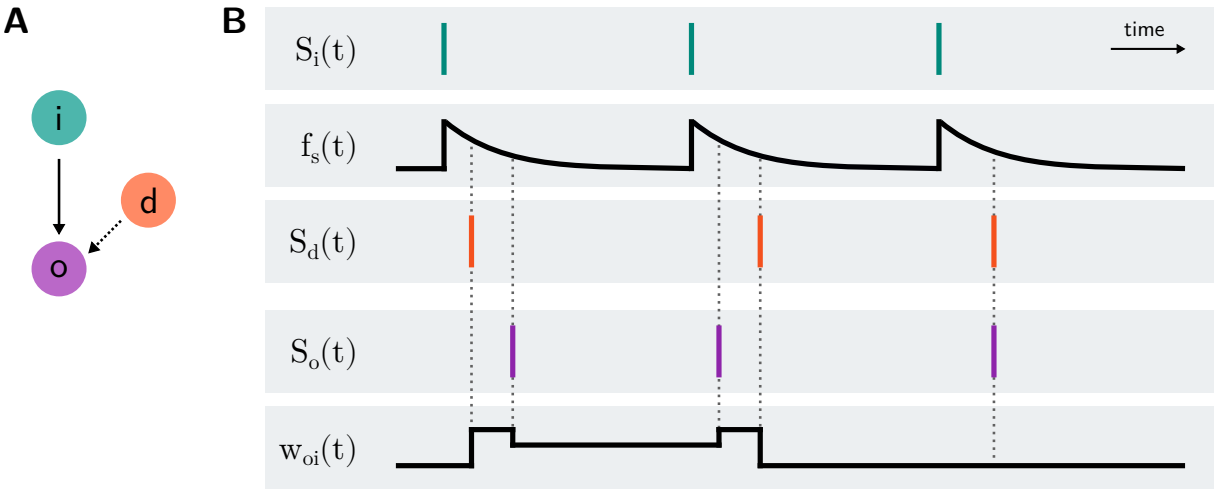


Figure 3.13: The learning rule of the remote supervised method (ReSuMe). **(A)** The neuron configuration. A pre-synaptic input neuron (green) is connected to a post-synaptic output neuron (purple). The orange circle represents the target spikes. **(B)** The main principle of the learning rule. Input spikes $S_i(t)$ elicit eligibility traces $f_s(t)$. If a target spike $S_d(t)$ is given at a position were no actual spike $S_o(t)$ occurs, the weight $w_{oi}(t)$ is increased, weighted by the trace of the input spike. If an actual spike is fired, but no target spike is intended, the weight is decreased. This again is weighted by the trace of the input spike. If actual and target spike match, no weight change is applied.

rules cancel out and no synaptic weight change is applied. In conclusion, if a spike happens directly after the targeted position, in sum, the weight is increased which in turn increases the chance that the spike occurs slightly *earlier* in the next iteration. If a spike takes place directly before a targeted spike, the weight is in sum decreased which increases the chance that the spike occurs slightly *later* next time. The effect of the ReSuMe rule is shown in Figure 3.13B.

According to Ponulak and Kasiński (2010), the rule for a change of the weight between the input and the output neuron w_{oi} is formally defined as

$$\begin{aligned}\frac{d}{dt}w_{oi}(t) &= \Big(S_d(t) - S_o(t)\Big) \cdot \Big(a_d + f_s(t)\Big) \\ &= \Big(S_d(t) - S_o(t)\Big) \cdot \left(a_d + \int_0^\infty a_{di}(s) S_i(t-s) ds\right),\end{aligned} \tag{3.6}$$

where a_d is a non-Hebbian term that induces a constant change to the weight, depending on the difference between the target spike and the actual spike, but independent of an input spike. According to Ponulak and Kasiński (2010), this parameter can be chosen as zero without harming the performance. $a_{di}(s)$ is an exponential kernel that determines the shape of the trace of the pre-synapatic neuron. It is defined as

$$a_{di}(s) = A_{di} \cdot \exp\left(\frac{s}{\tau_{di}}\right). \tag{3.7}$$

Here, $s = t_i^f - t_d^f$ is the difference between the pre-synaptic and the target spike, where f is the

number of the spike. In addition, $A_{di} > 0$ is an amplitude and $\tau_{di} > 0$ is a time constant.

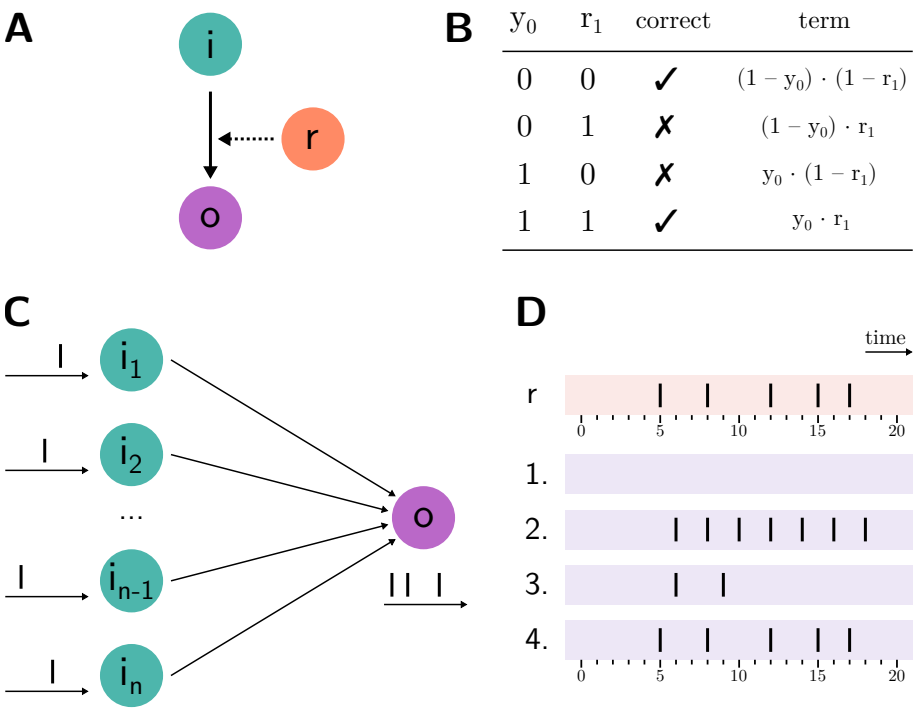
The ReSuMe rule was successfully applied in several studies. Ponulak already showed first results within his dissertation (Ponulak, 2006), demonstrating that spike trains can successfully be learned. Two years later, Glackin, McDaid, Maguire, and Sayers (2008) applied the principle of the rule within a feed-forward network structure and classified different data sets. It was further shown that the ReSuMe rule is able to produce results similar to the tempotron model (Răzvan V. Florian, 2008; Gütig & Sompolinsky, 2006). Further, Ponulak and Kasiński (2010) showed results for learning spike trains under several conditions, they applied a classification task and a spike shifting task. Wang, Belatreche, Maguire, and McGinnity (2011) developed the concept further and applied a new rule, based on the ReSuMe rule, for different classification tasks. In recent years it was further adapted (Luo, Zhang, Wang, Wen, & Zhang, 2018) and even applied to deep learning in spiking neural networks (Taherkhani, Belatreche, Li, & Maguire, 2015a, 2015b).

One of the tasks, analysed by Ponulak and Kasiński (2010) and most important for this work, is how a single post-synaptic neuron can be trained to perform a specific spiking pattern. It turned out that a sufficient number of pre-synaptic input neurons is necessary to provide enough parameters to adjust the weights. In addition, the learning procedure works nearly perfectly if every pre-synaptic neuron spikes only once over the whole time. If it spikes several times, the learning requires many more iterations and is is less likely to reach a perfect solution. Therefore, I hypothesize that the result for the output learning task will probably not be perfect if input from the anisotropic network is used, since in this setting some neurons in the network spike very frequently.

Implementing the re-ReSuMe learning rule on Loihi

The implementation on Loihi requires a new formulation of the learning rule. First, the non-Hebbian term is set to $a_d = 0$. As mentioned before, this should not have a negative effect on the learning rule. In addition, a reward spike is used to inject the target signals to the synapse. This principle is shown in Figure 3.14A. Unfortunately, Loihi currently does not support a reward spike indicator r_0 for learning rules directly. This functionality is officially available (see Davies et al., 2018), but currently not supported. Therefore, a small trick is applied and the eligibility trace variable of the reward r_1 is used for this purpose. The amplitude of the trace is set to 1 and decay is chosen as maximum. With this, the trace emits a 1, if a reward spike is given and 0 otherwise and behaves exactly like a reward spike indicator.

The modified learning rule depends on the post-synaptic output spike, the reward (target) spike and the pre-synaptic spike trace. Since the method is now based on reward, the adapted method is called reward-based remote supervised method (re-ReSuMe). To understand the new learning rule, we first focus on detecting the conditions in which a weight change will be applied. The post-synaptic spike indicator y_0 can be used to indicate an output spike, while r_1 is used for a target spike. With these two variables available, four different cases can be extracted.



y_0	r_1	correct	term
0	0	✓	$(1 - y_0) \cdot (1 - r_1)$
0	1	✗	$(1 - y_0) \cdot r_1$
1	0	✗	$y_0 \cdot (1 - r_1)$
1	1	✓	$y_0 \cdot r_1$

Figure 3.14: Reward-based remote supervised method (re-ReSuMe) applied to a single spike train. **(A)** The principle structure of how re-ReSuMe is applied to Loihi. A reward spike indicates the target spike for the output neuron. **(B)** The possible spike arrangements and related mathematical terms are listed. Only terms for mismatches are necessary for the learning rule. **(C)** The first experiment using the re-ReSuMe rule. Input neurons (green) are connected to an output neuron (purple). Reward spike generators are not shown. The pre-synaptic neurons fire exactly one spike at a random time position. **(D)** The learning progress for a single output spike train. The letter r labels the target spikes. Four trials are necessary until the synaptic weights are finally able to reproduce the targeted spike pattern at the output neuron.

Figure 3.14B shows all cases. If the post-synaptic spike and the target match (either 0/0 or 1/1), the spike is at a correct position and no weight change is required. But according to the re-ReSuMe rule, if a target spike should exist, but no actual spike occurs, a weight change is necessary. This case is detected by the term $(1 - y_0) \cdot r_1$. We also need to change the weight when a spike occurs, but no spike should happen, which is indicated by $y_0 \cdot (1 - r_1)$. With those two indicator terms, the learning rule can finally be defined as

$$dw = +c \cdot u_0 \cdot x_1 \cdot (1 - y_0) \cdot r_1 - c \cdot u_0 \cdot x_1 \cdot y_0 \cdot (1 - r_1), \tag{3.8}$$

where c is the learning rate and u_0 forces an evaluation of the terms in every time step. u_0 makes the algorithm more inefficient, but is unfortunately required since the event-based variable r_0 is not available, as mentioned above. x_1 is the eligibility trace of the pre-synaptic input neuron.

A comprehensive parameter search for related Loihi parameters is provided in the appendix in Figure 5.14. To test the implementation on Loihi, a simple example is applied. 20 input neurons spike one time at a random position within 20 time steps. Those neurons are connected to an output neuron. The synapses get reward spikes indicating the target spikes. The setup is illustrated in Figure 3.14C. The goal is to learn a spike train with 5 spikes within the 20 time

steps. All weights are initialized with a weight of 0. By allowing positive and negative weights, weight values on Loihi can vary between -254 and $+256$, but the resolution is reduced to 2, since one bit is used to encode the sign of the weight.

Figure 3.14D shows the learning procedure. The target spikes, labeled with an r, are shown in the first row with orange background. Rows with a purple background represent learning trials. Within the first trial, no spike occurs at all. The second trial provides too many spikes. But already after four trials, the target spike train is matched and the weights are settled. Note that the learning rule converges in this case and performs no further weight changes, even if more trials are applied.

With this, we can conclude that the implementation on Loihi was successful. In the next sections I will first extend the algorithm to learn a function while keeping controlled input spike conditions. In a last step, input from the anisotropic network is used and again a function is learned.

Learning a function with re-ReSuMe

The re-ReSuMe algorithm was previously designed to learn spike trains and not continuous functions. Therefore, it is necessary to find a solution for using re-ReSuMe to learn a function. The main idea is to create a pool of neurons and use their summed spikes at each time step as a proxy for a function. For this, two problems need to be solved. First, multiple output neurons need be learned using the re-ReSuMe implementation. Figure 3.15A shows how n input neurons can be used to train m output neurons. This is basically a fully-connected two layer neural network. Second, it is required to translate a function $f(t)$ into a spike pattern for the m output neurons. The distribution of each target spike train should ideally be as homogeneous as possible. In other words, the interspike interval (ISI) should be as large as possible. For this, I followed a stochastic approach to solve this problem and call this approach simply the *rate-to-spike algorithm*. A spike probability distribution vector represents the probability that a specific neuron with index i spikes. The probability distribution depends on the time step t and is defined as

$$P(t) = \Big(p_1(t), ..., p_i(t), ..., p_m(t)\Big)^\intercal. \tag{3.9}$$

The initial distribution at time step $t = 0$ is chosen as a uniform distribution

$$P(0) = \left(\frac{1}{m}, ..., \frac{1}{m}\right)^\intercal. \tag{3.10}$$

Now, we take the first value from the function and sample $f(0)$ times (e.g. $f(0) = 100$) from the distribution $P(0)$ without replacement. We define $S = \{s_1, ..., s_k\}$ as the set of all sampled neuron indices, where $|S| = k = f(0)$. All sampled neurons get a new probability close to zero,

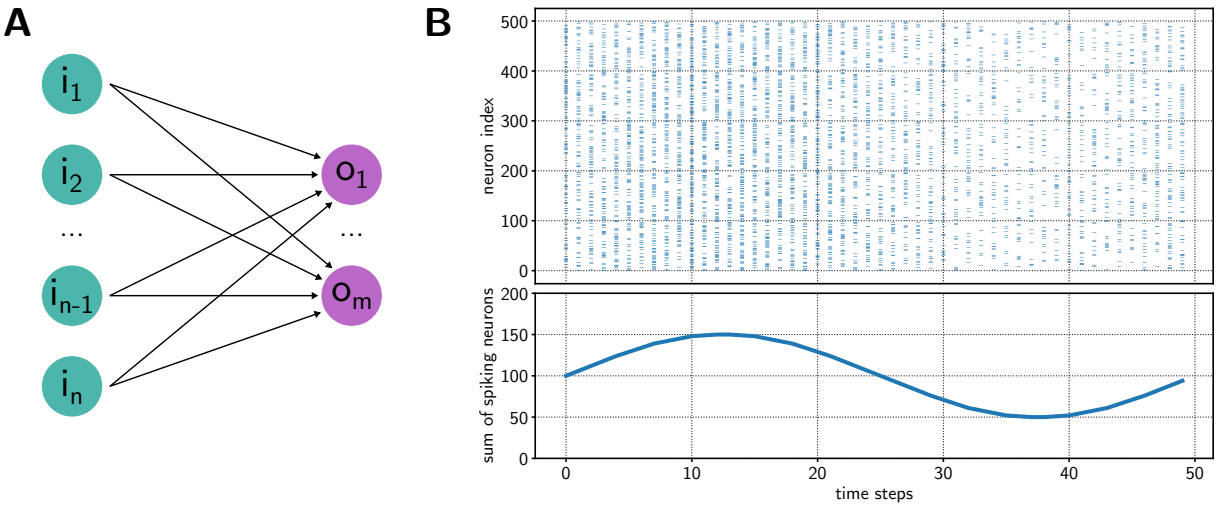


Figure 3.15: Training multiple outputs with re-ReSuMe. **(A)** A fully-connected feed forward network with two layers. The first layer contains n input neurons (green), the second layer m output neurons (purple). Every output neuron adapts its own weights in order to learn a specific spiking pattern. **(B)** A spiking pattern using the rate-to-spike algorithm represents a function. The spike trains (top) are obtained from a continuous function and their sum represent the targeted function with a resolution based on the number of available neurons (bottom).

e.g. $\tilde{p}_{s_1}(1) = ... = \tilde{p}_{s_k}(1) = 0.001$. The other probabilities stay untouched. We denote the new probability vector as $\tilde{P}(1)$. Finally, this vector of probabilities is normalized to obtain a distribution again, i.e.

$$P(1) = \frac{\tilde{P}(1)}{\sum_j^m \tilde{p}_j}. \tag{3.11}$$

From this new distribution we can draw $f(1)$ values and repeat the whole procedure for all $f(t)$ and $P(t)$ until the last available time step of the function is reached. Note that the normalization process increases the probability for neurons that have not fired. Therefore, the longer a neuron has not spiked, the higher is the chance for a spike in the next time step. An example for a simple sine function with 50 time steps is shown in Figure 3.15B. In this case the function is transformed in a spike pattern of 500 neurons. The number of output neurons determines the resolution of the function. Note that a function first needs to be transformed in a spike rate, before the algorithm can be applied.

Learning movement trajectories based on random input

To test the re-ReSuMe implementation with the rate-to-spike algorithm, the first dimension of a movement trajectory from Michaelis et al. (2020) is used. A margin of 5 time steps was always added left and right of the output function with zero values to allow the input spikes to reach a sufficient level of post-synaptic activity, required for the output learning rule. The margin is not shown in the figures.

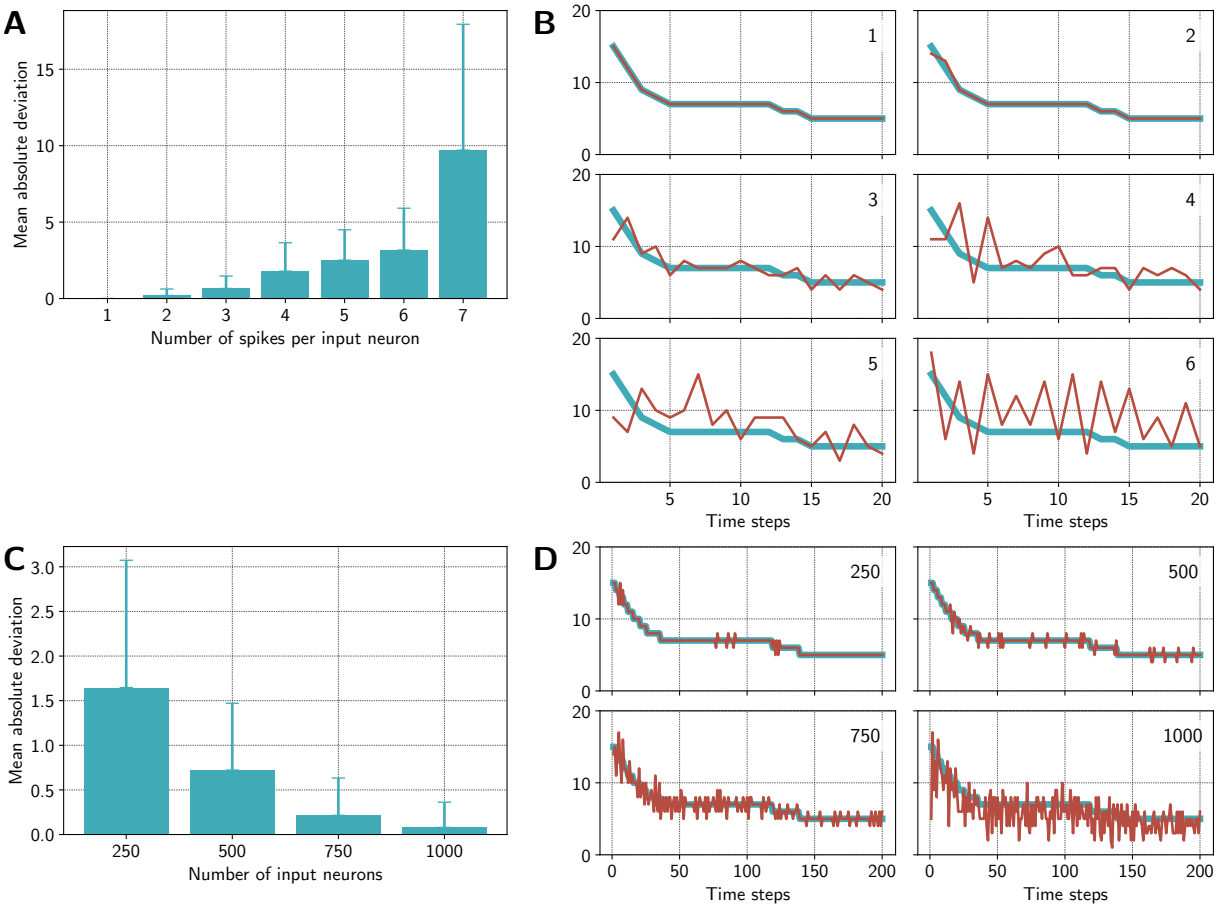


Figure 3.16: re-ReSuMe experiment based on random input. **(A)** Mean absolute deviation between a target function and the estimated function depending on the number of randomly placed spikes per input neuron for a function with a length of 20 time step. The more sparse the input neurons are spiking the better are the results. **(B)** For one random seed the estimated functions, based on the spiking of the output neurons, are compared. The number of spikes per input neuron used for the particular experiment are shown in the top right of each plot. **(C)** Mean absolute deviation between a target function and the estimated function depending on the number of input neurons for a function with a length of 200 time step. The more input neurons are available, the lower is the error. **(D)** For one random seed the estimated functions are compared. Numbers of input neurons used to perform the particular experiment are shown in top right of each plot.

In a first experiment, the number of spikes per input neuron was varied. From Ponulak and Kasiński (2010) we know that ReSuMe is very sensitive to multiple input spikes and best results are expected if every input neuron spikes only once. I expect similar results for the re-ReSuMe approach. For this first experiment, the output function was reduced to 20 time steps. Since the post-synaptic output neurons are independent, the performance does not depend on the resolution of the function and therefore not on the number of used output neurons. 200 input neurons were used as input for each of 200 output neurons. Figure 3.16A shows the mean absolute deviation between the target function and the learned output function. Values are averaged over 5 simulations with different random seed values. We can interpret the error as the average deviation per time step. As expected, the error increases with the number of spikes per input neuron. Surprisingly, for the single spike case, the error is zero in all five simulations. For one random seed, the actual output functions in comparison with the target functions are

shown for 1 to 6 input spikes in Figure 3.16B. It is clear that the error becomes unacceptable high with 4 or more input spikes per neuron. Note that the case for 7 input neurons is not shown, since the trained output function is mostly meaningless. I could therefore show that the results from Ponulak and Kasiński (2010) can not only be reproduced, but also extended for the output function. Up to 3 input input spikes per neuron lead to acceptable results. Considering the margin and therefore a total length of 30 time steps in this first experiment, one spike per 10 time steps can be seen as a limit for learning tolerable output functions.

A second experiment tests the influence of the number of input neurons. I assume that the more neurons are used, the more information is available for the learning mechanism. For this experiment, the output function was trained for a much longer period of 200 time steps, but the resolution was kept as before. The number of spikes per input neuron was fixed at 1 and the experiment was again applied 5 times. Indeed, Figure 3.16C shows that the more neurons are used as input for every output neuron, the lower is the error. With 1000 input neurons, a function over 200 time steps can be represented quite well, as shown in Figure 3.16D. This is a promising result, given the fact, that the total length of the trajectory in Section 3.3.1 had the same length. On the other hand, it makes clear, that a good result can only be expected when the input neurons spike sparsely.

Learning movement trajectories based on input from the anisotropic network

In a final step, input spikes from the anisotropic network were used. Since the pooling layer has only 72 neurons, from which some neurons stay silent even for the whole simulation time, I was not expecting any reasonable results, given the findings from the random input experiment. Also larger pooling layers with 200 or 288 neurons are possibly still too small. A test was therefore performed based on the whole excitatory neuron pool from the anisotropic network. The output function was again 20 time steps, plus the 5 time steps padding before and after the function, resulting in a total of 30 time steps. The resolution was kept at 200 output neurons.

From the spiking activity of the excitatory neurons of the anisotropic network from Michaelis et al. (2020) five trials were used for this experiment. From all 3600 spike trains of one trial, I filtered out those neurons which spike only a specific number of times. For example, in the first simulation I chose those spike trains from the 5 trials which contain only one spike during the 30 time steps. In the second simulation I chose all neurons which spike either 1 or 2 times. In the third simulation, neurons that spike 1, 2 or 3 time are chosen and so on. With this, not only the number of spikes increase, but also the number of available neurons increase. Figure 3.17B shows that the number of available input neurons linearly increases the more spikes are allowed per neuron. This leads to a "U" shape of the performance, as shown in Figure 3.17A. In the beginning, the spiking conditions are optimal with one spike per neuron, but the number of available neurons fulfilling this condition is too low for a reasonable representation of the function. The more spikes per neuron are allowed, the more neurons are available as input,

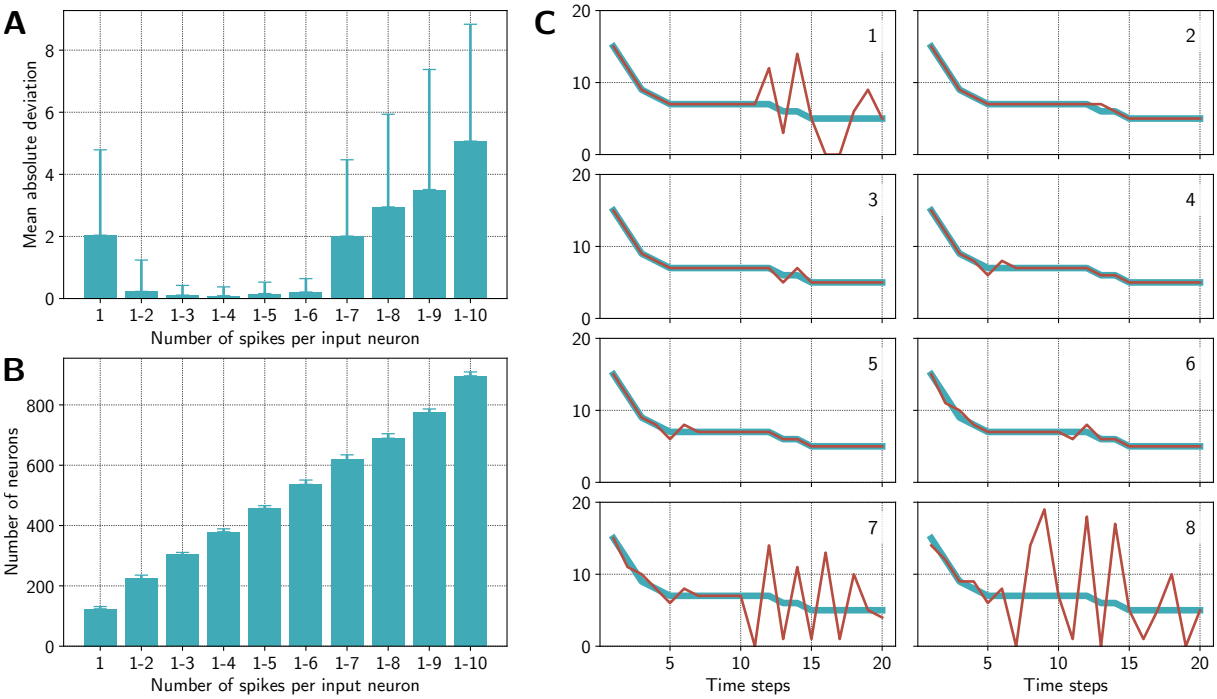


Figure 3.17: re-ReSuMe based on input from the anisotropic network (Michaelis, Lehr, & Tetzlaff, 2020). **(A)** Mean absolute deviation between a target function and the estimated function depending on the number of spikes per input neuron for a function with a length of 20 time step. **(B)** The number of available input neurons increases linearly with more spikes allowed for a spike train of a neuron within the given time period. **(C)** For one data trial of the anisotropic network simulation the estimated functions are compared. Numbers in the top right of each plot refer to the number of spikes allowed per neuron. The number indicates the end of the range, e.g. 5 represents $1 - 5$.

which reduces the error to a certain degree. When 7 or more spikes are allowed per input neuron, the error increases again since the spiking behavior becomes too dense and the number of additional available input neurons is not sufficient to compensate this adverse input condition. Figure 3.17C shows the learned output functions in comparison with the target function for the first 8 cases for one trial. Interestingly, re-ReSuMe is quite robust and allows a range of possible combinations. Nevertheless, a well performing output learning based on all excitatory neurons of the anisotropic network is not possible.

Conclusion

In summary, it becomes clear that input from the anisotropic network can not directly be applied to the re-ReSuMe learning approach. However, it was shown that the learning rule has promising properties and can represent functions quite well, provided the necessary conditions. Since the experiments unveiled these conditions, we can hypothesize possible improvements. First, the pooling layer could be used if larger simulations on Loihi become possible, probably in the near future. Note that the original implementation of the anisotropic network was implemented with a total of 18.000 neurons (Spreizer et al., 2019), whereas our implementation was restricted to a total of 4.500 (Michaelis et al., 2020). With a larger anisotropic network, also the pooling layer would scale up significantly, which could improve the results. Nevertheless, also in the

case of a much larger network, the anisotropic network would still need to spike sparsely. This could possibly be solved by plastic synapses between the anisotropic network and the pooling layer implementing a homeostatic synaptic plasticity mechanism, perhaps similar to the synaptic mechanism implemented in the cell assembly learning task in Section 5.2. This could help to adjust the weights to provide sparsely firing spike trains with high ISI, which would then possibly enable re-ReSuMe to represent output functions sufficiently and may even allow an application in a generalisation task.

3.3.4 Summary

This section described several solutions for encoding movement execution. A network with an anisotropic connectivity structure was identified as an effective approach to solve the variability-stability problem. A simulation with noisy inputs revealed a remarkable robustness, compared to a randomly connected network, that was further improved by a lognormal weight distribution and STDP. The reward-based re-ReSuMe learning rule, based on local mechanisms, enabled the representation of movement trajectories using random input spikes. I described how these two approaches could possibly be integrated using the pooling layer based on a larger anisotropic network, where the pooling layer weights could be adapted according to an homeostatic mechanism. Taken together, these results fit well into the HMSE model and provide a complete solution for its execution layer. I suggest these concrete implementations to foster the understanding of movement encoding and hope that they provide a valuable foundation for further research. Additional implications are addressed in detail in the following discussion.

Chapter 4

Discussion

Movement is a highly complex task and crucial for all animals. As outlined in the introduction, the motor system comprises a wide range of brain areas (Krakauer et al., 2019) controlling numerous muscles. Despite the important role of movement, possibly even for cognitive processes (Gallese & Lakoff, 2005; Gentsch et al., 2016), a comprehensive theory about the details and relations of the underlying systems is still in an early stage (Diedrichsen & Kornysheva, 2015). This thesis aimed to develop a theory about selection and execution of movements, described by the hierarchical motor selection and execution (HMSE) model and suggests concrete biologically plausible mechanisms for the underlying execution processes based on spiking neural networks (SNN). The developed theoretical and computational models in turn elicit questions for experimental research.

4.1 Summary of the results and open issues

The HMSE model, as the theoretical framework for the here presented work, includes four layers. The highest level represents long sequences of actions accompanied by a second layer that stores transitions between motor actions. These two upper layers *select* specific "atomic" motor tasks. The *execution* is based on the two bottom layers. The first stores robust spatio-temporal neural activity that provides the basis for actual movements. Single motor actions are then stored in output weights which provide a reliable spiking pattern in the final output neurons, described by the second execution layer. The hierarchical structure is supported by a network that gates and initializes movements and feedback mechanisms to enable the representation of new movements or adapt existing behavior.

I suggested two concrete mechanisms for the execution layers of the HMSE model. First, an anisotropic synaptic connectivity structure provides the required spatio-temporal spiking activity which is robust to noise and provides sufficient variability to learn multidimensional movement trajectories based on its activity in real-time. Second, a reward-based learning rule, called reward-based remote supervised method (re-ReSuMe), which allows the actual representation

of a trajectory within weights between input and output neurons. The input neurons could be represented by the anisotropic network, the output neurons comprise a pool of neurons for each dimension of the output function. The rate of the output neurons finally represents the targeted function. In addition to these two results, a learning rule allowing self-organized cell assembly formation on Loihi was developed, described in the appendix. This additional result is a crucial part for further developments of the HMSE model, especially important for the selection layers and the gating network that require neuron pools acting like cell assemblies representing states of an ongoing movement. Further research may allow using cell assemblies storing the state of a movement to coordinate smooth transitions between motor actions and prevent interference between sequences.

The most significant finding within this work is the ability of the anisotropic network to solve the variability-stability problem (Pehlevan et al., 2018). We could demonstrate that the spiking behavior of a network with an inhomogeneous local connectivity structure provides enough variability to robustly learn a three-dimensional movement trajectory (Michaelis et al., 2020). This network type performed significantly better compared to a traditional randomly connected network, even though the randomly connected network used biologically plausible assumptions as well. Moreover, our approach with the anisotropic network not only suggests a mechanism for the motor system in the brain, it may also provide a building block for future algorithms, especially for robotic applications based on neuromorphic hardware. A deeper analysis of the anisotropic network revealed conditions for a remarkable improvement of the network's ability to provide robust spatio-temporal streams of activity which allows nearly perfect prediction results under noise, both in the input and within the networks' neurons. We could show that this improvement is mainly based on choosing an appropriate locality for the connectivity, initializing synaptic weights with a lognormal distribution, common for biologically plausible randomly connected networks (Fukai et al., 2014; Lefort et al., 2009), as well as the application of spike-timing-dependent plasticity (STDP). The motifs of connections between neurons within the anisotropic network even match motifs from experimental findings (Song et al., 2005). I want to highlight that this match is not due to an exhaustive fine-tuning of several different parameters, but is determined by a single parameter, the locality σ. This indicates that the approach of the anisotropic network is by design already quite close to natural neural connection patterns.

Recently a similar approach of initializing networks with inhomogeneous connectivities was published by Spalla et al. (2021) and shows apparent similarities with the approach of the anisotropic network (Spreizer et al., 2019). This concurrent development of similar ideas indicates that possibly a whole field of research based on inhomogeneous connections could emerge that is beyond traditional state-based cell assemblies and provides robust spatio-temporal memories.

A further important finding concerns the transformation of high-dimensional spiking activity within a reservoir to a low-dimensional movement trajectory or – more generally speaking – an output function, based on local-only mechanisms. For this purpose, I proposed the so-called re-ReSuMe learning rule. It is a modified plasticity mechanism based on the ReSuMe rule (Ponulak, 2006; Ponulak & Kasiński, 2010). It is based on a neat learning mechanism that adapts the synaptic weights between several input neurons and an output neuron in order to

shift the spike position of the output neuron to match a targeted spike train. The adaptation is performed by a reward signal indicating the correct spike positions. The rule is supervised, but purely local with respect to a synapse which allows an implementation on the neuromorphic hardware Loihi. In order to allow re-ReSuMe to represent functions and not only single spike trains, the spike rate of a pool of neurons was used as a proxy for the function value. For this, I developed an algorithm that translates an arbitrary function into a spiking pattern, called rate-to-spike algorithm, described in Section 3.3.3

An analysis of the learning rule revealed that it is sensitive to the configuration of the input spike trains. The plasticity mechanism requires large interspike intervals (ISIs) and therefore relatively low firing rates of the input neurons, which is unfortunately not supported by the anisotropic network. The network has too high firing rates at the bump locations, but is silent at all other locations – and if it provides activity at other locations, this can be considered as noise. However, the re-ReSuMe rule performs well if the input spikes are randomly generated with a sparse spike pattern. If the activity of the anisotropic network could be transformed or modified to provide a sparser spiking behavior, considerable output learning results are expected. A promising approach could be to extend the pooling layer of the architecture introduced in Michaelis et al. (2020). Originally, we used this layer to reduce the readout and therefore the output load of the chip and showed that it has additional regularization abilities. The synaptic connections to the pooling layer are grouped and keep the spatial structure of the activity bumps but, however, are static. A homeostatic mechanism could be used for these synaptic weights that would enable a sparse spiking behavior of the pooling layer neurons which would maintain the spacial structure and stability. Such a homeostatic mechanism could be realized using synaptic scaling (Abbott & Nelson, 2000; Tetzlaff, Kolodziejski, Timme, & Wörgötter, 2011; Turrigiano & Nelson, 2004) as used for the cell-assembly formation described in Section 5.2, short-term plasticity (Hennig, 2013; Zucker & Regehr, 2002) or intrinsic plasticity (Debanne, Inglebert, & Russier, 2019; Desai, Rutherford, & Turrigiano, 1999; W. Zhang & Linden, 2003).

An improved re-ReSuMe rule or an optimized spiking behavior of the underlying network would not only allow to apply the generalization task to the output learning mechanism, but also to incorporate re-ReSuMe into the architecture of the robotic algorithm (Michaelis et al., 2020). Note that with the 2-chip Loihi system available to us, the Kapoho Bay, an integrated simulation would not be feasible due to restrictions in the amount of available reward channels, neurons and synaptic connections. However, it would may be possible on the next generation of the Loihi chip (Davies et al., 2018) or alternative systems like SpiNNaker (Stephen Furber & Brown, 2009; S. B. Furber et al., 2014).

While I am relatively confident that the re-ReSuMe learning rule will work well together with the anisotropic network, provided some further research, we should not overlook that the rule requires exact and perfectly prepared spike trains as a prototype to learn from, which is not biologically plausible or at least relocates a part of the task to a different brain area preparing the required spiking behavior. Unfortunately – to the best of my knowledge – no other approach for solving the output learning problem on a local basis for spiking neural networks based on input spikes from a reservoir network exists. For a more realistic implementation it would be

required to learn the trajectory based on global feedback. Gilra and Gerstner (2017) provide a solution that depends on feedback, but the network structure provides a monolithic solution and, therefore, an output learning mechanism can not directly be derived. However, since it is solely based on local mechanisms and follows in principle a reservoir computing approach with spiking units, it provides building blocks that can be useful for the development of future algorithms. Merging both ideas could lead to an algorithm where the output neurons are not trained individually and independently, but rather with a global synaptic weight adaptation based on the re-ReSuMe rule to increase or decrease the spike rate of the whole output neuron pool together. This would require to find a proxy for the firing rate of the pool which needs to be compared with the targeted function. Either way, a purely local and high-performing output learning mechanism is still an open problem and requires more research. First promising steps are taken, also as part of this thesis, which gives the impression that solutions are within reach.

A final result of the thesis is the development of several frameworks and tools providing efficient simulations with Loihi. I presented the `PeleNet` reservoir computing framework for Loihi, the `Brian2Loihi` emulator and the fitting tool for translating existing neuron models to Loihi. These frameworks not only provide open source software for other research groups working with Loihi, they also give insights into existing difficulties and limitations of the neuromormphic architectures in general and Loihi in particular. An obvious challenge is the unavailability of high-level software frameworks and a missing online community, due to the early stage of these hardware systems. It is hard to transfer neural network implementations between simulators since APIs are missing. If errors occur and bugs are identified, no public community is available to discuss and share these insights. Despite huge effort of the inventors and developers of such hardware systems, the communities are still very small and specific. But issues also exist on a technical level. A major drawback is the input/output bottleneck, i.e. monitoring spikes, voltages, currents and other variables is an expensive operation and rather limited which makes the development of new algorithms on these hardware systems difficult. Additional issues, like a limited bus and memory does not only constrain large networks, but also requires the researcher to be aware of deeper hardware specifications when implementing models on the hardware, which could probably be improved by advanced high-level frameworks, as mentioned above. All in all, we need to be aware that, beside impressive progress within the last years (see e.g. Fan & Markram, 2019), the development of neuromorphic hardware is still in an early development stage. An essential contribution of this work is therefore that most neuroscientific experiments within this thesis were successfully simulated on-chip, demonstrating that neuroscientific research is possible even on the first generation neuromorphic hardware Loihi. It is important to mention that beside the demanding and challenging task of doing research on-chip, it has two outstanding advantages: the experimenter is forced to think more in a biologically plausible way and it is easier to transform findings from research into technical applications.

4.2 Perspectives and future research

Reflecting about movement and neuromorphic hardware as well as the implemented solutions resulted in several new perspectives. This part includes some major implications that may provide inspirations and concrete experimental perspectives for future research.

Two fundamental memory types

As a result of working with the anisotropic network for storing spatio-temporal information, described in Section 3.3.1 and 3.3.2, and the representation of states using cell assemblies, introduced in Section 5.2, we can possibly define two fundamental types of memory. The on-off behavior of the cell assemblies was early and repeatedly criticized for its stateful behavior (Milner, 1957; Spalla et al., 2021). While one would intuitively identify the anisotropic network as the superior approach, as it provides these features inherently, in the context of the HMSE model, this general assumption is not necessarily justified. The ability of cell assemblies to provide a controllable state indicator is still a valuable property for the HMSE model and beyond. Many processes inevitably need to store an explicit state to control ongoing processes. This behavior is not satisfactorily supported by the anisotropic network. Streams of activity can be started and stopped, but not on an arbitrary timescale due to the transient character of the activity within the anisotropic network. In addition, while a start "command" activates a specific region in the network, similar to a cell assembly, a stop command would imply to inhibit the whole network since we have no information about the current position of the moving bump of spiking activity. Within a cell assembly we can inhibit the same neuron cluster that was activated before, probably inhibiting only a share of the previously activated neuron group would even suffice. Therefore, the stream-like propagation of activity within the anisotropic network is in general not appropriate for control signals.

On one hand, the state-like and explicitly controllable property of a stationary activity within cell assemblies can be identified as a specific memory indicating states of ongoing processes within neural networks. In particular, working memory studies show that keeping a state over a specific time period is crucial for memorizing information and is closely related to attention (Axmacher et al., 2007; Ruchkin, Grafman, Cameron, & Berndt, 2003). This sustained activity in the context of working memory was shown in several different brain areas (Malecki, Stallforth, Heipertz, Lavie, & Duzel, 2009). Moreover, sustained state-like activity is also known from movement onsets (Schultz & Romo, 1992). A preparatory activity within the cortex possibly indicates the initialization of a movement and has no direct functional meaning (Churchland, Cunningham, Kaufman, Ryu, & Shenoy, 2010), supporting state-like dynamics similar to cell assemblies. However, brain activity prior to movement onset is complex and several different types of preparatory activity need to be distinguished (Svoboda & Li, 2018). On the other hand, the ansiotropic network with its stream-like spatio-temporal activity, similar to continuous attractors (Spalla et al., 2021), defines another type of memory that provides a basis for

trajectories of any kind. Several experimental studies investigating working memory show that transient activity can be distinguished from sustained activity (Courtney, Ungerleider, Keil, & Haxby, 1997; Yantis et al., 2002). Again, this distinction is also known from movement-related experiments (Salari, Freudenburg, Vansteensel, & Ramsey, 2018). Certainly this implicitly assumes that a sustained activity matches a state-like activity and a transient activity is based on a spatio-temporal activity, which is so far not clearly shown.

The assumption of these two hypothesized neural processes may form a symbiotic relationship. While the *state process* would avoid interference and provides control and order, probably more as a top-down process controlled by higher regions, the *spatio-temporal process* would form the basis of the actual operation, like a movement or a memory. Imagine a runner in a competition waiting for the start signal. A state process would attentively suppress any movement until the sensory system recognizes the start signal. The inhibition of the movement is stopped and the running behavior can be performed, based on a spatio-temporal process acting as basis for representing the movement itself. The same processes could also underlie cognitive tasks. We can consider the method of loci, a famous and powerful approach for memorization (Qureshi, Rizvi, Syed, Shahid, & Manzoor, 2014; Verhaeghen & Marcoen, 1996), based on visualizing a "walking" process and placing pieces of information along the way. Due to its movement-related character it is not too speculative to assume a spatio-temporal process underlying this cognitive operation. At the same time, either during learning or during retention, a state process needs to keep attention by suppressing other cognitive processes, as described above. Both examples illustrate that a state and a spatio-temporal neural process possibly form basic types of computation and interact mutually.

A theoretical model for large-scale simulations

In the last years and decades computational power has increased significantly. Gradually, larger neural network simulations for SNNs are supported. However, disruptive innovations in hardware design and manufacturing processes do not guarantee the availability of reasonable algorithms. A neural network simulation of millions of neurons is impressive, but meaningless if it does not produce any useful result. As described in more detail in Section 2.3.1, algorithms for large-scale SNN simluations are urgently needed (Steve Furber, 2016; Nguyen et al., 2021; Schuman et al., 2017; Zhu et al., 2020).

The HMSE model presented within this work, derived from experimental findings (reviewed by Krakauer et al., 2019) and theories integrating previous results (reviewed by Diedrichsen & Kornysheva, 2015), provides a theoretical framework that connects several networks in a meaningful way. It therefore provides a basis for exploiting novel computing systems, either based on traditional or neuromorphic hardware architectures. If the size of the anisotropic network's topology is increased to e.g. 1000×1000 to approach more realistic network sizes and possibly improve performances, e.g. for the re-ReSuMe learning rule, the anisotropic network alone would reach a million neurons and billions of synapses. Including pooling layers and the output learning mechanism requires hundreds of reward channels and again numerous neurons

and synapses. This calculation so far does not contain any estimation about the size of the selection layers of the gating network, nor does it consider connections between the layers. This shows apparently that traditional computer architectures will soon reach limits, especially when considering learning rules that would slow down calculations on traditional von Neumann hardware drastically and would require a lot of memory and energy. However, first large-scale neuromorphic systems, like e.g. the SpiNNaker project (Stephen Furber & Brown, 2009; S. B. Furber et al., 2014), are probably capable of simulating the whole HMSE model already today. Soon more neuromophic hardware systems will be capable to simulate such huge and computationally expensive network simulations, like the second generation of Loihi or future versions of BrainScaleS (Friedmann et al., 2017; Schemmel et al., 2010). The local architecture of neuromophic hardware is in principle ideal to simulate large SNNs with a rich variety of learning rules and the HMSE model provides a theory that has to be tested with large-scale simulations.

However, it is important to bear in mind that only the execution layer of the HMSE model is underpinned with concrete SNN implementations in this work. And even in the execution layer open issues exist. It is for example desirable to improve the output learning mechanism. That is to say, before components are stacked together in large-scale simulations, it is important to make sure that all components perform reasonably well by themselves.

Next generation neuromorphic hardware

The second generation of the Loihi chip has many promising features, as describes in Section 2.3.3, to improve the performance and ability of the provided results. The anisotropic network requires numerous neurons and synapses, due to its topology and recurrent structure. Loihi2 provides many more neurons on one hand and a compressed axon routing on the other hand, which probably increases the number of available synapses drastically. Within this work, the number of synapses was a clear bottleneck, which forced us to reduce the size of the anisotropic network to its absolute minimum, a quarter of its original size. With more neurons and synapses at hand, simulations of the anisotropic network could perform considerably better on neuromorphic hardware, especially when plasticity is used. In addition, robotic applications based on our findings (especially regarding Michaelis et al., 2020) could become serious candidates for robotic control. Further, the proposed solution for the SSSP problem, presented in Section 5.1, could benefit from graded spikes. This feature would allow the algorithm to run much faster. With this, the SSSP algorithm could possibly become a competitor of conventional algorithms.

4.3 Conclusion

Within this thesis a theoretical model about how selection and execution of movements may be stored in the brain was developed, based on existing scientific findings. A concrete SNN im-

plementation of the execution layer was proposed and developed directly on the neuromorphic research chip Loihi. It was demonstrated that the anisotropic network, as a promising candidate for providing spatio-temporal spiking activity in a timescale of hundreds of milliseconds is capable to solve the variability-stability trade-off and provides in particular a promising basis for robust movement representation in real-time. A reward-driven learning rule was applied, showing that movement trajectories can be learned locally and on-chip. Both findings do not only contribute to the understanding of motor related functionalities of the brain, but moreover enable applications for robotic control. The research was accompanied by the development of several open source frameworks and tools for the neuromorphic hardware Loihi, including the `PeleNet` framework for reservoir computing and the `Brian2Loihi` emulator. In addition, supplementary results provide a novel on-chip solution to the SSSP problem and propose a balanced learning rule for the self-organized formation of cell assemblies. A highly biologically plausible computational approach demonstrated how movement could be encoded in the brain and despite challenges of the first generation of Loihi, the simulations were successfully performed on neuromorphic hardware.

Chapter 5

Appendix

5.1 Single source shortest path problem on Loihi

Contribution statement

This work was performed together with Elena Offenberg and Andrew B. Lehr. Andrew B. Lehr and me supervised the project, Elena Offenberg implemented the solution within her bachelor thesis. The figures within this sections are based on figures from Elena Offenberg, but redesigned by me. My contribution to this section was about 15%.

5.1.1 Introduction

The SSSP problem

The SSSP problem describes the problem of finding a path between a source element and all other elements. Between those source and target elements, several other interconnected elements exist. Graphs can be used to describe this problem mathematically (Magzhan & Jani, 2013). *Nodes* (also denoted as *vertices*) represent elements, including the source and target elements. *Edges* describe the existing directed connections between all elements. In addition, a cost value is assigned to every edge. A common example would be to find a route from location A to location B on a map, where the edges are streets and the crossroads are represented by nodes. Aspects like speed limits, the condition of the road and roadworks all influence the cost of a road section. The algorithm finally shall compute the shortest path, constrained to all conditions in-between, defined by edges and their costs. Formally, we define a directed graph as an ordered pair $G = (N, E)$, containing a set of nodes N and a set of edges E. A particular edge is an

ordered pair of nodes, i.e. $(n_x, n_y) \in E$ where $n_x, n_y \in N$. The edge is directed and points from node n_x to node n_y. In addition, every edge is weighted by the weight function $w : E \to \mathbb{R}$ which then results in a *weighted directed graph*. We denote the weight as *cost* of the edge.

Conventional algorithms

Many algorithms for solving shortest path problems in general and the SSSP problem in particular already exist (Kumawat, Dudeja, & Kumar, 2021; Magzhan & Jani, 2013). Four conventional algorithms are Dijkstra's Algorithm (Dijkstra, 1959), the Floyd-Warshall Algorithm (Floyd, 1962; Roy, 1959; Warshall, 1962), the Bellman-Ford Algorithm (Bellman, 1954; Ford, 1956) and genetic algorithms, e.g. developed by Karas and Atila (2011) or S. Zhang and Zhang (2018). Two further types of path planning algorithms include sampling approaches and the usage of artificial neural networks, both approaches are introduced by Kulvicius, Herzog, Tamosiunaite, and Worgotter (2021).

Algorithms based on spiking neural networks

Another class of possible solutions to the SSSP problem is based on spiking neural networks (SNN). The main advantage of SNN-based algorithms is the opportunity to run them efficiently on neuromorphic hardware, at least in case they are based on local-only mechanisms. This possibly allows a fast and, first and foremost, a highly energy-efficient computation.

One such algorithm exists and was already suggested as early as 1991 (Aibara, Mitsui, & Ae, 1991). Within this approach, nodes are simply represented by neurons and edges match synapses. Edge costs are coded in delays at the synapses. This algorithm was recently applied to the neuromophic hardware Loihi (Davies et al., 2021).

Two main problems with this algorithm exist. First, since the costs are encoded in the synaptic delays of the neural network, the total simulation time highly depends on the resolution of the costs. If the costs require a high resolution of potentially hundreds or thousands of different values, the delay exponentially rises and slows down the execution. In addition, Loihi provides only a maximum of 62 delay values, which limits the resolution by hardware. The second problem affects the refractory period. The algorithm can be designed to store the shortest path within the synaptic weights. For this, STDP can be used (Ponulak & Hopfield, 2013). If the pre-synaptic neuron and the post-synaptic neuron fire in a row, the weight between them is increased. To assure that only the shortest path is increased in weight, the refractory period of the neurons is set to a very high value. On the currently available Loihi chip, the maximum refractory period is 64, which limits the size of applicable graphs to either small graphs or tasks with a small resolution for the costs.

5.1.2 The add-and-minimize algorithm

In 2020, a novel approach was proposed by Aimone et al. (2020). The approach was suggested as a theoretical idea and was not actually implemented. It mainly consists of a minimization and a summation process. We developed all necessary components and integrated them in a functioning SSSP algorithm. We name the finally working approach the add-and-minimize (AM) algorithm, since it mainly consists of two consecutive steps: adding cost values at edges and choosing the minimum of incoming costs at nodes. Both steps are realized as a neural component, practically two small neural networks. First, the functionality of both components is introduced. Second, it is explained how these components are combined to a larger network for a path finding task.

The addition component

The first component is the *addition* component. A pre-defined cost value for the edge is added to the current sum of cost values at a given branch. We can consider an example with two inputs $\mathrm{I}_0 = 101$ and $\mathrm{I}_1 = 001$. The output should then be $\mathrm{O} = 110$. A simple neural component for adding two binary values is shown in Figure 5.1. Note that the most significant bit has the index 2 and the least significant bit has index 0. The input bits are represented by neurons, resulting in 3 neurons per input. Furthermore, 3 output neurons represent the sum of the inputs. 4 carry neurons, indicated by C, signal if a carry has to be kept for the addition of the input bits. The first carry bit is fixed at 0.

An AND_I neuron gets input from both input neurons for the current bit. If both neurons are active, the AND_I neuron gets activated as well. A ONE neuron gets inhibitory input from the AND_I neuron. In addition, excitatory connections come from the input neurons as well. All in all, the ONE neuron gets activated when exactly only one of the input neurons is active. If both neurons are active or if an inhibitory signal comes from the AND_I neuron, the ONE neuron stays inactive. A further neuron is denoted as AND_C and gets input from the ONE neuron and the previous carry value. With this, the output neuron is activated if only one bit value (activation of one neuron or the carry) or three bit values (both inputs and the carry) are active. The next carry value is activated if two or more bit values are active. The output neurons finally provide the sum of the input values, which can be used in the next component, the minimum component.

The minimum component

The other component of the AM algorithm is the *minimum* component. Several cost values are given as binary input. For example, consider two input values $I_0 = 101$ and $I_1 = 100$. The minimum of both inputs is $O = 100$. The neural component for obtaining the minimum of the two inputs is shown in Figure 5.2. Again, each input comprises 3 neurons for each digit. In

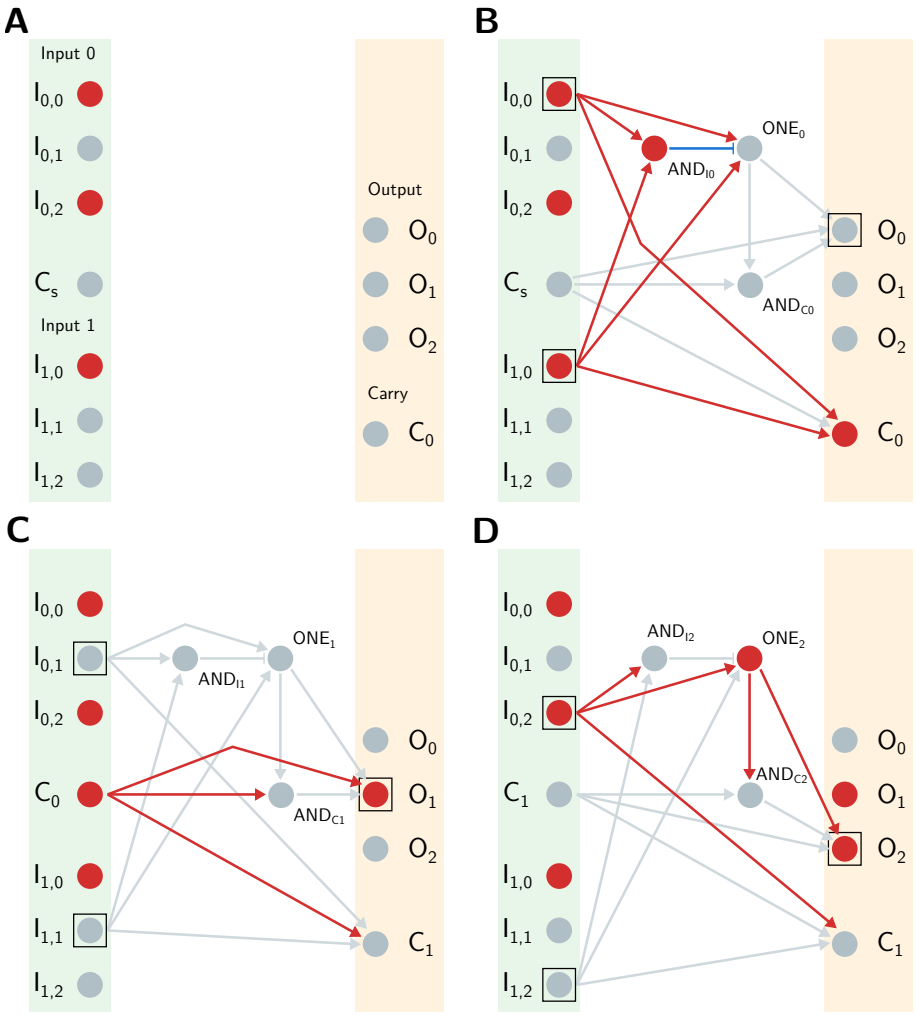


Figure 5.1: An example of the addition component with the binary numbers 101 and 001 as inputs. **(A)** The basic setup with input neurons on the left side (green background) and output neurons on the right side (yellow background). **(B)** The addition process of the least significant bit, **(C)** for the second and **(D)** for the most significant bit. More details are given in the text.

addition, 4 neurons indicate if a particular input bit is still a candidate. Finally, 3 neurons form the output that contains the final minimum value. The network iterates through the input digits sequentially and therefore comprises 3 iterations.

A ZER neuron is excited by the candidate neuron CD and inhibited by the input neuron of the current bit. Therefore, the ZER neuron is only active if the particular input bit was a candidate in the previous step and the current input bit neuron is not active. An OR neuron gets input from all ZER neurons of the current iteration and indicates if one of the ZER neurons is active. The OR neuron is a prerequisite to reject an input as a candidate. In addition, an INH neuron gets inhibitory input from the ZER neuron and excitatory input from the OR neuron. If this INH neuron is active, it finally inhibits the candidate. While this structure finds out which input is still a candidate, it is necessary to have the actual value of the minimum input value directly available for further computations within the graph. This is performed by AND neurons which get input from the input neuron and the candidate neuron of the current bit. If both are active, a corresponding output neuron O is activated for each bit. The O neurons finally contains the

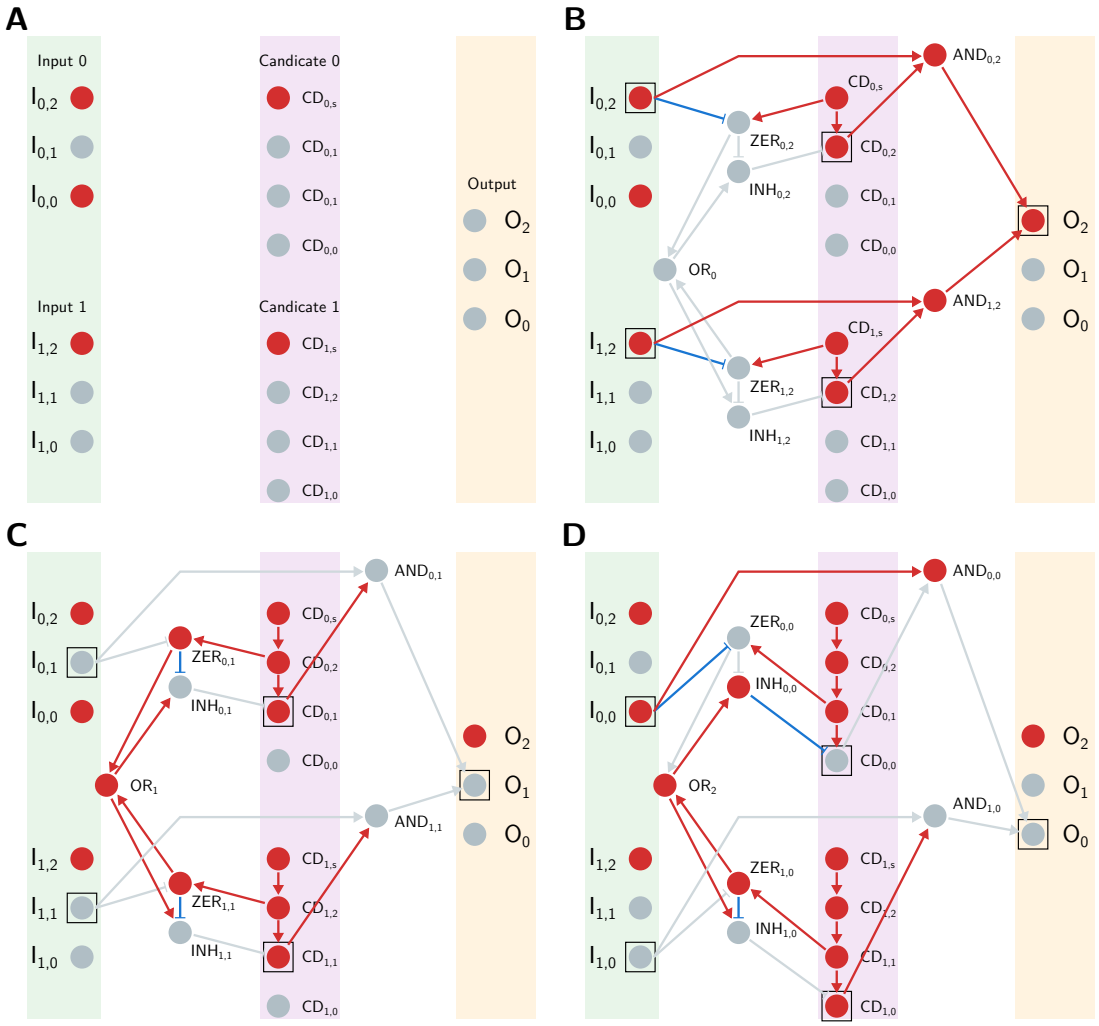


Figure 5.2: An example of the minimum component with the binary numbers 101 and 100 as inputs. **(A)** The basic setup with input neurons on the left side (green background), candidate indicator neurons in the center (purple background) and output neurons on the right side (yellow background). **(B)** The minimum comparison of the most significant bit, **(C)** for the second and **(D)** for the least significant bit. More details are given in the text.

value of the minimum input, which can either be used as input for the addition component or actually forms the final cost result.

Costs and delays

In a final step, we need to combine the components to a complete network. The addition component replaces the edge of the graph, the minimum component replaces the node. Both is depicted in Figure 5.3. Two further extensions are necessary to obtain a functional path finding network. First, the costs are given from outside. Note that for this only one active neuron is necessary. This neuron is then connected to those input bit neurons of the add component which shall be a 1. This external cost neuron is activated periodically. Due to the periodic input, it is necessary to avoid an interference of signals between the neurons within the components. Therefore, every neuron need to have a specific decay constant for the membrane voltage to

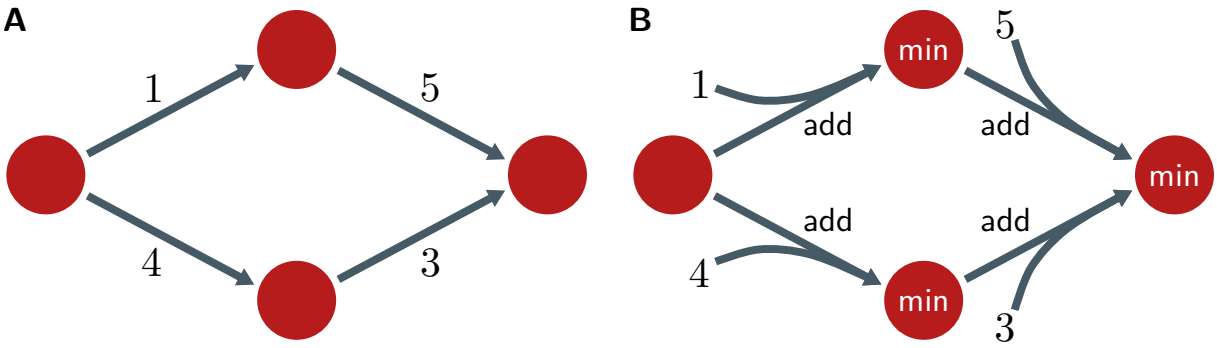


Figure 5.3: Translation of the shortest path graph to the add-and-minimize (AM) network. Nodes are represented by red dots, grey arrows indicate edges. **(A)** An example of a single source shortest path (SSSP) problem. The left node is the source node and all other nodes are potential target nodes. Numbers indicate the costs of every edge. The upper path has a summed cost of 6 and the lower path a summed cost of 7. Therefore the upper path is the shortest path and the minimum overall cost is 6. **(B)** Translation of the graph into the AM network. Costs are added at the edges using the addition (add) component. The nodes are replaced by the minimum (min) components.

assure that the neuron is reset when the next cycle starts.

5.1.3 Discussion

The AM algorithm is a novel path finding approach which has the ability to exploit neuromorphic hardware. It can potentially not only increase the speed of the SSSP problem, but can possibly also save a lot of energy while performing the task, compared to traditional algorithms running on conventional CPU architectures. Based on a principle algorithm description from Aimone et al. (2020), we developed all necessary components and could show that they can successfully be integrated into a functioning algorithm. The code for the AM algorithm implementation, based on the `Brian2Loihi` emulator, is available on GitHub[1].

Nevertheless some limitations still exist. The number of neurons required is obviously much higher than in the delay algorithm. While the delay algorithm consumes ideally one neuron per node and one synapses per edge, the AM algorithm needs several more neurons for both, nodes and edges. However, we believe that the number of neurons and synapses is not a severe limiting factor. Existing neuromorphic systems already provide millions of neurons and many more will be supported by new neuromorphic hardware systems, like Loihi2 (see Section 2.3.3). In addition, the AM algorithm requires several more time steps to propagate the information through the graph. Where the delay in the delay algorithm is the limiting factor for the algorithm's speed, in the AM algorithm the size of the addition and minimum components increase the number of time steps. Note that the resolution of the costs is the key factor in both algorithms. For the delay algorithm, the needed time steps grow exponentially when the resolution is increased. The AM algorithm has a linear increase in time steps. There is a break-even point where the AM

[1] https://github.com/elena-off/sssp-loihiemulator

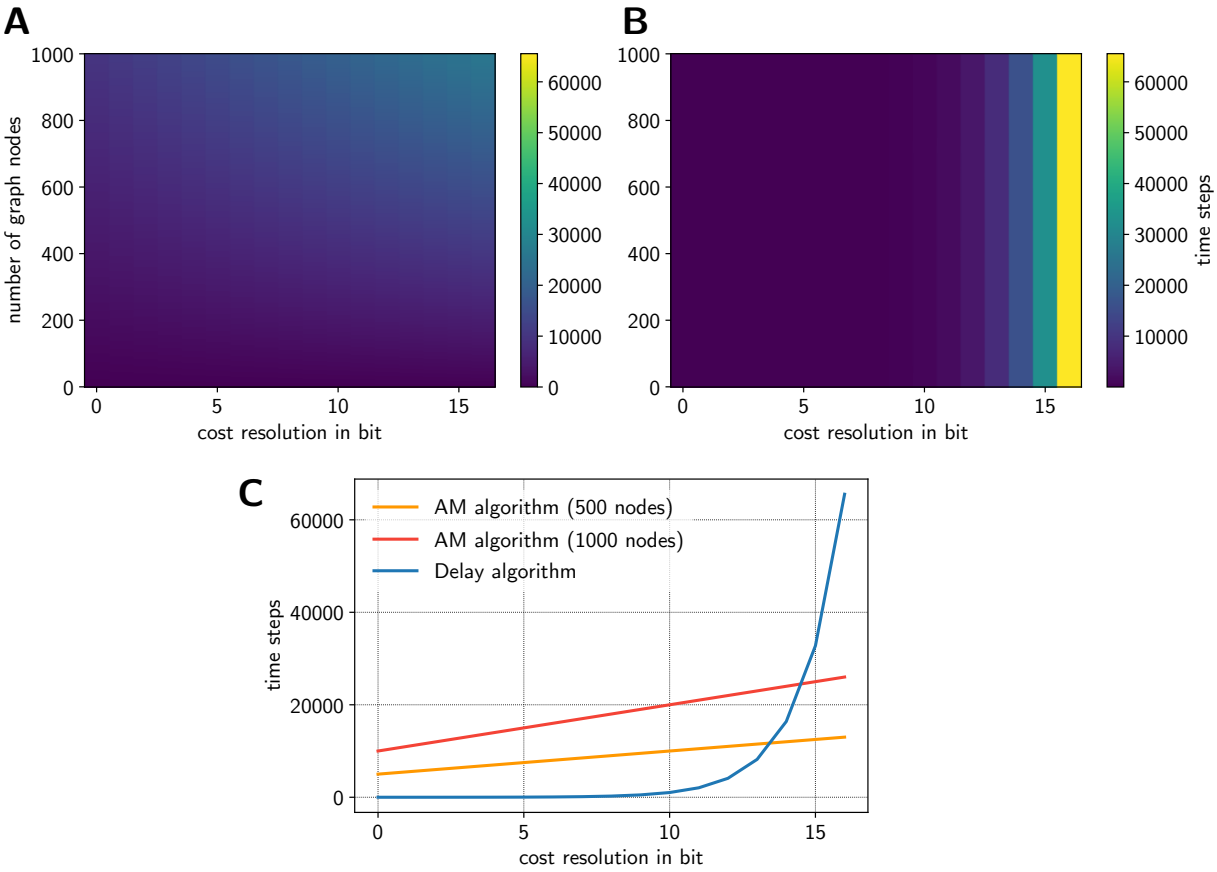


Figure 5.4: Run time comparison between the add-and-minimize (AM) and the delay algorithm, depending on the implemented number of nodes in the graph and the resolution of the costs in bit. **(A)** Number of necessary time steps for the AM algorithm. **(B)** Run time for the delay algorithm, which only depends on the resolution of the costs and not on the number of nodes. **(C)** The exponential rise in run time for the delay algorithm. In addition, two graph examples with 500 & 1000 nodes are plotted for the AM algorithm and show that the slope is higher when the graph becomes larger, but still follows a linear course.

algorithm becomes faster than the delay algorithm, when a certain resolution is exceeded. The necessary run time depending on resolution and number of nodes within the graph is shown in Figure 5.4.

There are two possible extensions for the algorithm. Currently the algorithm only provides the minimum cost value of the SSSP problem, but not the path itself, which is probably the largest drawback compared to the delay algorithm. We think that it should be possible to add a second *mirror network*. This mirror network is more simple and models nodes by simple neurons and edges by synapses. Nodes from the AM network are connected to the node neurons of the mirror network. By using plasticity, a mirror network represents the shortest path in its synapses. This would not only allow to store a shortest path with one source, it would allow to store solutions for different sources. For any shortest path problem, a new mirror network can be created, storing the result. With this, not only the computation, but also the memory is included in a neuronal structure, which would allow a full on-chip algorithm. The other aspect affects the costs. Since the costs enter the algorithm via neural activity, it can possibly be used for dynamic costs. By adding more external cost neurons, costs could be added dynamically

to every edge, depending on external input. For example, a routing algorithm for maps could easily be extended by live information about traffic, roadworks or road closures. While the delay algorithm requires a new initialization for every change in costs, the initialization of the AM algorithm needs to be performed only once and inputs can dynamically be fed into the system, changing the cost conditions by simply injecting spikes to the cost neurons.

Finally we can consider implementing the AM algorithm to other neuromorphic systems. First, the BrainScaleS chip, introduced in Section 2.3.2, could be an interesting candidate for the AM algorithm. Since it is able to simulate at high speeds, 1.000 to 10.000 faster then real time simulation speed, it could result in very fast solutions of the SSSP problem. But note that the BrainScaleS chip is non-deterministic. It is possibly necessary to make the algorithm robust against fluctuations of the membrane potential. Another interesting improvement could be possible by using the next generation of Loihi. The Loihi2 chip provides graded spikes with a resolution of up to 32 bits (see Section 2.3.3). The AM algorithm could possibly be adapted to reduce the amount of neurons drastically, which would increase the speed and improve scalability, while all other advantages would remain.

5.2 A balanced plasticity rule forms cell assemblies on Loihi

Contribution statement

All results within this section were performed by me. I did the experiments, the analysis and created the figures. My contribution to this section was 100%.

5.2.1 Introduction

In 1949, Donald O. Hebb did not only suggest a first and important correlation based plasticity mechanisms (introduced in Section 2.1.2), but did also hypothesize that memory in the brain may base on strongly connected groups of neurons, called *cell assemblies* (Abeles, 2011; Hebb, 1949). These strong connections are formed by repeated activation of a specific group of neurons that represents a specific memory. The formation is possibly based on various synaptic plasticity mechanisms (Carrillo-Reid, Lopez-Huerta, Garcia-Munoz, Theiss, & Arbuthnott, 2015; Chiappalone, Massobrio, & Martinoia, 2008; Hiratani & Fukai, 2014; Tetzlaff, Dasgupta, Kulvicius, & Wörgötter, 2015; van Rossum, Bi, & Turrigiano, 2000; Zenke, Agnes, & Gerstner, 2015).

An important feature of cell assemblies is its ability for pattern completion (Braitenberg, 1978; Buzsáki, 2010; Guzowski et al., 2004). If only a portion of the assembly is activated, the whole densely connected group follows. The idea: if only an association of a memory is given – activating only a small part of the memory related network –, possibly the whole memory can be activated as a consequence. This property is mediated by a threshold, a necessary minimum activity level, that determines if the cell assembly becomes active or not. Whereas cell assemblies provide a perfect concept for representing binary information, i.e. active or not, reliable computation over time, useful for learning low dimensional output functions, is hard to obtain. Several approaches have been applied (e.g. Tetzlaff et al., 2015; Zenke et al., 2015), but none of those concepts are able to solve the variability-stability problem, describes in the introduction, with solely biologically plausible methods. Novel approaches, like for example networks with inhomogeneous synaptic connectivity structures, outperform classical cell assembly approaches based on randomly connected networks in this task (Michaelis et al., 2020).

However, the basic mechanism of cell assemblies, especially its ability to represent stable binary information, could be a valuable ingredient for the hierarchical motor selection and execution (HMSE) model, introduced in Section 3.2. On the selectional level, it is important to keep track of which sequence is currently performed. Therefore, a cell assembly would not store a movement itself, but rather the state of the movement. This would help to avoid interference with competing alternative movements in the context of the HMSE model. Fortunately, previous studies provide a rich pool of insights about how different plasticity rules support the formation

of cell assemblies (for example Panda & Roy, 2017; Tetzlaff et al., 2015; van Rossum et al., 2000; Zenke et al., 2015). These insights can be used to derive basic principles of a cell assembly that can then be tuned to represent state information.

The following approach is a simple proof of principle that shows how cell assemblies can dynamically be added to a randomly connected spiking neural network (SNN). The approach is based on a balanced plasticity rule comprising Hebbian plasticity and synaptic scaling (Tetzlaff, Kolodziejski, Timme, Tsodyks, & Wörgötter, 2013; Turrigiano, 2008). A simple protocol provides not only the self-organization of a cell assembly, but also allows adding a second assembly in the network after the first one was learned. In addition, the randomly connected network is initialized with biologically plausible parameters. Finally, the whole simulation was performed on Loihi, which shows that complex learning rules, including homeostatic mechanisms, can successfully be implemented on the neuromorphic hardware.

5.2.2 Methods

Within this section I will first introduce the methods used to perform the cell assembly formation experiment, which contains two parts. First, it I describe how the network is initialized and how the chosen parameters are largely motivated by experimental findings. Second, the applied learning rule is briefly derived and its implementation on Loihi is explained.

Network initialization

The weights of the randomly connected SNN were initialized with a lognormal distribution. For several different brain areas it has been shown that excitatory postsynaptic potentials (EPSP) are distributed lognormal (Fukai et al., 2014), rather than following a Gaussian distribution. This results in many weak and some few strong synaptic weights. Such a distribution of weights is able to provide a self-sustained asynchronous-irregular activity (Kriener et al., 2014). The lognormal distribution for the weight w is defined as

$$p(w) = \frac{1}{\sigma w \sqrt{2\pi}} \cdot \exp\left(-\frac{(\ln(w) - \mu)^2}{2\sigma^2}\right), \tag{5.1}$$

where μ is the mean and σ is the standard deviation of the distribution. Teramae, Tsubo, and Fukai (2012) suggest the parameters $\sigma = 1.0$ and $\mu - \sigma^2 = \ln(0.2)$, based on experimental findings. Note that if the random variable w is distributed lognormal, $w \sim Lognormal(\mu, \sigma^2)$, then $\ln(w)$ is a normal random variable, i.e. $\ln(w) \sim \mathcal{N}(\mu, \sigma^2)$.

Since the weights on Loihi are 8 bit integer values which obviously differ from typical floating point weight values, it was necessary to scale the weights according to a stability criterion. In order to get the desired stable spiking activity, the eigenvalues were calculated and the largest eigenvalue, the so-called spectral radius was obtained. The spectral radius was chosen smaller

than 1 and preferable close to 0.8, according to (Lukoševičius, 2012). The parameters of the distribution were kept as $\sigma = 1.0$ and $\mu = \ln(0.2) + 1.0$, but the distribution was scaled with a factor of $s = 20/255$, according to

$$\tilde{p}(x) = \lfloor s \cdot p(x) \rceil, \tag{5.2}$$

where $\lfloor \cdot \rceil$ denotes an integer operation. An example of a weight matrix based on the lognormal initialization is shown in Figure 5.6A1 and the related distribution of weights is shown in Figure 5.6B1. The mean spectral radius for the weight matrix W over 10 network initializations was finally $\rho(W) = 0.895 \pm 0.039$.

The neurons within the network were chosen either with only inhibitory or only excitatory synapses, according to Dale's principle (see Section 2.1.1). The ratio between those excitatory and inhibitory neurons was chosen 4 : 1. This ratio is biologically motivated and can be found in the mammalian cortex (Wehr & Zador, 2003). The basic structure of the network is illustrated in Figure 5.5A.

Balanced synaptic learning rule

The learning rule, applied to the network includes two processes. One process is simply Hebbian plasticity as introduced in Section 2.1.2. The other component is *synaptic scaling*, a homeostatic mechanism that helps to stabilize the weights. Hebbian plasticity implements a positive-feedback that accelerates the synaptic strength between neurons with correlated activity and reduces the synaptic strength to zero if neurons are not co-active. This process tends to destabilize network activity which can be stabilized again by synaptic scaling (Abbott & Nelson, 2000; Tetzlaff et al., 2011; Turrigiano & Nelson, 2004). The underlying biological process of synaptic scaling is linked to calcium-dependent sensors that detect changes in their own firing rates (Turrigiano, 2008). The number of receptors is regulated depending on those firing rate changes.

We can define a stabilizing learning rule with the two mentioned components. A discrete weight change dw on Loihi is determined by a correlation based Hebbian component and a synaptic scaling component as

$$dw = c_H \cdot x_1 \cdot y_1 \cdot x_0 \cdot y_0 - c_S \cdot y_0 \cdot w^2, \tag{5.3}$$

where $c_H = 2^{-4}$ and $c_S = 2^{-3}$ are the learning rates for the Hebbian term and the synaptic scaling term respectively. $x_0 \in \{0, 1\}$ and $y_0 \in \{0, 1\}$ indicate a spike of the pre- and post-synaptic neuron. In addition, x_1 and y_1 represent activity traces of the pre- and post-synaptic neuron. Therefore, the first term increases the synaptic weight, weighted by the product of the pre- and post-synaptic activity. The second term is the synaptic scaling term. Tetzlaff et al. (2011) have shown that stability is generally reached with a convex non-linear weight, which leads to a squared weight dependency. The synaptic scaling depends on the spiking behavior of

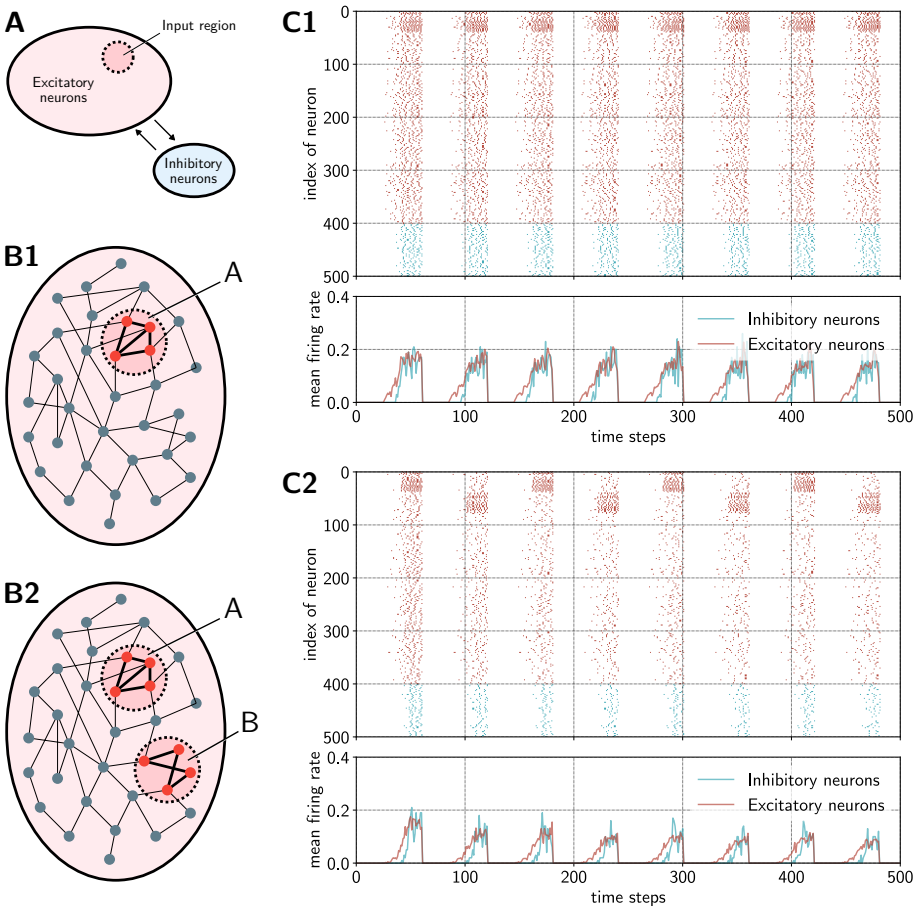


Figure 5.5: The cell assembly formation experiment. **(A)** The main network structure with a pool of excitatory and a pool of inhibitory neurons. In addition, an input region is defined that stimulates a fraction of the excitatory neurons. **(B)** The excitatory pool of neurons with input regions. The first image shows one input region A and represents the first experiment (top). For the second experiment, two distinct input regions A and B are stimulated alternatingly (bottom). Bold black connections represent stronger synaptic connections between neurons. **(C)** Spike trains and firing rates for both experiments (top/bottom) and both neuron pools, excitatory (red) and inhibitory (blue). The input regions can be recognized by strong activity within the spike trains. The network's spiking activity is reset after a trial.

the post-synaptic neuron y_0.

With this, the SNN is initialized based on several biologically plausible mechanisms. It is moreover equipped with a stabilizing learning rule, based on Hebbian plasticity and synaptic scaling. In a final step, a simple protocol is applied to the network structure which allows the self-organized formation of cell assemblies within the synaptic structure of the network.

5.2.3 Results

In order to form cell assemblies in the SNN described above, an input region within the pool of the excitatory neurons is defined, as illustrated in Figure 5.5A. One trial is simulated for 50 discrete time steps. During every trial, excitatory background noise in the network is active. After 30 time steps a strong input is given to the input region. After a trial, the activity in the

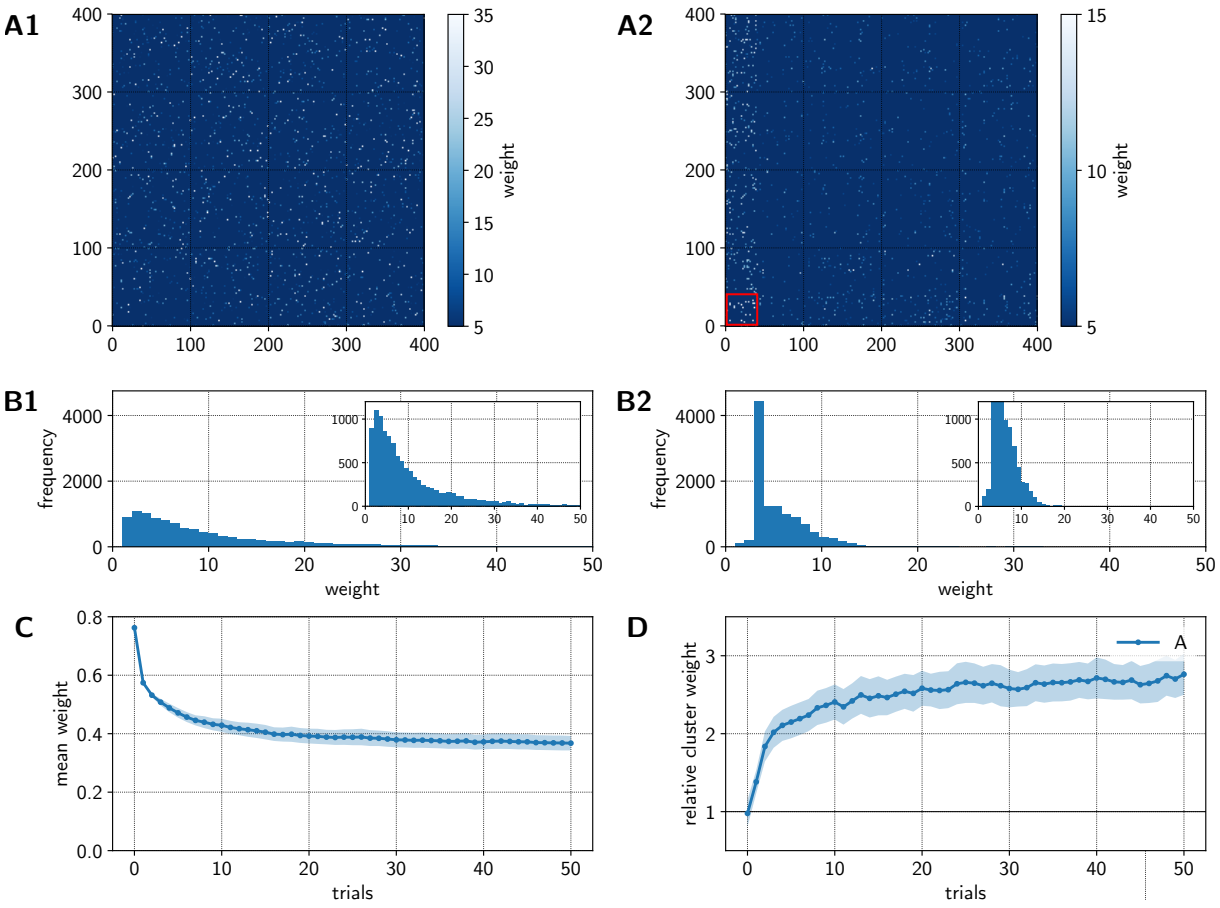


Figure 5.6: Weight analysis of a single cell assembly. **(A)** The initial weight matrix (left) and the weight matrix after learning (right). In the left bottom corner of the weight matrix, stronger connections emerged. **(B)** Distributions of the weights before (left) and after (right) the cell assembly formation. After learning, the tail of large weights is shorter and low weights are less frequent in favor of more moderate weights. **(C)** The total mean of all synaptic weights in the network over trials. It decreases quickly, but stabilizes after some trials. **(D)** The mean synaptic weight of the cell assembly region in relation to the total synaptic mean of the network over trials. The cell assembly gets strongly interconnected and stabilizes at a mean weight which is about 3 times higher than the network's total average weight.

network is reset and the next trial can start. For one input, called A, the protocol is shown in Figure 5.5C1. It is intended that the synapses within the input region form a cell assembly, a group of strongly interconnected neurons. Figure 5.5B1 shows the principle of this formation. The dotted circle illustrates the cell assembly area. Included neurons, undergoing input, are shown in red and bold lines between the neurons indicate strong synaptic weights. After this protocol is applied for 50 trials, the weights are stored for the second experiment. Within this second experiment, two input regions are stimulated in an alternating manner. We denote the two input regions as A and B respectively. This is again applied for 50 trials. The spiking activity for the second experiment is shown in Figure 5.5C2. The hypothesis is that a second cell assembly B is added to the network while the first assembly A is maintained, like it is illustrated in Figure 5.5B2.

A comparison between the initial weight matrix and the matrix after the first assembly was formed is shown in Figure 5.6A. Comparing those two matrices clearly indicates a change in

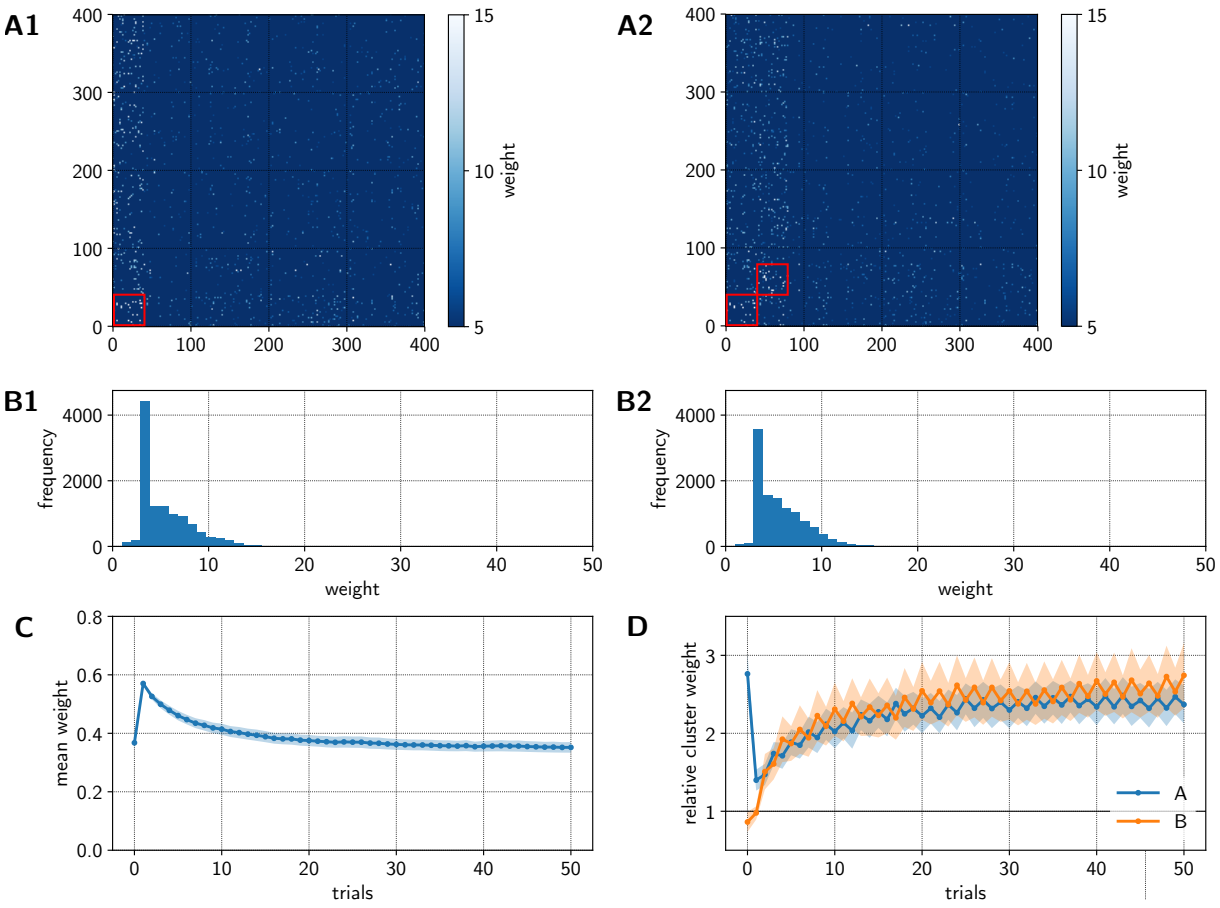


Figure 5.7: Weight analysis of an additional second cell assembly. **(A)** The weight matrix with one cell assembly (left) and the weight matrix after a second cell assembly was learned (right). **(B)** Distributions of the weights before (left) and after (right) adding an additional cell assembly. The distributions have not substantially changed. **(C)** The total mean of all synaptic weights in the network over trials. The stimulation of the second input region increases the mean weight quickly, but stabilizes again after some few trials. **(D)** The mean synaptic weight of the cell assembly regions in relation to the total synaptic weight mean of the network over trials. The cell assemblies become strongly interconnected and stabilize at a mean weight which is about $2-3$ times higher than the network's total average weight. The "zigzag" pattern indicates an ongoing moderate competition between the cell assemblies.

the weight structure. In the left bottom corner of the right matrix, the strength of the synaptic weights is increased in relation to the background. The distribution of the weights has changed a bit, as shown in Figure 5.6B. The long tail of the distribution is lost and also very small weights are less frequent. Nevertheless, the exponential shape is in principle maintained. Figure 5.6C shows that the mean of the synaptic weights decreases quickly, but stabilizes after some time. This mean synaptic weight is averaged over 10 different network initializations. Finally, Figure 5.6D demonstrates that the mean synaptic weight within cluster A is in the end nearly three times higher than the mean synaptic weight of the whole network. This ratio stabilizes after some time, similar to the stabilization of the total mean. The relative cluster weight is also averaged over 10 network initializations.

The results of the second experiment are shown in Figure 5.7. Part A shows the weight matrices before and after the alternating input was presented. Note that Figure 5.7A1 equals Figure 5.6A2; the final weight matrix of the first experiment is the initial matrix of the second

experiment. After the second input region was added, the structure of the weight matrix has clearly changed and a second cell assembly has emerged. Histograms in Figure 5.7B demonstrate that the additional experiment has not changed the weight distribution essentially. Also the mean synaptic weight in Figure 5.7C ends up at the same level, after a short peak in the beginning of the experiment due to the changes induced by the second input region. Figure 5.7D shows that indeed the second input region causes a drastic reorganisation in the network. Cell assembly A looses significant connectivity while B gains connectivity quickly within the first 1 to 2 trials. After some time, both cell assemblies stabilize. Interestingly, a certain competition can consistently be observed, indicated by the "zigzag" pattern.

5.2.4 Discussion

A combination of a biologically plausible initialization of the network and a specific learning rule, based on Hebbian and homeostatic synaptic plasticity mechanisms, was used to form cell assemblies, fully self-organized. It was possible to not only form one cell assembly, but also to dynamically add a second cell assembly within the network. The weight matrix reflects the cell assemblies' representations and while the mean weight reduces in the first trials, it converges to a nearly constant value after some time. Likewise, the relative cluster mean reached a stable state but, however, remained in a competition when two assemblies are learned alternatingly. This self-organized approach was performed on the neuromorphic hardware Loihi, which shows that neuromorphic hardware is able to simulate highly biologically motivated SNNs.

This small study is not meant to be a comprehensive cell assembly study, it is first and foremost a building block for other applications. The precisely balanced network structure and plasticity mechanism is of value for many aspects within this thesis. First, the HMSE model, introduced in Section 3.2, could profit from the cell assembly formation. On the selection level of the model, it is required to store the state of the currently ongoing movement sequence. This possibly avoids an interference between different sequences and helps learning mechanisms to operate properly. Second, the reward-based remote supervised method (re-ReSuMe) approach requires an improved learning rule for the pooling layer, if the output learning rule should be able to effectively use spiking activity from the anisotropic network as input, as discussed in Section 3.3.3. Especially the synaptic scaling is a promising homeostatic approach that could provide a low-frequent spiking activity of the pooling layer neurons, which is essential for a good performance of the re-ReSuMe rule.

In conclusion, the cell assembly formation described in this section combines several promising properties which could be useful for a variety of future research approaches and applications. The code of the experiments is available online[2].

[2]https://github.com/sagacitysite/pelenet

5.3 Supplementary material for the anisotropic network analysis

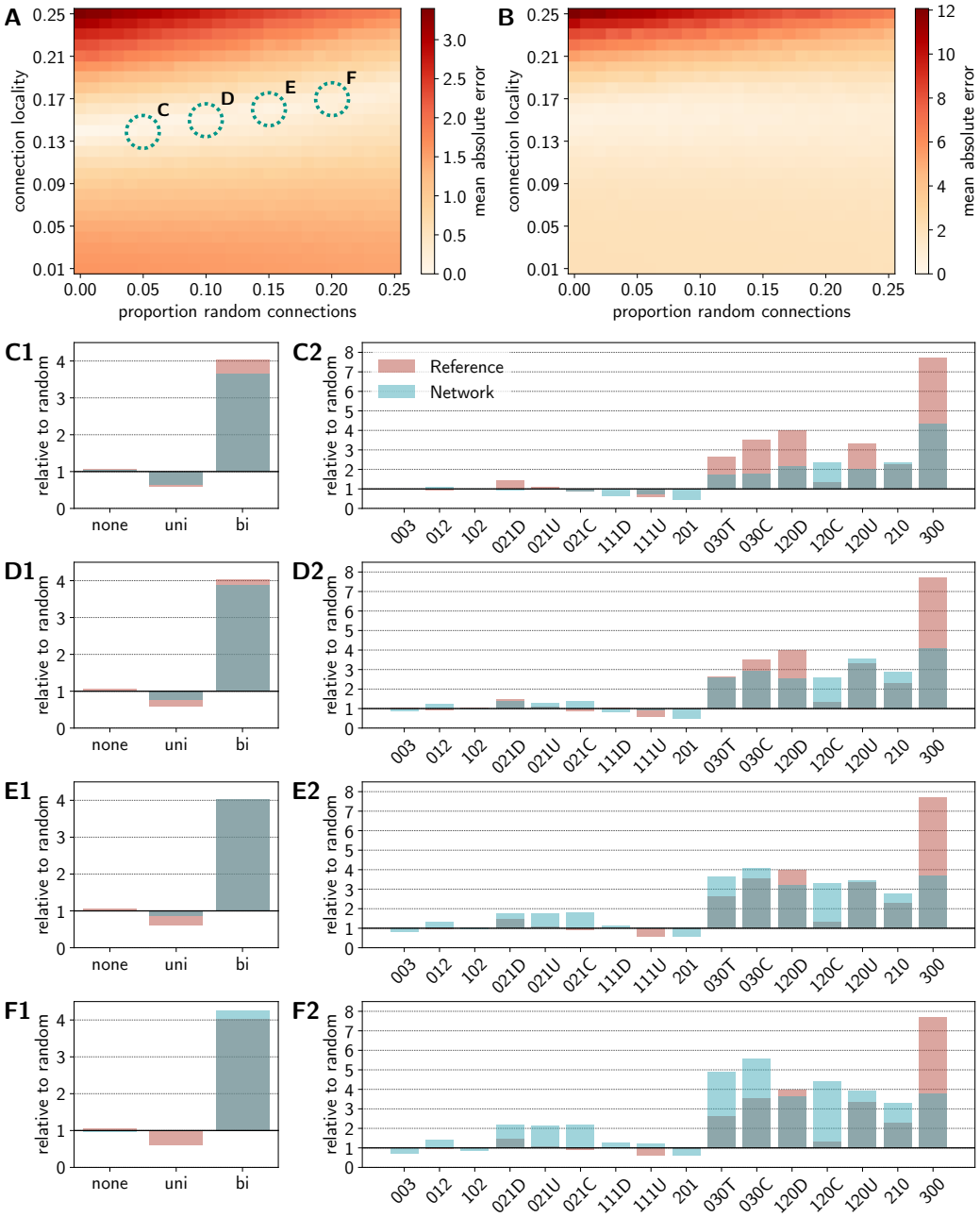


Figure 5.8: Parameter scan to match motifs between experimental data from Song, Sjöström, Reigl, Nelson, and Chklovskii (2005) and simulated data from the anisotropic network. **(A)** Mean absolute error between motif counts in the network's connectivity structure relative to random motifs for all *two* neuron motifs. The error depends on the proportion of random connections, replacing local connections, and the locality of the synaptic connections. Some parameters with a proportion of random connections and a low error are marked with green circles. **(B)** Mean absolute error between motif counts in the network's connectivity structure relative to random motifs for all *three* neuron motifs. **(C-F)** Relative motifs for all *two* (left) and *three* (right) neuron motifs for the parameter set related to errors marked with C to F in plot A.

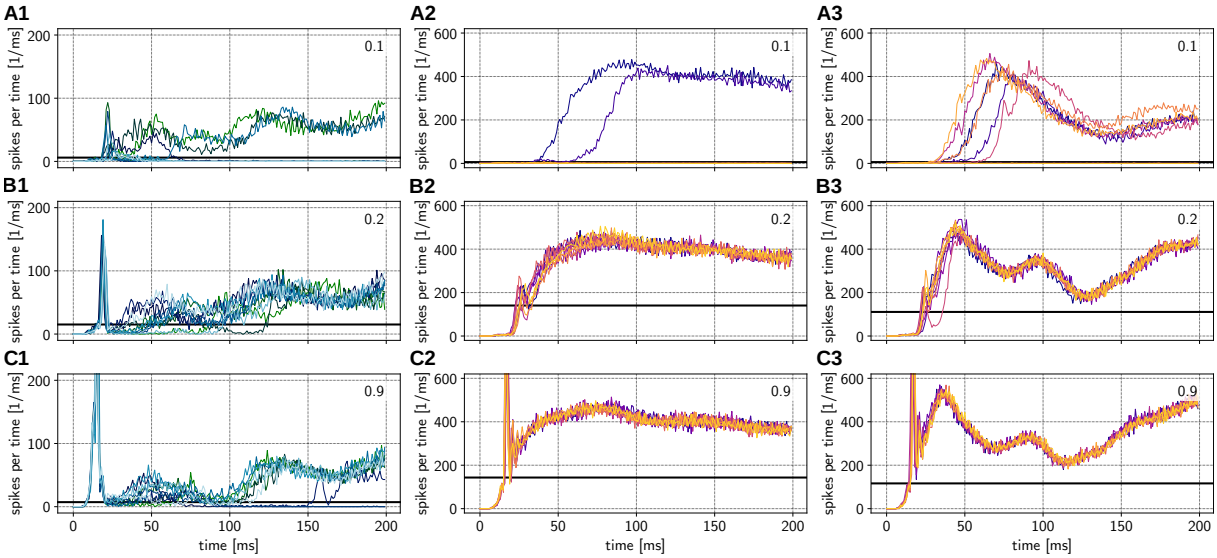


Figure 5.9: Firing rates of activity in the anisotropic network, initialized with lognormal weights. The firing rates are shown for the original parameter set (left), the biological parameter set before learning (center) and the biological parameter set after learning (right). It is based on an experiment with an input share of $r_I = 0.1$ (A), $r_I = 0.2$ (B) and $r_I = 0.9$ (C).

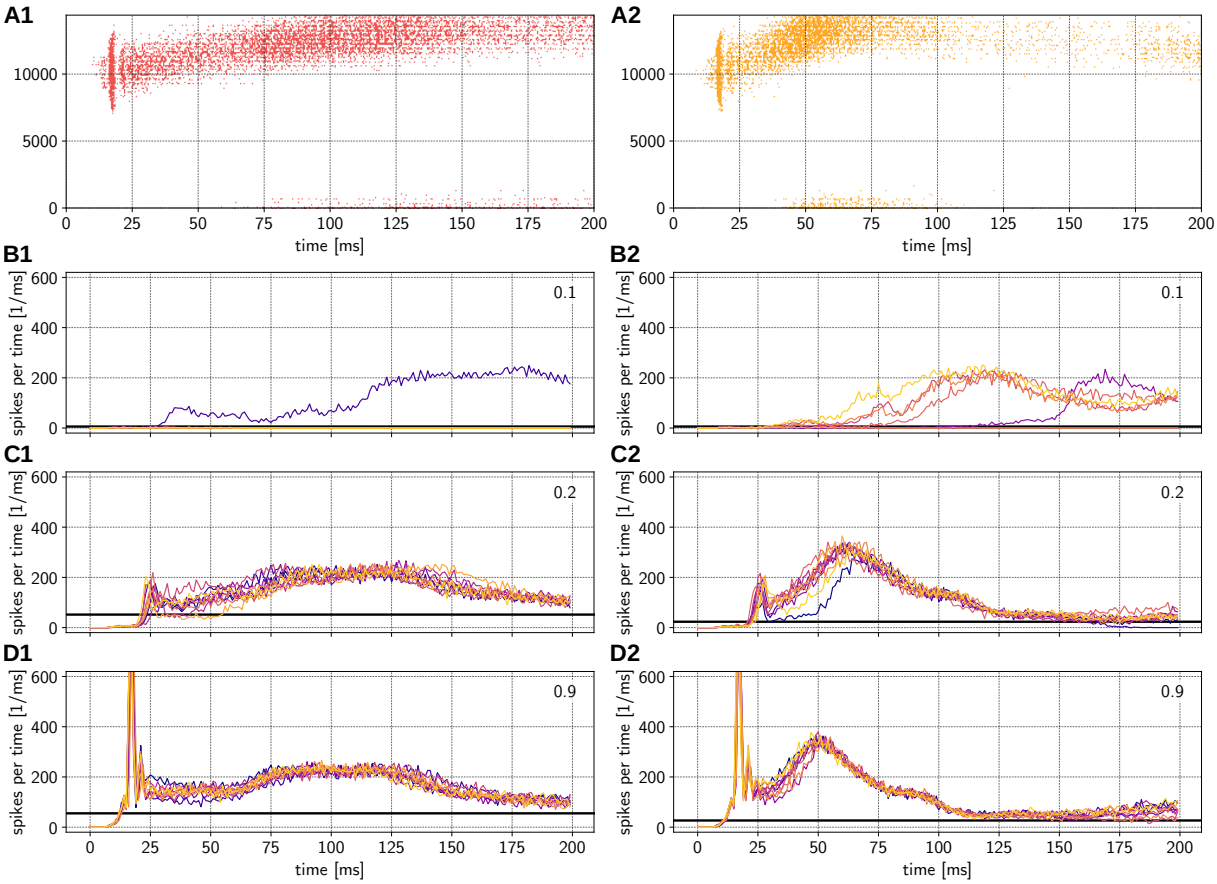


Figure 5.10: Spike trains and firing rates of activity in the anisotropic network, initialized with equal weights for the biological parameter set before learning (left) and the biological parameter set after learning (right). **(A)** Spike trains based on a trial with an input share of $r_I = 0.9$. **(B-D)** Firing rates are shown for input shares of $r_I = 0.1$, $r_I = 0.2$ and $r_I = 0.9$.

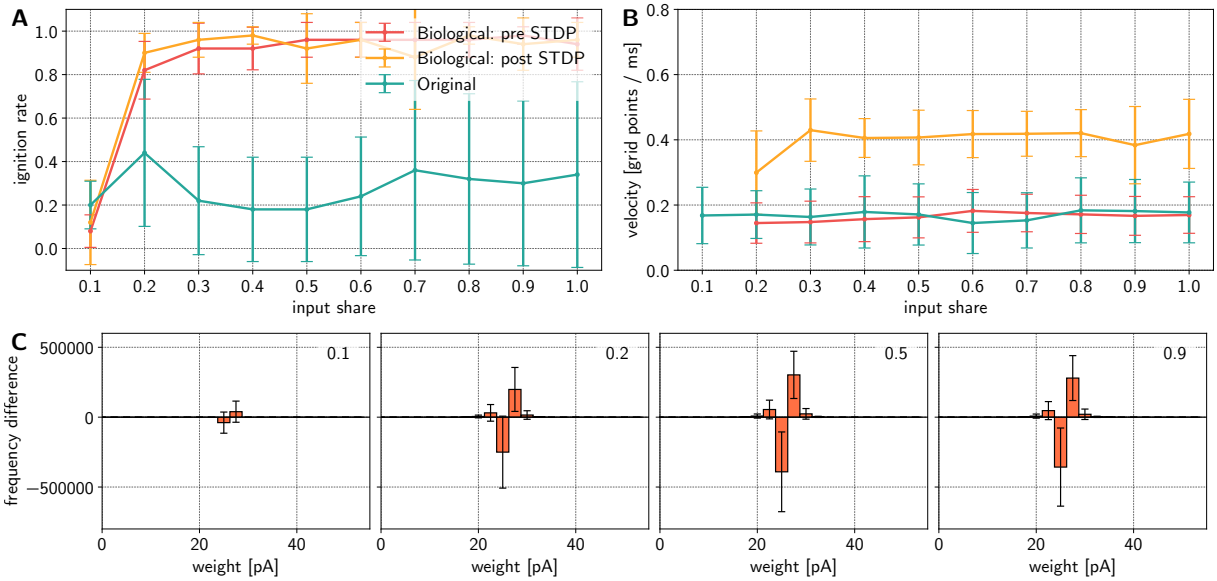


Figure 5.11: Ignition rates, velocities & weight changes of the anisotropic network, initialized with equal weights. **(A)** Ignition rate depending on the input share. A rate of 1 indicates that the activity in all trials was sustainably started by the input. A value of 0 means that the activity in all trials died out shortly after the input was given. **(B)** Velocity of the activity bump moving within the network depending on the input share. How velocities are obtained is described in the text in Section 3.3.2. **(C)** Synaptic weight differences between initial weights and weights after STDP was applied. The weight differences are shown for four input shares: 0.1, 0.2, 0.5 and 0.9 (from left to right).

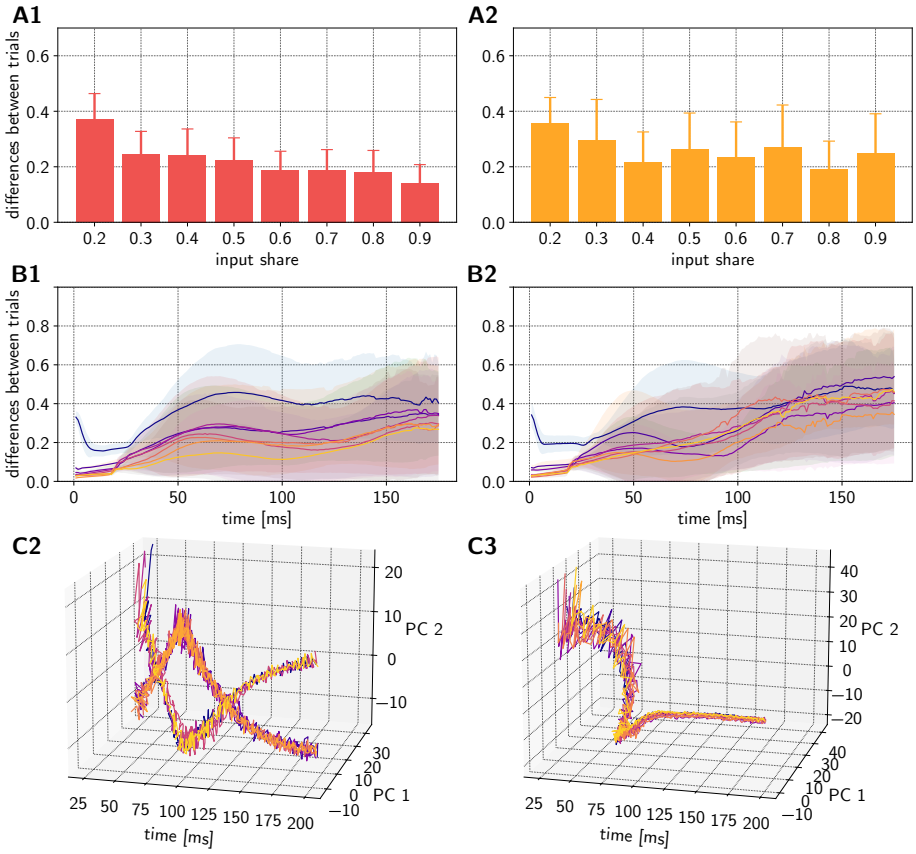


Figure 5.12: Trial-to-trial differences & PCA for activity in the anisotropic network, initialized with equal weights. Results are shown for the biological parameter set before learning (left) and the biological parameter set after learning (right). **(A)** Mean differences between trials depending on input shares. **(B)** Differences between trials for one experiment per input share. **(C)** The first principle components of the activity over time within a PCA space for an experiment with an input share of 0.9.

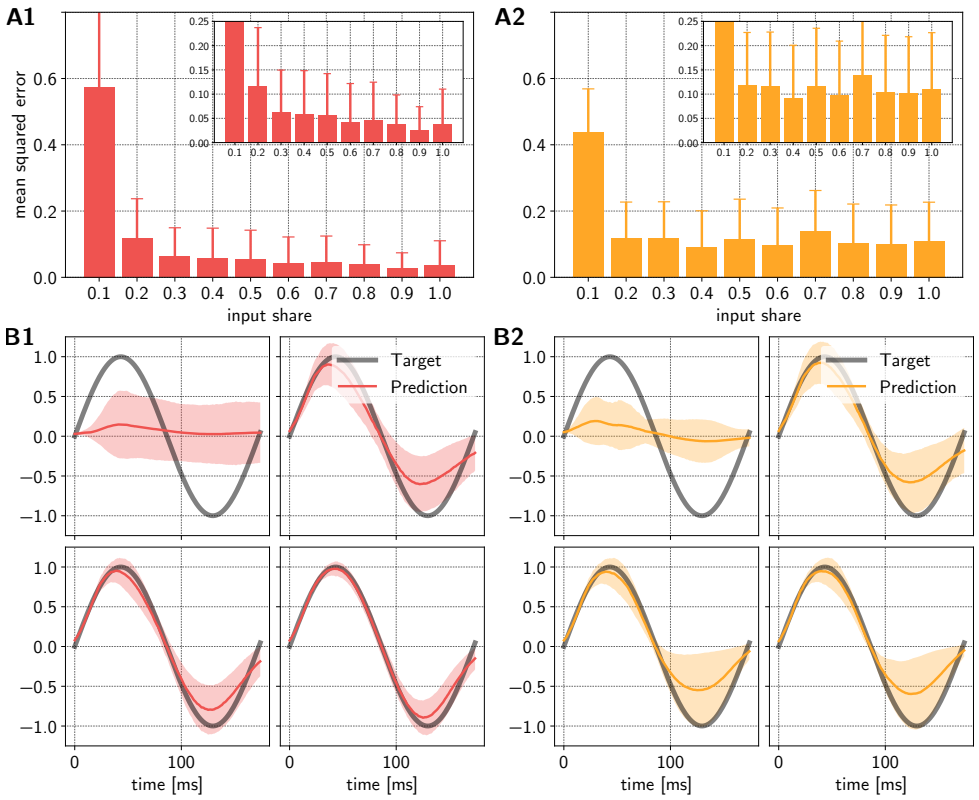


Figure 5.13: Learning a sine wave function based on the activity of the anisotropic network, initialized with equal weights. Results are shown for the biological parameter set before learning (left) and the biological parameter set after learning (right). **(A)** Mean squared errors between predicted and target function depending on input shares. Inlays show errors on a smaller scale. **(B)** Predicted and target functions for one experiment and four input shares: 0.1, 0.2, 0.5 and 0.9 (from left top to right bottom).

5.4 Parameter search for the re-ReSuMe output learning

This appendix section provides supplementary material to Section 3.3.3. Here I performed a parameter search that was applied to a spike train with 20 time steps in order to find optimal parameters for the reward-based remote supervised method (re-ReSuMe). Results are shown in Figure 5.14. The 6 varied parameters are:

- dw: learning rate
- $vThMant$: membrane threshold of the neuron
- $x1Impulse$: spike trace amplitude
- $x1TimeConstant$: spike trace time constant
- $compartmentCurrentDecay$: current decay of the synaptic input
- $compartmentVoltageDecay$: voltage decay of the membrane potential

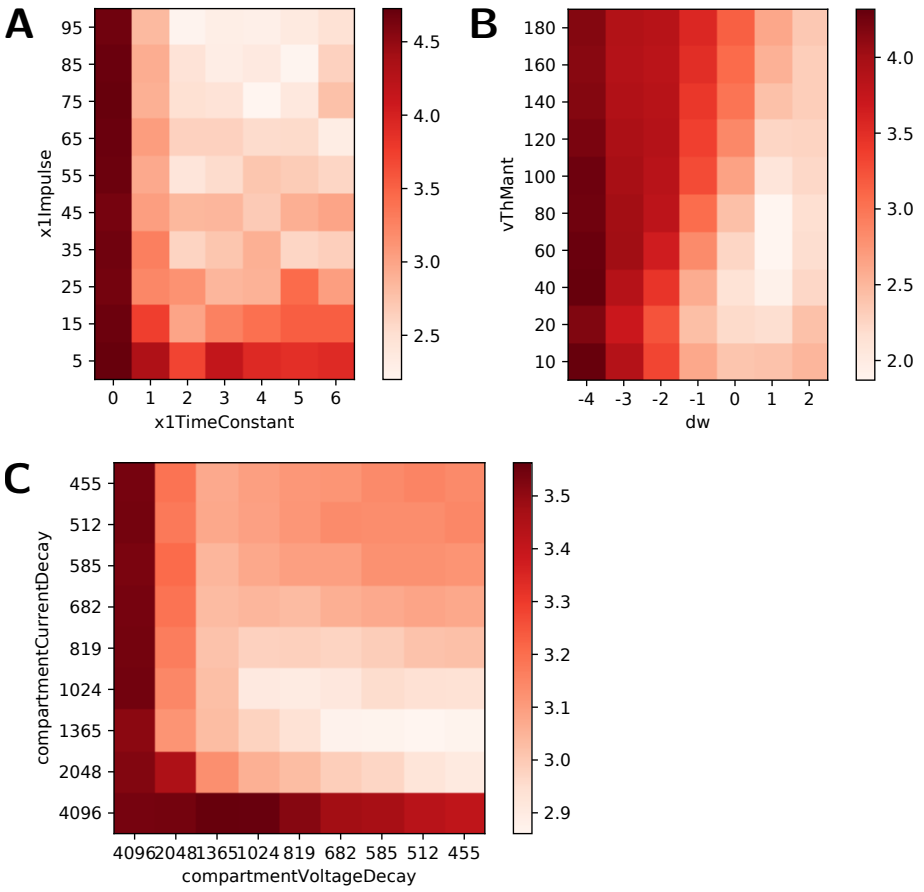


Figure 5.14: A parameter search for the re-ReSuMe output learning. Darker colors indicate higher errors. The 6 applied parameters within this scan are described in the corresponding text.

Chapter 6

Papers and Preprints

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 26 November 2020
doi: 10.3389/fnbot.2020.589532

Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi

Carlo Michaelis, Andrew B. Lehr and Christian Tetzlaff*

Department of Computational Neuroscience, University of Göttingen, Göttingen, Germany

Neuromorphic hardware has several promising advantages compared to von Neumann architectures and is highly interesting for robot control. However, despite the high speed and energy efficiency of neuromorphic computing, algorithms utilizing this hardware in control scenarios are still rare. One problem is the transition from fast spiking activity on the hardware, which acts on a timescale of a few milliseconds, to a control-relevant timescale on the order of hundreds of milliseconds. Another problem is the execution of complex trajectories, which requires spiking activity to contain sufficient variability, while at the same time, for reliable performance, network dynamics must be adequately robust against noise. In this study we exploit a recently developed biologically-inspired spiking neural network model, the so-called anisotropic network. We identified and transferred the core principles of the anisotropic network to neuromorphic hardware using Intel's neuromorphic research chip Loihi and validated the system on trajectories from a motor-control task performed by a robot arm. We developed a network architecture including the anisotropic network and a pooling layer which allows fast spike read-out from the chip and performs an inherent regularization. With this, we show that the anisotropic network on Loihi reliably encodes sequential patterns of neural activity, each representing a robotic action, and that the patterns allow the generation of multidimensional trajectories on control-relevant timescales. Taken together, our study presents a new algorithm that allows the generation of complex robotic movements as a building block for robotic control using state of the art neuromorphic hardware.

Keywords: robot control, neuromorphic computing, Loihi, anisotropic network, spiking neural network, computational neuroscience

OPEN ACCESS

Edited by:
Jorg Conradt,
Royal Institute of Technology, Sweden

Reviewed by:
Alejandro Linares-Barranco,
Sevilla University, Spain
Terrence C. Stewart,
National Research Council Canada (NRC-CNRC), Canada

****Correspondence:***
Carlo Michaelis
carlo.michaelis@phys.uni-goettingen.de

Received: *30 July 2020*
Accepted: *28 October 2020*
Published: *26 November 2020*

Citation:
Michaelis C, Lehr AB and Tetzlaff C (2020) Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi. Front. Neurorobot. 14:589532. doi: 10.3389/fnbot.2020.589532

1. INTRODUCTION

During infancy, humans acquire fine motor control, allowing flexible interaction with real world objects. For example, most humans can effortlessly grasp a glass of water, despite variations in object shape and surroundings. However, achieving this level of flexibility in artificial autonomous systems is a difficult problem. To accomplish this, such a system must accurately classify inputs and take appropriate actions under noisy conditions. Thus, increasing robustness to input noise is crucial for the development of reliable autonomous systems (Khalastchi et al., 2011; Naseer et al., 2018).

Neuromorphic hardware is based on highly parallel bio-inspired computing, which employs decentralized neuron-like computational units. Instead of the classical separation of processing and memory, on neuromorphic hardware information is both processed and stored in a network of these computational units. Neuromorphic architectures offer faster and more energy-efficient

computation than traditional CPUs or GPUs (Blouw et al., 2019; Tang et al., 2019), which is a vital feature for autonomous systems. However, porting existing robot control algorithms (e.g., Ijspeert et al., 2002) to neuromorphic hardware is *per se* ambitious (but see Eliasmith and Anderson, 2004; DeWolf et al., 2016; Voelker and Eliasmith, 2017) and difficult to optimize to the specific hardware architecture. At the same time, the development of new algorithms is also challenging due to the decentralized design principle of neuromorphic hardware as a network of computational units (Lee et al., 2018).

The basic network type for the various neuromorphic architectures developed in recent years (Schemmel et al., 2010; Furber et al., 2014; Davies et al., 2018; Neckar et al., 2018) are spiking neural networks (SNNs), coined third generation neural networks (for review, see Maass, 1997; Tavanaei et al., 2019). In particular, the reservoir computing paradigm, such as echo state networks (Jaeger, 2001, 2007) or liquid state machines (Maass et al., 2002), often serves as an algorithmic basis. In reservoir computing a randomly connected SNN provides a "reservoir" of diverse computations, which can be exploited by training weights from the reservoir units to additional units that constitute time-dependent outputs of the system.

The internal dynamics of the reservoir or SNN generally provide a sufficient level of variability such that arbitrary output functions on a control-relevant timescale can be read out. However, the system fails if the input is noisy or perturbations arise while the trajectory is being performed (Maass et al., 2002; Sussillo and Abbott, 2009; Laje and Buonomano, 2013; Hennequin et al., 2014). That is to say, spiking dynamics in SNNs are often unstable, meaning that small changes in the initial conditions result in different spiking patterns (Sompolinsky et al., 1988; Van Vreeswijk and Sompolinsky, 1996; Brunel, 2000; London et al., 2010). Thus, when an output is trained using such a spiking pattern, low levels of noise lead to a deviation of the estimated output from the target output and stable trajectories can only be obtained on a timescale of milliseconds. On the other hand, attractor dynamics provide highly stable, persistent activity (Amit, 1992; Tsodyks, 1999); however, they tend to lack the variability in the spiking dynamics required for complex output learning (Nachstedt and Tetzlaff, 2017). This implies a stability-variability trade-off, also denoted as a robustness-flexibility trade-off (Pehlevan et al., 2018).

A number of approaches have been developed in recent years to stabilize the spiking dynamics of SNNs while retaining sufficient variability for output learning (Laje and Buonomano, 2013; Hennequin et al., 2014; Pehlevan et al., 2018; Vincent-Lamarre et al., 2020). To improve stability, recent approaches used feed-forward structures (Pehlevan et al., 2018) or employed supervised learning rules (Laje and Buonomano, 2013). While feed-forward structures provide stable activity patterns, in general these play out on a very fast timescale (Zheng and Triesch, 2014) or require neural/synaptic adaptation such that activity moves between neuron groups (York and Van Rossum, 2009; Itskov et al., 2011; Murray et al., 2017; Maes et al., 2020). And since for supervised learning all states in the network need to be accessible at each computing unit, these so-called global learning rules are not compatible with most neuromorphic hardware.

Thus, achieving stable activity patterns on a control-relevant timescale in a network architecture and learning regime capable of running on neuromorphic hardware remains an open problem. Necessary criteria are that (1) learning or adaptation mechanisms in the SNN should be local to individual synapses, or synapses should be static, (2) sequential activity patterns should remain active for hundreds of milliseconds, (3) spike patterns should contain sufficient variability for arbitrary output learning, and (4) the network should possess noise-robust neuronal dynamics. Meeting these criteria is especially difficult for recurrent network structures, like reservoir networks. However, the so-called anisotropic network model appears to be a promising candidate (Spreizer et al., 2019). The model is based on a biologically-inspired rule for forming spatially asymmetric non-plastic connections. Thus, synapses are static, meeting the first criterion, and the timescale of activity sequences is on the order of tens to hundreds of milliseconds, fulfilling the second criterion. However, whether the model also fulfills the third and fourth criteria, sufficient variability and stability under input noise, has not yet been assessed.

In this paper we use the anisotropic network as a building block for a novel algorithm yielding robust robotic control. We implement the network architecture on Kapoho Bay, a neuromorphic hardware system from Intel containing two Loihi chips (Davies et al., 2018), and show that this approach can be used to learn complex trajectories under noisy input conditions on a control-relevant timescale. Furthermore, we demonstrate that this neuromorphic network architecture can not only robustly represent complex trajectories, but even generalize beyond its training experience.

2. METHODS

We first describe the architecture of the novel algorithm implemented on the neuromorphic chip Loihi, which supports robust robotic control of movement trajectories. The anisotropic network and its implementation is then explained in detail. Finally analyses methods to evaluate the implementation of the anisotropic network on Loihi, the stability of its network dynamics, and the learning of complex movement trajectories are described.

2.1. Architecture of the Algorithm for Robotic Control

The architecture, shown in **Figure 1A**, was designed to support the storage and execution of stable movement trajectories in real-time. The architecture consists of an input layer, an anisotropic network layer, and a pooling layer, all of which are fully implemented on Loihi. Spike patterns from the anisotropic network or the pooling layer are read out and serve as the basis for training output units.

The basic computational structure for the robotic control algorithm is the anisotropic network. Excitatory and inhibitory neurons are initialized with local, spatially inhomogeneous

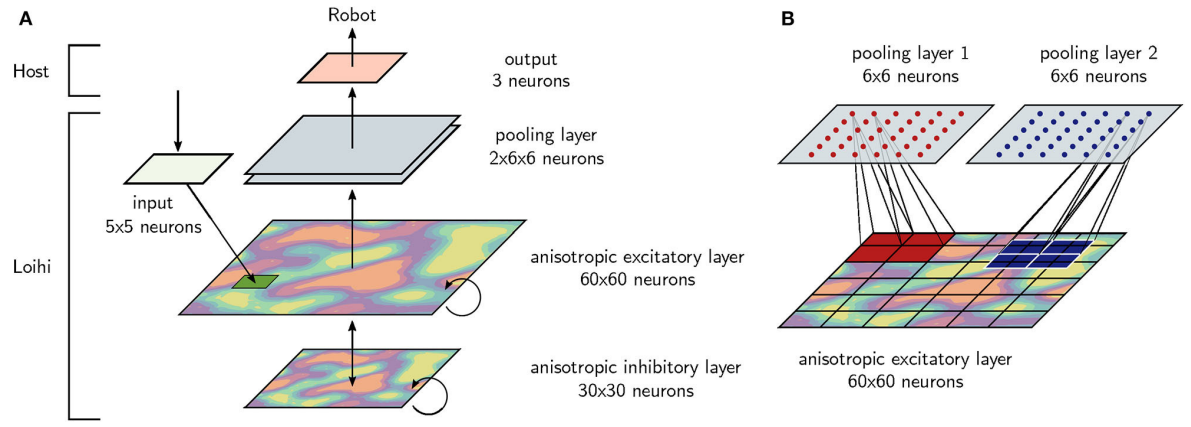


FIGURE 1 | The architecture of the robot control algorithm. **(A)** The anisotropic network is the basis of the architecture, indicated by the colored layers, and consists of 3,600 excitatory and 900 inhibitory neurons. An area of 25 excitatory neurons receive external input. The excitatory neurons of the anisotropic network project to a pooling layer. The anisotropic network, the input neurons and the 72 pooling neurons are simulated on-chip. Finally, the spiking activity is used to train a trajectory. **(B)** In detail, the pooling layer contains 72 neurons organized into two 6 × 6 grids. Each pooling neuron receives all-to-one feed-forward connections from a 10 × 10 square patch of the anisotropic network's excitatory layer. Hence, the pooling layer has 10 × 10 window size and stride 5.

connections as described below. An input is connected to a grid of 5 × 5 excitatory neurons in order to start spiking activity with a short input pulse.

The excitatory neurons of the anisotropic network are connected to a pooling layer with 72 excitatory neurons. Pooling layer neurons are organized into two grids with a size of 6 × 6 neurons, as shown in **Figure 1B**. Each neuron in the pooling layer receives input from a 10 × 10 group of excitatory neurons from the anisotropic network. These projections are all-to-one and all feed-feedforward weights are equal. In other words, the pooling layer has 10 × 10 window size and stride 5.

Depending on the task, either the excitatory neurons of the anisotropic network or the 72 neurons of the pooling layer are read out. Since reading out data from Loihi is a bottle neck that reduces the simulation speed considerably, the pooling layer is designed to reduce read out and therefore increase simulation speed. Finally, linear regression is applied to the spiking activity of the read-out (see section 2.5).

2.2. The Anisotropic Network

We briefly describe the main principles of the anisotropic network. For an in depth treatment, we refer readers to Spreizer et al. (2019).

In locally connected random networks (LCRNs), neurons are distributed in (connectivity) space (e.g., on a 2D grid or torus) and the connection probability between two neurons decreases (possibly non-monotonically) with the distance between them. Stable bumps of spatially localized activity can arise in LCRNs (Roxin et al., 2005; Hutt, 2008; Spreizer et al., 2017, 2019) and these activity bumps can move through the network in a stream-like manner if spatial asymmetries are introduced into the local connectivity (Spreizer et al., 2019).

The anisotropic EI-network consists of both excitatory and inhibitory neurons arranged on a 2D torus. Neurons project their axons in a distance dependent way with connection probability decreasing monotonically according to a Gaussian distribution. In a standard LCRN, axon projection profiles are centered at the neuron and axons project symmetrically in all directions. In the anisotropic EI-network, the Gaussian distribution is shifted for excitatory neurons such that connections to other excitatory neurons are formed preferentially in a particular direction (**Figure 2A**).

A so-called landscape is computed on the torus using Perlin noise (Perlin, 1985), and each point on the grid (neuron) is assigned a direction based on this. The Perlin landscape ensures that the preferred direction of nearby neurons are similar while preferred directions of those far apart are uncorrelated (**Figure 2B**). Each excitatory neuron's connectivity profile is shifted by one grid point in its preferred direction, resulting in spatially asymmetric but correlated connectivity. When a set of neurons in close proximity are stimulated, spatio-temporal sequences of activity lasting tens to hundreds of milliseconds are elicited (**Figure 2C**).

Taken together, a biologically plausible rule can generate spatially asymmetric connectivity structures supporting spatio-temporal sequences. Spreizer et al. (2019) show that if (1) individual neurons project a small fraction (∼2–5%) of their axons preferentially in a specific direction (**Figure 2A**), and (2) neighboring neurons prefer similar directions (**Figure 2B**), then sequences of neural activity propagate through the network (**Figure 2C**). This simple generative connectivity rule results in feed-forward paths through the otherwise locally connected random network.

2.3. Anisotropic Network Implementation

We adapted the anisotropic EI-network model from Spreizer et al. (2019). Since the total number of connections currently supported by the Loihi NxSDK-API is limited (see section 4),

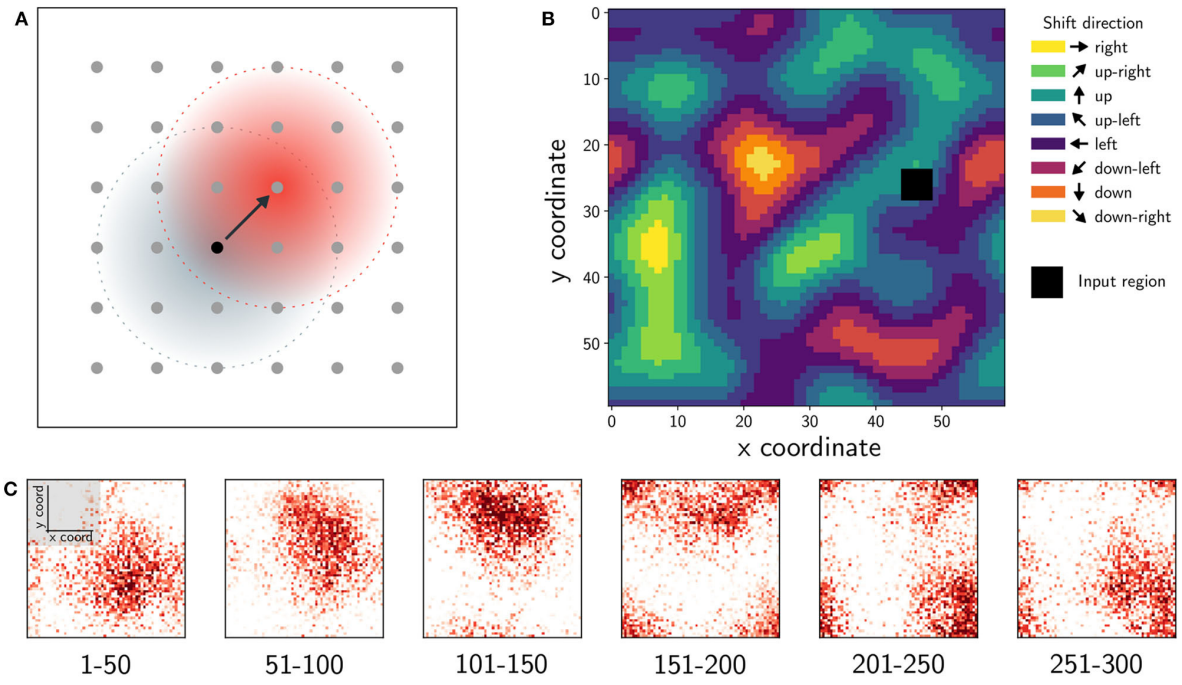


FIGURE 2 | The connectivity underlying the anisotropic network and resulting network dynamics. **(A)** Neurons are evenly distributed on a 2D grid, folded to form a torus. The black dot represents a reference neuron. From here outgoing connections can be drawn locally with a Gaussian distribution, either symmetrically (gray-blue) or non-symmetrically (red). In the non-symmetrical case the center of the Gaussian distribution is shifted by one neuron. This shift can be chosen in different ways. Either with the same shift direction for every neuron (homogeneously), a random shift direction for every neuron, or with a specific distribution. **(B)** If for each neuron the shift direction is chosen based on Perlin noise, the distribution of shift directions for the whole network results in an *anisotropic* connectivity structure. The black region represents spatially-localized input. **(C)** Given local input, the anisotropic network forms a bump of activity, which moves in a stream-like pattern through the inhomogeneous structure of connections. Here, we used the Loihi implementation and binned the spikes into non-overlapping time windows. Each graph shows the average firing rate over 50 time steps, color code depicts firing rate.

it was necessary to reduce network size by a factor of four to $npop_E = 3,600$ and $npop_I = 900$. Each neuron projects to $p_{conn} \times npop_E = 180$ excitatory targets and $p_{conn} \times npop_I = 45$ inhibitory targets, where $p_{conn} = 0.05$ is the connection probability. Connection probability decreases with distance according to a Gaussian distribution with space constants given in **Supplementary Table 1**. We first adapted the anisotropic EI-network model within NEST and then transferred it to Loihi, tuning the network to qualitatively match the behavior of the NEST simulation.

NEST Implementation

Neurons were modeled as leaky integrate-and-fire (LIF) neurons, with sub-threshold membrane potential v of neuron i evolving according to:

$$C_m \frac{dv_i}{dt} = -g_L(v_i(t) - E_L) + I_i(t) + I_i^{input}(t), \qquad (1)$$

where C_m is the membrane capacitance, g_L the leak conductance, and E_L the reversal potential. For neuron i, $I_i(t)$ is the total synaptic current from its recurrent connections and $I_i^{input}(t)$ the current induced by external input.

The total synaptic current $I_i(t)$ to neuron i at its recurrent synapses is the sum of the current transients at each of its synapses, $I_i(t) = \sum_j I_{ij}(t)$. When a pre-synaptic neuron spikes, a current transient is elicited with temporal profile given by an alpha function:

$$I_{ij}(t) = J^{syn} \frac{t - t_{j,k}}{\tau_{syn}} \exp\left(- \frac{t - t_{j,k}}{\tau_{syn}} \right). \qquad (2)$$

Note here that the superscript syn can denote both excitatory (exc) and inhibitory (inh) synapses. Synaptic strength is J^{syn}, synaptic time constant is τ_{syn}, and spike time is $t_{j,k}$ for the k^{th} spike from neuron j.

To compensate for the decreased network size and hence fewer recurrent connections (see above) we scaled up the synaptic weights. The excitatory synaptic current was scaled up by a factor of four to $J^{exc} = 40.0\,pA$. To ensure persistent spiking activity in response to an input pulse, the ratio of recurrent inhibition and excitation was reduced to $g = 4$. As a result, $J^{inh} = -g \times J^{exc} = -160.0\,pA$.

Activity was triggered by external input to a subset of neighboring neurons, each of which receives an input pulse of 500

spikes with synaptic strength $J^{input} = 1.0\,pA$ arriving according to a Gaussian distribution with standard deviation of 1 *ms*.

Loihi Implementation

For the implementation on neuromorphic hardware we used the research chip Loihi from Intel (Davies et al., 2018), which is a digital and deterministic chip that is based on an asynchronous design. The board we used contains two chips, with each chip providing 128 neuron cores and three embedded x86 CPUs. Each neuron core time-multiplexes the calculation and allows the implementation of up to 1,024 neurons each. We distributed the total of 4,572 utilized neurons (reservoir and pooling layer) with 20 neurons per core. Computation on the chip is performed in discrete time steps and has no relation to physical time. Finally, the Loihi board is connected to a desktop computer, called host in the following, via a serial bus (USB).

We translated the `NEST` implementation to the `NxSDK` (version 0.9.5-daily-20191223) for Loihi, provided by Intel labs (Lin et al., 2018). For this, we developed a software framework `PeleNet`[1], based on the `NxSDK`, especially for reservoir networks on Loihi. This framework was used for all simulations in this study.

The Loihi chip implements a leaky integrate-and-fire (LIF) neuron with current-based synapses and the membrane potential v of neuron i evolves according to

$$\frac{dv_i}{dt} = -\tau_v^{-1} v_i(t) + I_i(t) + I_i^{input}(t) - v_{th}\sigma_i(t), \tag{3}$$

where τ_v describes the time constant, v_{th} the firing threshold, $I_i(t)$ the total synaptic current from recurrent connections, $I_i^{input}(t)$ the current induced by the input, and $\sigma_i(t)$ denotes whether neuron i spiked at time t. The first term on the right-hand side controls voltage decay, the second/third term increases the voltage according to the synaptic/input currents, and the last term resets the membrane potential after a spike occurs.

While the `NEST` implementation uses alpha-function shaped synaptic currents (see Equations 1 and 2), Loihi's current-based synapses implement instantaneous rise and exponential decay. The total synaptic current from recurrent connections to neuron i is given by

$$I_i(t) = \sum_{i \neq j} J_{ij}^{syn} (\alpha_I * \sigma_j)(t) + I_i^{bias}, \tag{4}$$

where J_{ij}^{syn} is the synaptic strength from neuron j to neuron i which can be excitatory (J^{exc}) or inhibitory (J^{inh}) and I_i^{bias} is a bias term. The $\sigma_j(t)$ represents the incoming spike train from neuron j and $\alpha_I(t)$ a synaptic filter. The spike train for a neuron j is given by a sum of Dirac delta functions with

$$\sigma_j(t) = \sum_k \delta(t - t_{j,k}), \tag{5}$$

where $t_{j,k}$ is the time of spike k for neuron j. The function simply indicates whether neuron j spiked in time step t. The spike train is convolved with a synaptic filter given by

$$\alpha_I(t) = \tau_I^{-1} \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{6}$$

where τ_I is a time constant and $H(t)$ the unit step function.

With Equations (5) and (6), we can bring Equation (4) into a form which is comparable to Equation (2). Setting $I_{bias} = 0$, we get

$$\begin{aligned} I_i(t) &= \sum_{i \neq j} J_{ij}^{syn} (\alpha_I * \sigma_j)(t) \\ &\overset{\text{Equation(5)}}{=} \sum_{i \neq j} J_{ij}^{syn} \sum_x \alpha_I(x) \sum_k \delta((t-x) - t_{j,k})) \\ &= \sum_{i \neq j} J_{ij}^{syn} \sum_k \alpha_I(t - t_{j,k}) \\ &\overset{\text{Equation(6)}}{=} \sum_{i \neq j} J_{ij}^{syn} \sum_k \tau_I^{-1} \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}). \end{aligned} \tag{7}$$

Due to the filter, the input current induced by a pre-synaptic spike decays exponentially for each following time step. And instead of rising slowly, at the time of a spike, $t = t_{j,k}$, synaptic current increases by $\tau_I^{-1} J_{ij}^{syn}$. Thus, compared to the neuron model from the anisotropic network implementation in `NEST` (Spreizer et al., 2019), the hardware-implemented neuron model on Loihi differs since it lacks a current rise time.

2.4. Comparing the Implementations

Network activity was started with the input mentioned above and 500 discrete time steps in Loihi and 500 *ms* in `NEST` were recorded. In `NEST` the resolution was set to $dt = 0.1\,ms$ (see also **Supplementary Table 1**) per simulation step, while in Loihi a physical time is not defined. After the simulation, the `NEST` spiking data was binned to 1 *ms* to match the Loihi data. Note that, given the refractory period of 2 *ms*, the binned spike trains still contain binary values, but with a less precise information about the sub-millisecond spike times. In the end, both data sets, the spike trains for `NEST` and the spike trains for Loihi contained 500 discrete steps.

To compare the spiking patterns between `NEST` and Loihi quantitatively, we calculated the mean firing rate of groups of excitatory neurons in both networks, which is shown in **Figure 3B**. For this, we split the two dimensional network topology into a 6 × 6 grid, analogous to the grid used for the pooling layer (see **Figure 1B**) such that each grid position represents a group of 100 neurons. The indices of the groups are chosen from top left to bottom right. For each group, we averaged the firing rate over the 500 time steps resulting in 36 values.

2.5. Stability and Output Learning

To analyse the stability (**Figure 5**) of the network and for the output learning (**Figure 6**), we applied another protocol. Note

[1] https://github.com/sagacitysite/pelenet/tree/neurorobotics

that from this point on, the NEST implementation was not used. Out of the 25 excitatory neurons connected to the input, we stimulated only 24 neurons such that 1 neuron stays silent (**Figure 4A**). This input grid then allows 25 different input configurations and therefore 25 different trials with a noise level of 4%. Every trial was recorded for 215 time steps and then the activity was stopped by resetting the membrane voltages. We did this by applying a C code that runs on one x86 core on each chip (a so called SNIP). After waiting 30 time steps, the next input was applied to the network. The applied protocol is indicated by arrows in **Figure 5A** on top of the spike train plots.

To learn trajectories from the spike patterns we used multiple linear regression, which was applied to two different tasks. In the *representation task*, we estimated model parameters based on all 25 trials and tested on one of them. In the *generalization task*, training was performed on only 24 trials and testing was done using the remaining trial. Both tasks are sketched in **Figure 4B**. To compare the anisotropic network with a classical reservoir computing approach, we also implemented a randomly connected network on Loihi and exchanged the anisotropic network in our network architecture with a randomly connected network of equal size. We set the parameters of the random network such that the main statistics of both networks match. The firing rate in both networks is in a range of $0.1 - 0.2$ spikes per number of neurons (**Figure 5A** bottom) and the mean Fano factor over all trials is relatively similar with $\overline{FF}_{\text{rand}} = 0.80 \pm 0.01$ for the randomly connected network and $\overline{FF}_{\text{aniso}} = 0.84 \pm 0.002$ for the anisotropic network.

After all data were recorded, in a first step, we prepared the data for the estimation of the regression model. Due to the slow rise in the firing rate of the network (**Figure 3C**), the very first time steps contain little information. Therefore, we omitted the first 5 time steps which reduces the length of the data set to 210 per trial. Before the linear regression was applied to the spike data, the spike trains were binned in order to smooth our spiking data. We used a sliding window with a width of 10 time steps, which reduced the length of the data set again from 210 to 200.

Next we used the binned data of the 200 time steps to estimate the regression parameters. In addition to the spiking data from the neurons, an intercept was added such that the number of parameters equal the number of neurons plus one. The linear regression model was performed on the CPU of the host computer, using the spiking data from the readout provided by Loihi. The two different tasks, the representation task and the generalization task, were performed using the spiking data from the anisotropic network as well as those from the randomly connected network. Furthermore, we estimated output weights based on either the pooling layer neurons (72 neurons) or the excitatory neurons of the reservoir (3, 600 neurons), see also **Figure 1**. The excitatory neuron readout serves as a control and compares the pooling layer approach to a traditional readout.

For the estimation based on all excitatory reservoir neurons, we applied an elastic net regularization (Zou and Hastie, 2005) to avoid overfitting, due to the numerous parameters. This regularization approach for regression models simply combines LASSO and ridge regression. We used the `fit_regularized` function from the `statsmodels` package in `Python`, which applies elastic net as

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left[\|y - X\beta\|_2^2 + \alpha \left((1-\lambda)\|\beta\|_2^2 + \lambda\|\beta\|_1 \right) \right]. \quad (8)$$

In this variant the parameter α determines the degree of regularization and λ balances between LASSO ($L1$ regularization) and ridge regression ($L2$ regularization).

To better compare the predicted function with the target function, we applied a Savitzky–Golay filter (Savitzky and Golay, 1964) to smooth the predicted function. For this we used the `savgol_filter` function of the `Python` package `scipy`. We chose a window length of 21 and an order of the polynomial of 1 as parameters for smoothing.

We trained our algorithm on 7 different trajectories performing ordinary robotic tasks. The tasks are *hide*, *unhide*, *move down*, *move up*, *pick and place*, *put on top* and *take down* (see e.g., Wörgötter et al. (2020)). Movement data is given in 3 dimensional Cartesian coordinates, resulting in three outputs or – biologically speaking – in three rate coded output neurons.

3. RESULTS

We start by demonstrating that the main principles of the anisotropic network are preserved by the Loihi implementation and then confirm that the Loihi-based anisotropic network admits noise-robust spiking dynamics. Based on these findings, we demonstrate that our architecture can learn complex trajectories under noisy input conditions.

3.1. Implementing the Computer-Based Anisotropic Network on Loihi

Due to the different hardware architectures, we first assess the extent to which the Loihi-based implementation of the anisotropic network agrees with the computer-based NEST simulation. Please note that it is not our goal to compare two neural network simulators, but to ensure that the anisotropic network implementation on Loihi preserves the main features. For the sake of comparison, we used the same connectivity structure and input positions for both implementations. The networks were initialized at rest and spike patterns were evoked via a spatially-localized input. Raster plots of evoked spike trains indicate that, although the detailed spiking activity is not identical, the overall spiking pattern is mainly preserved (**Figure 3A**). Accordingly, the mean firing rate of the network for each implementation evolves similarly over time (**Figure 3C**).

We confirmed the similarity between both implementations quantitatively, comparing the mean firing rate and firing rate variability over several input and network initializations. **Figure 3D** shows the distribution of mean firing rates over (1) 15 different input positions for the same network connectivity and over (2) 15 different initializations of the network connectivity with a fixed input position. Across input positions in the same network, firing rates for the Loihi implementation were $\bar{f}^{\text{L}}_{\text{inp}} = 0.131 \pm 0.008$ and for the NEST implementation $\bar{f}^{\text{N}}_{\text{inp}} = 0.118 \pm 0.004$. Across network initializations, firing rates were $\bar{f}^{\text{L}}_{\text{init}} = 0.120 \pm 0.013$ for Loihi and $\bar{f}^{\text{L}}_{\text{init}} = 0.113 \pm 0.009$ for NEST. In

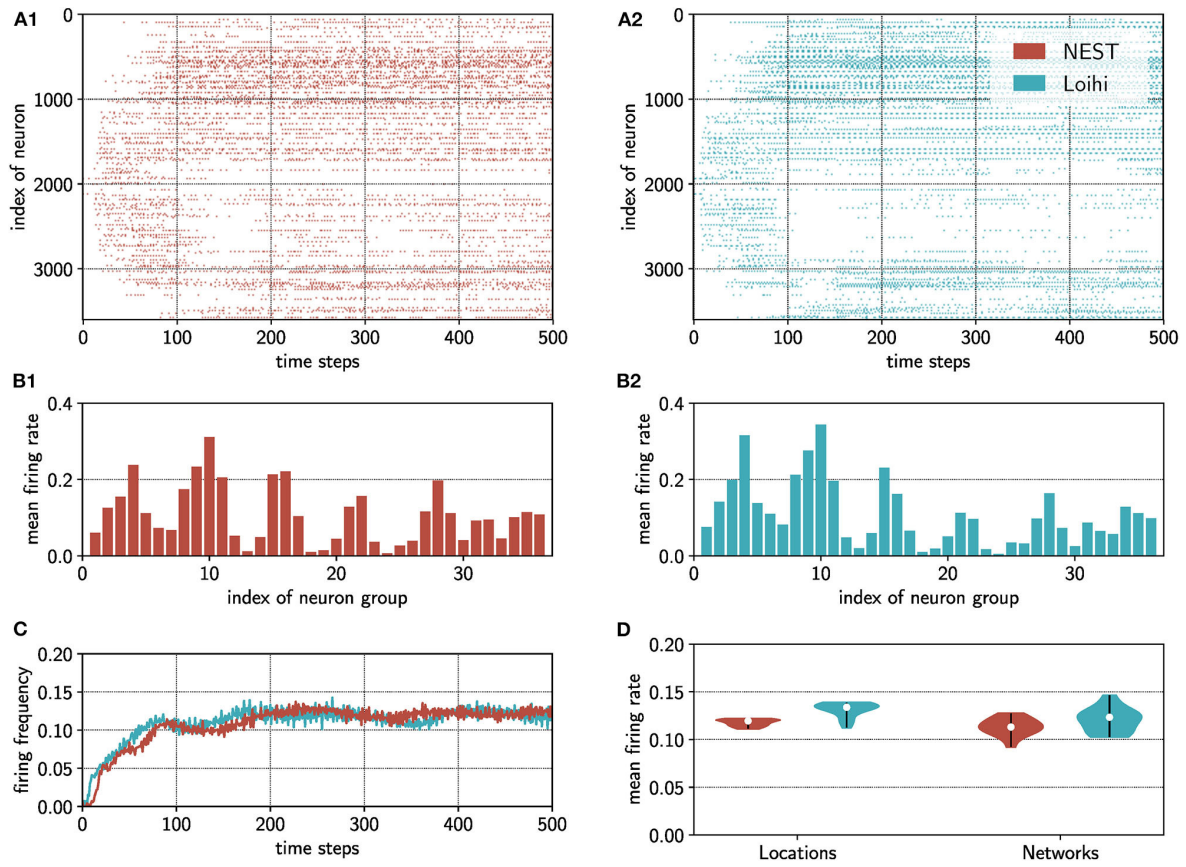


FIGURE 3 | One example of the anisotropic network implementation on Loihi compared to activity statistics of the NEST simulations. **(A)** The spiking activity is shown as spike raster plots. Red: NEST simulation; blue: Loihi implementation. Both networks are initialized with the same weight matrix and triggered with the same input. **(B)** By pooling neurons into groups, each consisting of 100 neurons, and binning across the whole simulation time, we obtain the distribution of mean firing rates (# spikes / # neurons per group) for the NEST and Loihi implementation. **(C)** The time course of the network firing rate (# spikes / # neurons) from the NEST and Loihi simulations is depicted. **(D)** Testing one network initialization with 15 different input locations (Locations) and one location but 15 different network initialization (Networks) for the NEST as well as the Loihi implementation. White dots indicate the median of the mean firing rate (# spikes / # neurons) of the 15 simulations for each case.

addition, the ranges (minimum to maximum mean firing rate) of the obtained mean firing rates are very tight and overlap largely between both implementations. For the locations, the values for Loihi are in a range of $0.11 \leq f^{\mathrm{L}}_{\mathrm{inp}} \leq 0.14$ and for NEST in an interval of $0.11 \leq f^{\mathrm{N}}_{\mathrm{inp}} \leq 0.12$. In case of the different initializations, we obtained mean firing rates between $0.10 \leq f^{\mathrm{L}}_{\mathrm{init}} \leq 0.15$ for Loihi and $0.09 \leq f^{\mathrm{N}}_{\mathrm{init}} \leq 0.13$ for the NEST implementation. To compare the variability of the firing rate in both implementations, we evaluated the Fano factor (FF): For different input positions, we obtained a mean of $\overline{FF}^{\mathrm{L}}_{\mathrm{inp}} = 0.83 \pm 0.03$ for Loihi and $\overline{FF}^{\mathrm{N}}_{\mathrm{inp}} = 0.86 \pm 0.01$ for NEST. In the case of the 15 network initializations, the mean FF for Loihi is $\overline{FF}^{\mathrm{L}}_{\mathrm{init}} = 0.84 \pm 0.02$ and $\overline{FF}^{\mathrm{N}}_{\mathrm{init}} = 0.86 \pm 0.01$ for NEST. All FF values between Loihi and NEST are very close to each other and indicate that spiking is less variable than a Poisson process.

Given that the neural activity in the anisotropic network forms spatially-localized bumps moving through the network, we next measured its average spatial distribution. For the spike rasters shown in **Figure 3A**, we pooled the neurons into groups of 100, taking into account the topology of the network (see **Figure 1B**), and calculated the mean firing frequencies averaged across the whole simulation time. This procedure provides a distribution of the mean activity across the network for both implementations (**Figure 3B**). Normalizing these distributions and comparing them with a Kolmogorov–Smirnov test reveals that the activity distributions from the NEST- and Loihi-based implementations do not differ significantly ($D = 0.11$, $p = 0.97 > 0.05$). Hence the spatial structure of activity patterns is similar in both implementations.

Taken together, we conclude that the Loihi implementation matches the NEST-based anisotropic network implementation according to diverse statistics of the network activity. This

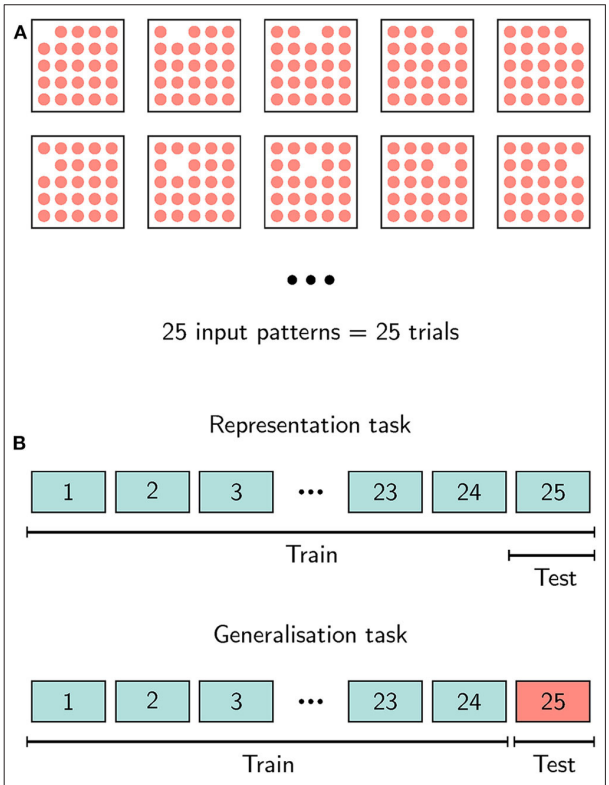


FIGURE 4 | The input protocols for the representation and generalization task. **(A)** A square represents an area of 25 possible input neurons. During a trial, we triggered the network activity by activating 24 out of the 25. This leads to 25 different input patterns and therefore to 25 different trials. **(B)** We used two different training tasks. In the representation task, for the trajectory training, we used the spiking activity of all 25 trials and tested on one of them. In the generalization task, we trained only on the spiking activity of 24 trials and tested on the spikes of a different trial, which was not used for training.

indicates a successful transfer of the core principles of the anisotropic network to neuromorphic hardware despite the differences in architecture.

3.2. The Loihi Implementation of the Anisotropic Network Is Robust to Input Noise

Next, to assess the robustness of the Loihi-based anisotropic network to input noise, we evaluate the stability of spiking dynamics. An input pulse is administered to an area of the anisotropic excitatory layer consisting of 25 neurons (**Figure 1A**). In each trial, 24 of these neurons were activated and a different neuron was systematically excluded from the input, leading to 25 different possible input configurations and, thus, to 25 unique trials (**Figure 4**).

For each trial the network activity was started with a short input pulse of one time step. We then recorded 200 time steps of activity, stopped the activity manually and activated it again by the next input. The protocol is also indicated on top of **Figure 5A**.

As a control, we applied the same protocol to a randomly connected network implemented on Loihi and compared it with the anisotropic network implementation. For this, we implemented the same algorithmic architecture, but exchanged the anisotropic network with a randomly connected network of equal size. The spiking activity of the first three trials is shown in **Figure 5A1** for the anisotropic network (green) and **Figure 5A2** for the randomly connected network (brown). Due to the inhomogeneous connectivity structure, the activity of the anisotropic network spreads out like a stream in the network. Note that, given the torus network topology (see section 2), the activity stream wraps around from neurons with low indices to neurons with high indices (**Figure 5A**). As expected, in the randomly connected network such a stream-like spread of activity does not form.

The population firing rates progress differently in the anisotropic and randomly connected networks. The mean firing rate of the anisotropic network increases slowly until it reaches a relatively constant rate slightly above 0.1. The randomly connected network was tuned such that it generates a similar mean population firing rate (see section 2). However, unlike in the anisotropic network, the firing rate does not rise gradually, but instead starts at about $0.1 - 0.2$ straight away. The slow start in the anisotropic network is due to the relatively small input area and the local connectivity of the network. While moving forward in the 2D-topology, the area of activity grows step by step, which can intuitively be understood as a snowball effect.

In order to measure the stability of the spiking dynamics between different input trials, we calculated the pairwise differences between the spike patterns of all combinations of the 25 trials. The mean and standard deviations of these differences are shown in **Figure 5B** for both the anisotropic network (green) and the randomly connected network (brown). The differences between trials are much higher over the whole time course for the randomly connected network than for the anisotropic network. For the anisotropic network, the deviations of the trial-to-trial differences are very small in the beginning and drift apart over time. To quantify this, we performed a Levene test with three samples at time steps 10, 100, and 190 which revealed that the variance stays constant between the differences of the randomly connected network trials ($W = 0.60$, $p = 0.54 > 0.05$) but increases for the anisotropic network trials over time ($W = 208.87$, $p = 3.36 \cdot 10^{-75} < 0.05$). This means that, over time, spiking patterns between some trials stay very similar whereas some trial comparisons tend to differ more. Therefore, the anisotropic network tends to slowly diverge with time, which can also be seen by the increasing mean differences. Importantly the mean differences in the anisotropic network remain much lower than the spiking differences between the trials in the randomly connected network, even at the end of the 200 time steps. This clearly demonstrates the stabilizing feature of the anisotropic network.

To visualize differences between the single trials, we reduced the dimensionality of the spiking data by applying principal component analysis (PCA) to all trials (see section 2). The

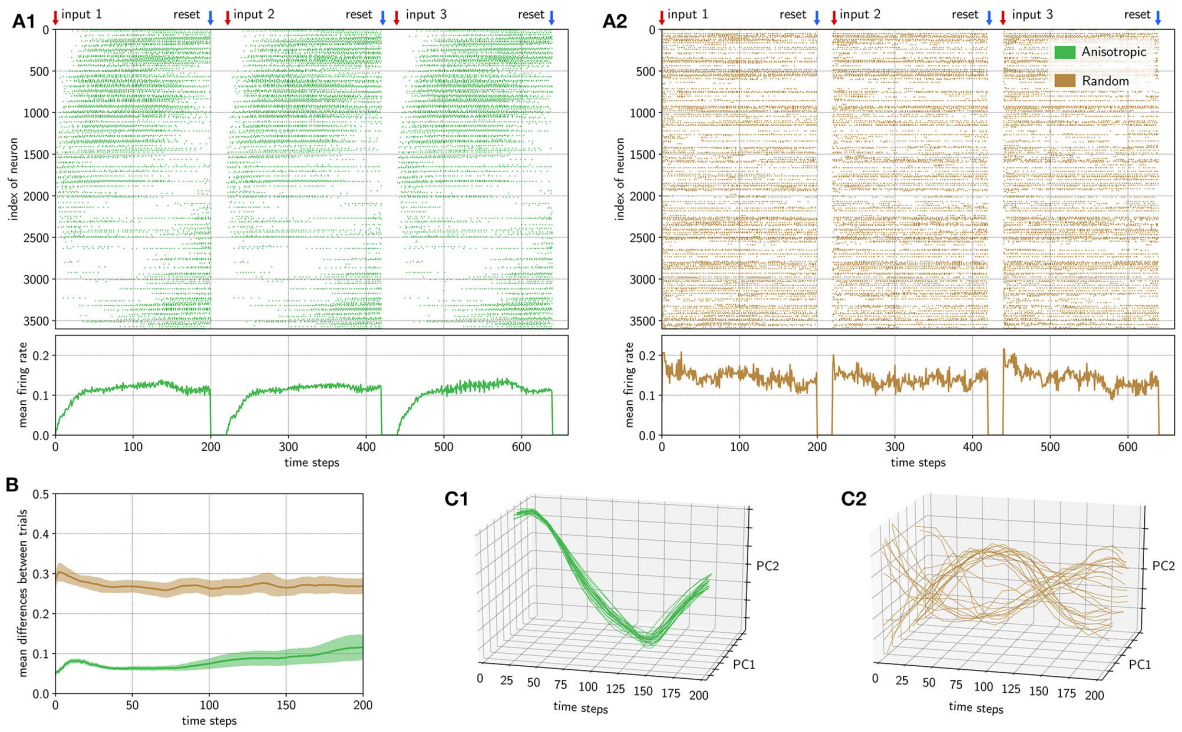


FIGURE 5 | The Loihi implementation of the anisotropic network is robust to varying input conditions, while a randomly connected network is not. **(A)** Three examplary trials out of the 25 trials are shown for a simulation based on an anisotropic connectivity structure (green) and with randomly initialized weights (brown). The anisotropic structure clearly has a stream-like spiking pattern, where the firing rate starts slowly until it reaches a constant rate. The randomly connected network shows a Poisson-like spiking pattern, where the firing rate starts directly at a high level. **(B)** The solid lines show the mean difference between all trial combinations and the shaded area indicates the standard deviation of the trial-to-trial differences for the anisotropic (green) and random (brown) network. **(C)** In the reduced space of the first two principal components of the activity of all 25 trials over time for both networks, we can clearly see that the spiking pattern of the anisotropic network are very similar between trials, while the activity in the randomly connected network differ much more between trials.

results are shown in **Figure 5C**. For the anisotropic network (**Figure 5C1**), all trajectories are very similar whereas for the randomly connected network (**Figure 5C2**) the trajectories differ considerably. We quantified this by calculating statistics in the first dimension of the PCA space. First, we obtained the pairwise normalized mean squared error between all trials for each network type. The normalized mean error between the trials of the randomly connected network is $\mathrm{MSE}_{\mathrm{rand}} = 1.66$, while the anisotropic network has a mean error of only $\mathrm{MSE}_{\mathrm{aniso}} = 0.03$, which is significantly lower (Mann–Whitney U-test: $U = 3572.0$, $p = 4.27 \cdot 10^{-85} < 0.05$). Even though some trajectories seem to follow a common path, in the random network, the mean standard deviation for the first principle component $\bar{\sigma}_{\mathrm{rand}} = 2.50$ is significantly higher than in the anisotropic network with $\bar{\sigma}_{\mathrm{aniso}} = 0.41$ (Mann–Whitney U-test: $U = 0.0, p = 1.56 \cdot 10^{-8} < 0.05$). This indicates sufficient stability over 200 time steps for the anisotropic network.

Taken together, this shows the ability of the anisotropic network to produce stable spiking dynamics under noisy input conditions. In addition it confirms the successful implementation of the network on Loihi. In the next step we will use this intrinsic stability feature of the anisotropic network to learn robust trajectories and examine if our network produces sufficient variability to learn arbitrary functions.

3.3. Learning Robust Trajectories

After having tested and demonstrated the stabilizing feature of the anisotropic network, we aimed to use its robustness to train arbitrary output trajectories. This step makes use of the underlying network architecture shown in **Figure 1** and adds a linear regression model on top of this architecture for a robot control task. The overall algorithm contains the initialization, creation and simulation of the anisotropic network, which is running on the neuromorphic hardware Loihi, and the output learning of the trajectories, which is calculated on the host CPU.

To show the robustness of this algorithm, we learned 7 different 3D-trajectories commonly used in robotic research, like pick-and-place or put-on-top (see section 2). Using these target functions, we applied two different tasks, a representation and a generalization task, as shown in **Figure 4B**. In the representation task we estimated the linear regression model based on all 25 trials and predicted one of them, showing that

the variability in the anisotropic network is sufficient to learn an arbitrary function. To show the ability of our algorithm to robustly generalize for variations in the input, we also apply a generalization task, where we estimate the regression model on 24 trials and predicted the trajectory for an unseen trial.

As before, here we also compare the performance of the anisotropic network with the randomly connected network as a control. For our algorithm we estimate our model based on the 72 pooling layer neurons, which we can read out efficiently from the chip. Since we reduce the parameter space of the linear regression model by using only the spiking activity of the pooling layer neurons as data, we also estimated all models based on the 3,600 excitatory reservoir neurons for comparison.

Results for the representation task, based on all excitatory neurons, revealed that the excitatory reservoir neurons contain enough variability to represent an arbitrary output function with high accuracy. An example is shown in **Supplementary Figure 1A1** for the randomly connected network and in **Supplementary Figure 1A2** for the anisotropic network. If the model was estimated on the 72 pooling layer neurons the number of available parameters is heavily decreased by a factor of 50. But still the amount of information seems to be satisfactory for the anisotropic network (**Supplementary Figure 1A4**), but not for the randomly connected case (**Supplementary Figure 1A3**). Normalized root mean squared error between the predicted trajectory and the target trajectory, averaged over 7 3D-trajectories, are shown in **Figure 6A1**. The errors for all trajectories using the spiking activity of the 3,600 excitatory reservoir neurons are very low (left plot) for both networks, but interestingly even lower for the anisotropic network. For the errors of the estimation based on the pooling layer neurons (right plot in **Figure 6A1**), the mean error over all trajectories is still low for the anisotropic network $e_{\text{aniso}} = 0.02 \pm 0.003$, compared to the randomly connected network $e_{\text{rand}} = 0.33 \pm 0.07$. Due to the inhomogeneous weight structure and the stream-like spread of spiking activity in the anisotropic network, the neurons in the pooling layer can maintain variability, as can be seen in **Supplementary Figure 1B1**. Intuitively, since nearby neurons have correlated activity patterns, pooling over them preserves information. In contrast, as shown in **Supplementary Figure 2B2**, the spiking activity in the pooling layer of the randomly connected network simply produces downsampled random spiking activity and therefore reduced variability.

In the generalization task, the parameters for the movement trajectory were estimated based on the spiking activity of 24 trials. We then predicted the same trajectory based on the spiking activity elicited by a 25th trial, not seen during training. This task was designed to test the robustness of the system to a variation in initial conditions. To compare the classical reservoir computing approach with our network architecture, we trained the network based on all 3,600 neurons and on the 72 output neurons. In addition, this tested the ability of the pooling neurons to preserve sufficient variability while reducing the number of parameters.

For the full network read-out, we applied a linear regression model based on all excitatory neurons of the anisotropic network. Since fitting a model based on all 3,600 neurons requires many parameters, here we used an elastic net regularization estimation method (see section 2) to reduce the number of parameters and to avoid overfitting. Optimizing the regularization parameters resulted in $\alpha = 0.001$ and $\lambda = 0.05$. For the pooling layer read-out, we estimated a linear regression model based on the pooling layer neurons without regularization.

In **Figure 6A2**, we show the average normalized root-mean-squared deviation over 7 trajectories. In both cases (using excitatory neurons or pooling layer neurons) the error of the anisotropic network is much lower, showing that the anisotropic network has a better performance compared to a classical randomly connected network.

For the network architecture with the randomly connected network, the elastic net approach, based on all excitatory neurons, has a better performance than the linear regression approach, based on the pool neurons (t-test: $t = -4.24$, $p = 0.0001 < 0.05$). Interestingly, the error for the anisotropic network is lower when the trajectories are estimated based on the pool neurons compared to the excitatory neurons (Mann–Whitney U-test: $U = 47.0$, $p = 6.75 \cdot 10^{-6} < 0.05$). This indicates that, for the anisotropic network, the pooling layer is an equivalent, or even better regularization method compared to the elastic net approach with all excitatory neurons.

Figure 6B shows all three dimensions of the predicted trajectory over time for a *hide* movement. The overall trajectory is shown in **Figure 6C**. We also calculated a smoothed version, using a Savitzky–Golay filter (Savitzky and Golay, 1964) (see section 2), to better compare the prediction with the target. This shows that the anisotropic network implemented on Loihi, combined with the pooling layer, contains sufficient variability to represent complex 3D trajectories while at the same time remaining stable for at least 200 time steps.

3.4. Simulation on Loihi in Real-Time

In addition to evaluating the stability of the system, we also looked at the speed of the network simulation. The data we used came from a Kuka robot arm, which can run fluently with an output frequency of 100 Hz, therefore the 200 time steps equal 2 s of movement. In the following we denote this reference as "real-time." To achieve a real-time output of spiking data from the Loihi chip, the speed of the simulation of the neurons and the data transfer from the chip to the host must be higher than the necessary data frequency of the robot for a smooth movement.

The simulation of these 200 time steps requires $t^{\text{aniso}}_{3,600} = 15.73\,s$ for one trial on Loihi, when all 3,600 excitatory reservoir neurons were read out from the system. This speed is about 8 times slower than real-time. When reading out only from the 72 pooling neurons, the speed increases to $t^{\text{aniso}}_{72} = 1.49\,s$ per trial, which is 25% faster than real-time and therefore well-suited for robot control.

The simulation speed of the anisotropic network ($t^{\text{aniso}}_{3,600} = 15.73\,s$ & $t^{\text{aniso}}_{72} = 1.49\,s$ per trial) and the randomly connected network ($t^{\text{rand}}_{3,600} = 16.11\,s$ & $t^{\text{rand}}_{72} = 1.73$ per trial) were nearly the same, which is expected since the number of neurons is the same and the number of synapses is similar. Thus, the anisotropic

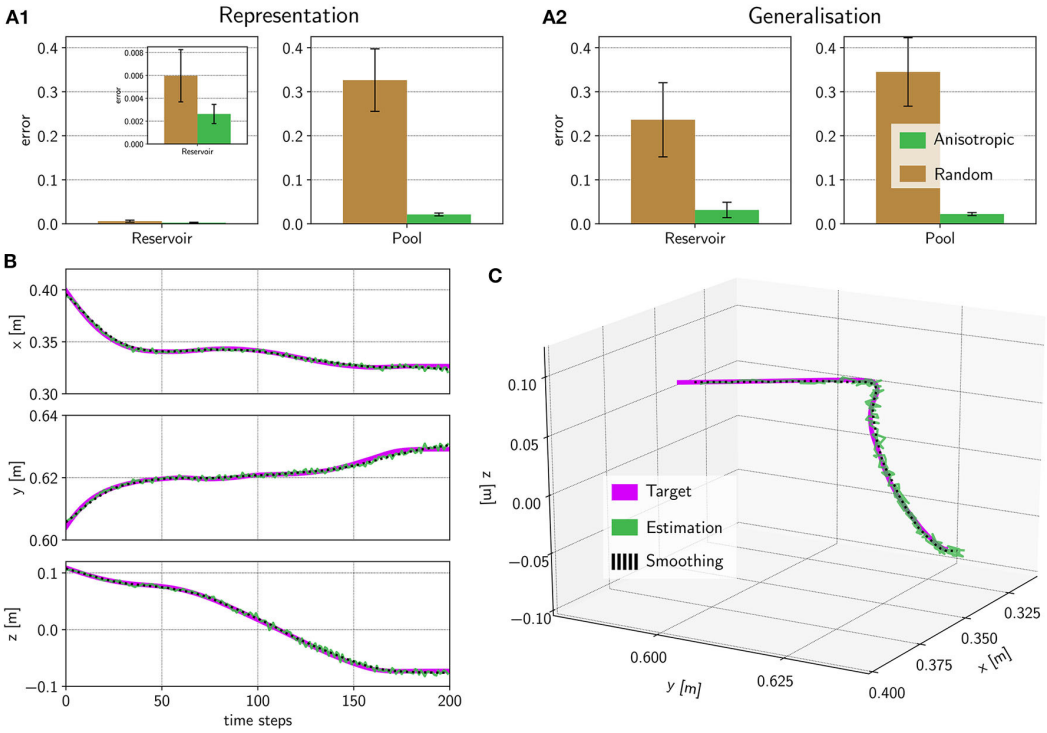


FIGURE 6 | Predicting a trajectory on the spiking activity. **(A)** We predicted 7 different trajectories with 3 dimensions each and calculated the error regarding the target trajectory using a normalized root-mean-square error. In all cases, the anisotropic network has a lower error than the randomly connected network. **(B)** For the *hide* movement, we show one example of the predicted trajectory of the anisotropic network for each of the 3 dimensions. Here, we used the pooling neurons (green) in comparison with the target (pink) and a smoothed variant of the prediction (black dotted). The whole trajectory is shown in **(C)**.

network has, in terms of speed, no disadvantage compared to the randomly connected network.

Therefore, the pooling layer does not only reduce the sensitivity of the system but also helps to speed up the system considerably. Together, this supports robotic applications where trajectories can be stored and replayed robustly in real-time.

4. DISCUSSION

We aimed to develop an algorithm for neuromorphic hardware, which provides stable spiking dynamics under noisy input conditions, in order to make use of the low power neuromorphic chips for future autonomous systems. For this, we derived an algorithm to store and control robotic movement sequences that unfold on a control-relevant timescale of seconds. To validate our approach, we chose a set of 2-s-long robot arm movements that were triggered by noisy inputs.

For our approach we chose a recently developed spiking neural network (Spreizer et al., 2019) with an inhomogeneous weight structure. In a first step, we successfully transferred the main principles of this network to the Loihi research chip from Intel (Davies et al., 2018), a neuromorphic hardware architecture implementing spiking neurons. In a second step, we tested the stability of the anisotropic network implementation and compared its stability to a classical randomly connected network, similar to echo state networks (Jaeger, 2001, 2007) or liquid state machines (Maass et al., 2002). We finally used a pooling layer (**Figure 1**) to efficiently read out spiking data from the chip. Using these spiking data we were able to learn 3D trajectories in a noise-robust way (**Figure 6C**). The pooling layer successfully increased the simulation speed to faster than real-time. It was also intended to make the spiking activity more invariant to small changes in the network, which is the exact purpose of using pooling layers in deep neural networks (Goodfellow et al., 2016, Chapter 9.3; Boureau et al., 2010). A pooling layer has been applied to spiking neural networks before (Tavanaei and Maida, 2017; Tavanaei et al., 2019), but – to the best of our knowledge – such a structure has never been applied to enhance the performance of read-outs from recurrent network architectures. The fact that the pooling layer improved performance for the anisotropic network in our study indicates that implementing pooling layers in reservoir computing architectures could be useful in other cases, for example when the reservoir has spatially-dependent connectivity (Maass et al., 2002), and especially for reducing parameters on algorithms running on neuromorphic hardware.

Taken together, in this study we provide an algorithm for storing stable trajectories in spiking neural networks, optimized

for the neuromorphic hardware Loihi. The network architecture is capable of executing these trajectories on demand in real-time given noisy, and even never-before-seen, inputs. While an exhaustive exploration of the parameter space remains the subject of future work, we have shown that the anisotropic network admits stable sequences with sufficient variability for output learning across hundreds of milliseconds, making it suitable for applications reaching far beyond motor control. Further, we demonstrated that spike-based pooling can implement on-chip regularization for the ansiotropic network, improving read out speed and accuracy. In contrast, in the randomly connected network nearby neurons show uncorrelated activity and spatial pooling has no benefit. Thus, spatial pooling in locally-connected SNNs proved to be a promising feature, specifically for real-time robotic control on neuromorphic hardware. Importantly, we provide the first neuromorphic implementation which has no global learning or adaptation mechanism and produces noise-robust spiking patterns on a control-relevant timescale with sufficient variability to learn arbitrary functions.

While other approaches employing spiking neural networks exist, in general they fail to meet at least one of the mentioned criteria. This means, in their current form, these models are either not implementable on neuromorphic hardware or do not produce sequences that are stable, variable and long enough. We briefly describe these models and highlight how they may be adapted for neuromorphic implementation.

Laje and Buonomano (2013) presented an "innate training" approach. The network was initialized with a short input pulse and a modified FORCE algorithm (Sussillo and Abbott, 2009) was used to train the recurrent connections. This stabilizes the innate structure of the recurrent connections and allows a network state between chaotic and locally stable activity patterns. A trained output trajectory was robust to perturbations, due to the tuned recurrent weights. Unfortunately this algorithm uses a rate coded network and non-local learning rules, both of which are not applicable for most neuromorphic systems.

Pehlevan et al. (2018) analyzed different approaches to solve the stability-variability trade-off in the context of songbird songs. One additional and important criterion for their evaluation was the ability of an algorithm to provide temporal flexibility, such that outputs can be replayed faster or slower. They concluded that a synfire chain model fits best to solve this task. While this approach seems to model the dynamics underlying songbird songs with flexible timing, synfire chains have a feed-forward structure which makes them less flexible than recurrent network types.

Hennequin et al. (2014) put more focus on getting stable output from unstable initial conditions. They used an optimization algorithm to build an inhibitory structure that helps to stabilize the excitatory activity. More precisely, the strength of existing inhibitory connections was changed or new inhibitory synapses were created or removed using an algorithm based on a relaxation of the spectral abscissa of the weight matrix (Vanbiervliet et al., 2009). With this they obtained relatively stable spiking dynamics. Interestingly, this approach is similar to our study in a sense that both approaches focus on the weight matrix. While their proposed solution to the stability-variability trade-off is promising, so far the algorithm has mainly been tested with rate coded networks. A more elaborate analysis with a spiking neural network would be of interest.

Another recent approach involves multiplexing oscillations in a spiking neural network (Miall, 1989; Vincent-Lamarre et al., 2020). Two input units inject sine-waves into a reservoir of neurons and the spiking dynamics in the reservoir follow a stable and unique pattern, which enables the learning of a long and stable output. Compared to our algorithm, the oscillating units provide a continuous input to the network. We see this approach as a potential alternative to the anisotropic network for robotic control. Interestingly, stability is encoded in time rather than space, which raises the question whether this approach could be combined with a pooling layer, reflecting temporal structure instead of spatial structure.

Maes et al. (2020) trained a recurrently connected spiking network such that small groups of neurons become active in succession and thus provide the basis for a simple index code. Via a supervisor signal, output neurons are trained to become responsive to a particular group or index from the recurrent network and, thus, fire in a temporal order encoded in the feed-forward weights to the output layer. Importantly, learning within the recurrent network and from the recurrent network to the output layer is done using spike-timing dependent plasticity. However, as is, their implementation has a few small, but likely reconcilable, incompatibilities with the neuromorphic hardware considered here. For example, learning and synaptic normalization is only local to the neuron, and not to the synapse and they rely on adaptive exponential integrate and fire neurons, which are not implemented by Loihi. With some modifications, their model may provide another neuromorphically implementable approach.

While our approach provides an algorithm for storing stable trajectories, our two-chip Loihi system is limited in the number of neurons available, constrained mainly by the high number of synapses in our recurrent network. Since this limitation is mainly caused by the current `NxSDK` software and not by hardware, we expect an improvement in upcoming releases. With more neurons available we expect even better stability, reducing the last remaining variations in our predictions and allowing even longer movement actions, beyond 2 s. At this point, further investigation of how performance depends on network size, network parameters, and pooling layer configuration will be of interest.

With more neurons available, one could add multiple inputs to the network. We hypothesize that nearby input locations lead to similar activity patterns, while input regions far from each other produce distinct activity patterns. This behavior could be used to train multiple trajectories from different input locations. With this, more complex robotic control tasks could be performed, beyond the generation of single trajectories.

One general hurdle in developing neuromorphic implementations is the difficulty in transferring existing spiking neural network models from CPU-based implementations to neuromorphic hardware. As outlined in the section 2, Loihi provides a fixed hardware-implemented neuron model. It is possible to adjust parameters, but not the neuron model itself. Therefore, a perfect match between traditional simulators like `NEST` (Gewaltig and Diesmann, 2007) or `Brian2` (Stimberg

et al., 2019) and neuromorphic hardware, like Loihi, is in general an issue for future neuromorphic algorithms. Efficient methods for translating neuroscientific models to Loihi is the subject of current work.

Finally, to complete the algorithm for autonomous use cases, in which Loihi is able to control a robot independently, an on-chip output learning algorithm is vital. This requires the implementation of an output neuron on the chip, with appropriate on-chip output weights. It is already possible to train weights offline and transfer them to Loihi, applied for example in `Nengo Loihi` (Bekolay et al., 2014; Hampo et al., 2020). We expect that the on-chip regularization inherent in spatial pooling will improve the robustness of future online output learning algorithms.

5. CONCLUSION

Taken together, we developed an algorithm which can serve as a basic unit in robotic applications. The anisotropic network structure offers stability against noisy inputs and the overall architecture, especially using the pooling layer, paves the way for further steps in the development of algorithms for neuromorphic hardware. Our study proposes an algorithm based on *intrinsic* self-stabilizing features of a well-initialized anisotropic connectivity structure, which can overcome the instability problem of spiking neural networks and support robust outputs on a timescale of seconds.

DATA AVAILABILITY STATEMENT

The PeleNet framework for Loihi, which was written for this study, can be found on GitHub (https://github.com/sagacitysite/pelenet/tree/neurorobotics). The data that support the findings of this study are available from the corresponding author on request.

AUTHOR CONTRIBUTIONS

AL contributed the network simulations. CM contributed the Loihi implementation, including the Pelenet framework. CT acquired funding and supervised the study. All authors designed the study and reviewed the manuscript.

FUNDING

The research was funded by the H2020-FETPROACT project Plan4Act (#732266) [CM, AL, CT], by the German Research Foundation (#419866478) [AL, CT], and by the Intel Corporation via a gift without restrictions.

ACKNOWLEDGMENTS

The authors are thankful to Osman Kaya for providing Kuka robot trajectories, to Arvind Kumar and Lukas Ruff for helpful discussions, and to Tristan Stöber for improving the text. All of them helped to improve the quality of this work. Furthermore, we thank the Intel Corporation for providing access to their Loihi chip.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2020.589532/full#supplementary-material

REFERENCES

Amit, D. J. (1992). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge: Cambridge University Press.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform.* 7, 1–13. doi: 10.3389/fninf.2013.00048

Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2019). "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop* (New York, NY; Albany, NY: Association for Computing Machinery), 1:8. doi: 10.1145/3320288.3320304

Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Madison, WI: Omnipress), 111–118.

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *J. Physiol.* 94, 445–463. doi: 10.1016/S0928-4257(00)01084-6

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

DeWolf, T., Stewart, T. C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proc. R. Soc. B Biol. Sci.* 283:20162134. doi: 10.1098/rspb.2016.2134

Eliasmith, C., and Anderson, C. H. (2004). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press.

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press, 330–334.

Hampo, M., Fan, D., Jenkins, T., DeMange, A., Westberg, S., Bihl, T., et al. (2020). "Associative memory in spiking neural network form implemented on neuromorphic hardware," in *International Conference on Neuromorphic Systems 2020*, 1–8. doi: 10.1145/3407197.3407602

Hennequin, G., Vogels, T. P., and Gerstner, W. (2014). Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron* 82, 1394–1406. doi: 10.1016/j.neuron.2014.04.045

Hutt, A. (2008). Local excitation-lateral inhibition interaction yields oscillatory instabilities in nonlocally interacting systems involving finite propagation delay. *Phys. Lett. A* 372, 541–546. doi: 10.1016/j.physleta.2007.08.018

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, Vol. 2 (Washington, DC: IEEE), 1398–1403. doi: 10.1109/ROBOT.2002.1014739

Itskov, V., Curto, C., Pastalkova, E., and Buzsáki, G. (2011). Cell assembly sequences arising from spike threshold adaptation keep

track of time in the hippocampus. *J. Neurosci.* 31, 2828–2834. doi: 10.1523/JNEUROSCI.3773-10.2011

Jaeger, H. (2001). *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks-With an Erratum Note*. German National Research Center for Information Technology, Bonn. GMD Technical Report.

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

Khalastchi, E., Kaminka, G. A., Kalech, M., and Lin, R. (2011). "Online anomaly detection in unmanned vehicles," in *The 10th International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC; Taipei: International Foundation for Autonomous Agents and Multiagent Systems), 115–122.

Laje, R., and Buonomano, D. V. (2013). Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nat. Neurosci.* 16:925. doi: 10.1038/nn.3405

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., et al. (2018). Programming spiking neural networks on intel's Loihi. *Computer* 51, 52–61. doi: 10.1109/MC.2018.157113521

London, M., Roth, A., Beeren, L., Häusser, M., and Latham, P. E. (2010). Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature* 466, 123–127. doi: 10.1038/nature09086

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Maes, A., Barahona, M., and Clopath, C. (2020). Learning spatiotemporal signals using a recurrent spiking network that discretizes time. *PLoS Comput. Biol.* 16:e1007606. doi: 10.1371/journal.pcbi.1007606

Miall, C. (1989). The storage of time intervals using oscillating neurons. *Neural Comput.* 1, 359–371. doi: 10.1162/neco.1989.1.3.359

Murray, J. M. et al. (2017). Learning multiple variable-speed sequences in striatum via cortical tutoring. *eLife* 6:e26084. doi: 10.7554/eLife.26084

Nachstedt, T., and Tetzlaff, C. (2017). Working memory requires a combination of transient and attractor-dominated dynamics to process unreliably timed inputs. *Sci. Rep.* 7, 1–14. doi: 10.1038/s41598-017-02471-z

Naseer, T., Burgard, W., and Stachniss, C. (2018). Robust visual localization across seasons. *IEEE Trans. Robot.* 34, 289–302. doi: 10.1109/TRO.2017.2788045

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., et al. (2018). Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proc. IEEE* 107, 144–164. doi: 10.1109/JPROC.2018.2881432

Pehlevan, C., Ali, F., and Ölveczky, B. P. (2018). Flexibility in motor timing constrains the topology and dynamics of pattern generator circuits. *Nat. Commun.* 9, 1–15. doi: 10.1038/s41467-018-03261-5

Perlin, K. (1985). An image synthesizer. *SIGGRAPH Comput. Graph.* 19, 287–296. doi: 10.1145/325165.325247

Roxin, A., Brunel, N., and Hansel, D. (2005). Role of delays in shaping spatiotemporal dynamics of neuronal activity in large networks. *Phys. Rev. Lett.* 94:238103. doi: 10.1103/PhysRevLett.94.238103

Savitzky, A., and Golay, M. J. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.* 36, 1627–1639. doi: 10.1021/ac60214a047

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modelling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Sompolinsky, H., Crisanti, A., and Sommers, H.-J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61:259. doi: 10.1103/PhysRevLett.61.259

Spreizer, S., Aertsen, A., and Kumar, A. (2019). From space to time: spatial inhomogeneities lead to the emergence of spatiotemporal sequences in spiking neuronal networks. *PLoS Comput. Biol.* 15:e1007432. doi: 10.1371/journal.pcbi.1007432

Spreizer, S., Angelhuber, M., Bahuguna, J., Aertsen, A., and Kumar, A. (2017). Activity dynamics and signal representation in a striatal network model with distance-dependent connectivity. *eNeuro* 4. doi: 10.1523/ENEURO.0348-16.2017

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

Sussillo, D., and Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron* 63, 544–557. doi: 10.1016/j.neuron.2009.07.018

Tang, G., Shah, A., and Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *arXiv preprint arXiv:1903.02504*. doi: 10.1109/IROS40897.2019.8967864

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Tavanaei, A., and Maida, A. (2017). "Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals," in *Neural Information Processing*, eds L. Derong , X. Shengli, L. Yuanqing, Z. Dongbin,and E.-A. El-Sayed (Cham: Springer International Publishing), 899–908. doi: 10.1007/978-3-319-70136-3_95

Tsodyks, M. (1999). Attractor neural network models of spatial maps in hippocampus. *Hippocampus* 9, 481–489. doi: 10.1002/(SICI)1098-1063(1999)9:4<481::AID-HIPO14>3.0.CO;2-S

Van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi: 10.1126/science.274.5293.1724

Vanbiervliet, J., Vandereycken, B., Michiels, W., Vandewalle, S., and Diehl, M. (2009). The smoothed spectral abscissa for robust stability optimization. *SIAM J. Optimizat.* 20, 156–171. doi: 10.1137/070704034

Vincent-Lamarre, P., Calderini, M., and Thivierge, J.-P. (2020). Learning long temporal sequences in spiking networks by multiplexing neural oscillations. *Front. Comput. Neurosci.* 14:78. doi: 10.3389/fncom.2020.00078

Voelker, A. R., and Eliasmith, C. (2017). Methods for applying the neural engineering framework to neuromorphic hardware. *arXiv [Preprint]*. arXiv:1708.08133.

Wörgötter, F., Ziaeetabar, F., Pfeiffer, S., Kaya, O., Kulvicius, T., and Tamosiunaite, M. (2020). Humans predict action using grammar-like structures. *Sci. Rep.* 10, 1–11. doi: 10.1038/s41598-020-60923-5

York, L. C., and Van Rossum, M. C. (2009). Recurrent networks with short term synaptic depression. *J. Comput. Neurosci.* 27:607. doi: 10.1007/s10827-009-0172-4

Zheng, P., and Triesch, J. (2014). Robust development of synfire chains from multiple plasticity mechanisms. *Front. Comput. Neurosci.* 8:66. doi: 10.3389/fncom.2014.00066

Zou, H., and Hastie, T. (2005). Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B* 67, 301–320. doi: 10.1111/j.1467-9868.2005.00503.x

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers

Supplementary Material

1 SUPPLEMENTARY TABLES AND FIGURES

1.1 Figures

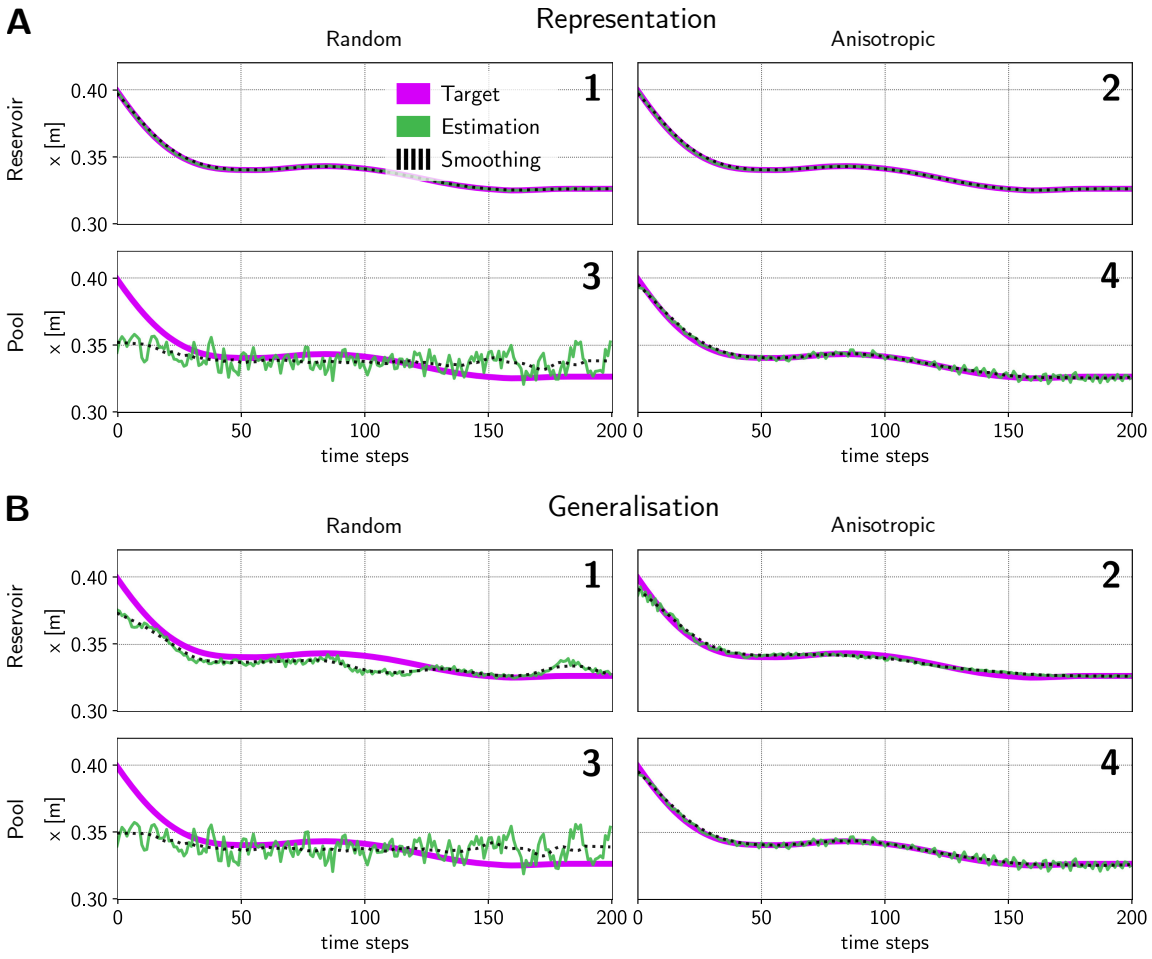


Figure S1. A single trajectory estimation for the x-dimension for all different tasks (representation & generalisation), networks (randomly connected network & anisotropic network) and estimation methods (excitatory reservoir neurons with elastic net regularization & pooling layer neurons).

Supplementary Material

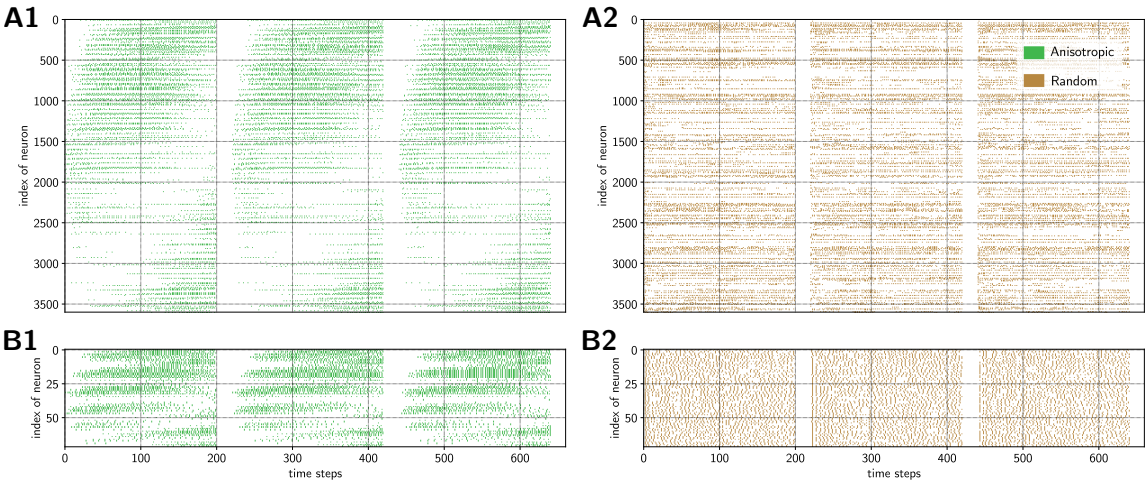


Figure S2. Spike trains of the excitatory reservoir neurons **A** compared with the spike trains of the pooling layer neurons **B**. In the anisotropic network (green) the stream-like structure of the excitatory reservoir neurons are reflected in the pooling layer.

1.2 Tables

Parameter		NEST	Loihi
temporal resolution	dt	$0.1\,ms$	N/A
excitatory neurons	$npop_E$	3600	3600
inhibitory neurons	$npop_I$	900	900
membrane capacitance	C_m	$250.0\,pF$	N/A
leak conductance	g_L	$25.0\,nS$	N/A
threshold potential	v_{th}	$-55.0\,mV$	64000
resting potential	E_L	$-70.0\,mV$	0
reset potential	v_{reset}	$-70.0\,mV$	0
refractory period	t_{ref}	$2.0\,ms$	2
synaptic time constant (exc.)	τ_{exc}	$5.0\,ms$	N/A
synaptic time constant (inh.)	τ_{inh}	$5.0\,ms$	N/A
current decay	τ_I	N/A	380
voltage decay	τ_v	N/A	400
synaptic delay	d	$1.0\,ms$	1
synaptic weights (excitatory)	J^{exc}	$40\,pA$	12
synaptic weights (inhibitory)	J^{inh}	$-160\,pA$	48
connection probability	p_{conn}	0.05	0.05
perlin scale	κ_{perlin}	4	4
gaussian sigma (exc.)	σ_E	12	12
gaussian sigma (inh.)	σ_I	9	9
shift magnitude	n_{shift}	1	1

Table S1. Comparison of parameters used for the `NEST` and the Loihi simulation. Both implementations use leaky integrate-and-fire neurons with current-based synapses. The `NEST` model has an additional alpha-function shaped synaptic current rise, which is not available on Loihi.

PeleNet: A Reservoir Computing Framework for Loihi

Carlo Michaelis

24. November 2020

Note: This is a draft

Abstract

High-level frameworks for spiking neural networks are a key factor for fast prototyping and efficient development of complex algorithms. Such frameworks have emerged in the last years for traditional computers, but programming neuromorphic hardware is still a challenge. Often low level programming with knowledge about the hardware of the neuromorphic chip is required. The PeleNet framework aims to simplify reservoir computing for the neuromorphic hardware Loihi. It is build on top of the NxSDK from Intel and is written in Python. The framework manages weight matrices, parameters and probes. In particular, it provides an automatic and efficient distribution of networks over several cores and chips. With this, the user is not confronted with technical details and can concentrate on experiments.

Introduction

Several different neuromorphic hardware chips have been developed in recent years (reviewed by Schuman et al., 2017; Young et al., 2019; Rajendran et al., 2019). All of them promise to be a key factor in future neuroscientific research as well as technological developments in artificial intelligence. The main benefit of neuromorphic systems is their low power consumption and speed (Rajendran et al., 2019). Shown for Loihi for example from Tang et al. (2019). This advantage of brain inspired hardware comes with a solution to the von Neumann bottleneck (Backus, 1978). While novel neuromorphic hardware becomes more and more powerful, algorithms for such hardware systems are still in an early stage. A specific field of spiking neural network algorithms, which can be used for neuromorphic hardware, is reservoir computing. For details about reservoir computing I refer the reader to the literature (Jaeger, 2001; Maass et al., 2002; Jaeger, 2007; Schrauwen et al., 2007; Lukoševičius et al., 2012; Goodfellow et al., 2016).

Here, I focus on the neuromorphic hardware chip Loihi (Davies et al., 2018). The chip is digital and includes a current-based (CUBA) leaky integrate-and-fire (LIF) neuron. A chip contains 128 cores and each core time-multiplexes 1024 compartments. In addition, every chip contains three conventional x86 CPUs. Several chips can be used in parallel on one board. The parameters of the synapses and the compartments can be adapted, but the neuron model itself is fixed. A single compartment neuron can be extended to a multi compartment neuron which comes with the cost of less available neurons. Intel provides a software development kit, the so-called `NxSDK` which is written in `Python` and already allows higher-level programming of the chip (Lin et al., 2018). In addition, `C` scripts can be used to run code on the x86 CPUs of the chip. The `NxSDK` allows to define *compartment prototypes* and *compartment groups* which can be combined to *neuron prototypes* and *neurons*. Using *connection prototypes* and *connections* these compartments and neurons can be interconnected using a connection matrix. Spikes can be injected

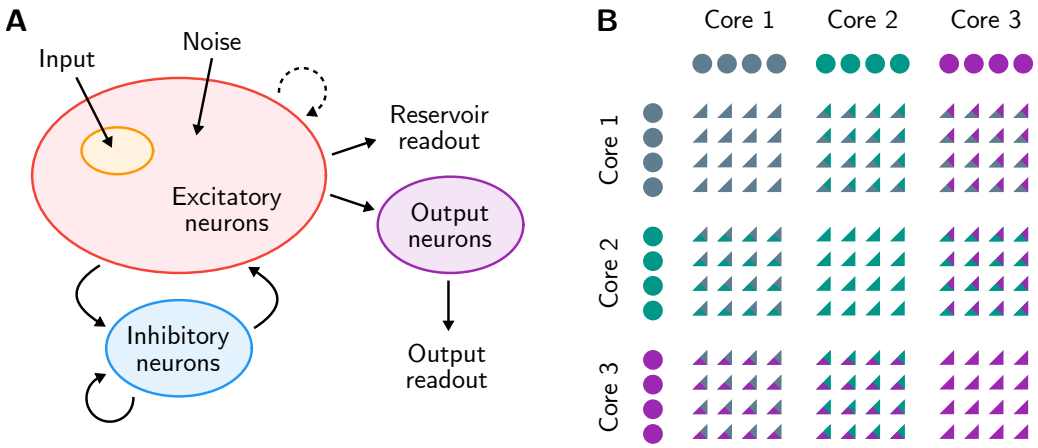


Figure 1: **(A)** The reservoir network consists of a pool of excitatory neurons (red) and a pool of inhibitory neurons (blue). Those pools are connected within and in-between. The excitatory neurons can be stimulated by an input (orange) and/or noise. Optionally a pool of output neurons (purple) can be defined. The spiking data can be read out from the reservoir itself and/or from the output neurons. `PeleNet` supports configuring a learning rule for the connections of the excitatory neurons (dotted arrow). **(B)** The example shows 12 neurons which are distributed to three cores, where each core contains four neurons. Each core is color coded (grey, green, purple). Every triangle symbolises a potential connection between two neurons. The color of the triangle indicates which cores need to be interconnected to interconnect the related neurons. In this case, neurons spread over three cores require 9 connection matrices.

using *spike generators* and several different types of *probes* can be defined. Finally, *learning rules* can be used to make connections in the reservoir plastic.

However, the `NxSDK` defines compartments for each core separately. If bigger networks are used, it is necessary to split the connections manually to the Loihi cores. Compartments between cores need then to be interconnected manually, which results in $n_{\text{conn}} = n_{\text{core}}^2$ connection matrices, if all potential connections should be possible. For example, if we create 3 compartment groups, distributed to $n_{\text{core}} = 3$ cores, and we want to interconnect all of them, we need to define $n_{\text{conn}} = 9$ connection weight matrices, which is illustrated in Figure 1B. Note that this amount of matrices is necessary even for sparse reservoirs networks, since we do not want to exclude any possibility a priori. In addition to handling these connection matrices, also probes can only be taken for each core individually. Probing the whole network from the example above requires defining and handling 3 probes for the compartment groups and 9 probes for the connection weight matrices. Note that this problem is only difficult in recurrent structures, especially in a reservoir where all neurons can potentially connect to each other. In feed-forward structures, even in deep spiking neural networks (already applied to several neuromorphic hardware systems, see e.g. Diehl et al., 2016; Schmitt et al., 2017; Patino-Saucedo et al., 2020; Massa et al., 2020), this problem is less predominant since the layers are connected in series. Multiple probes still need to be defined, but the connection matrices scale linear with the number of neuron groups and not quadratic as in reservoirs.

The `PeleNet` framework was developed to solve the distribution of connections efficiently and to make it easy for the user. In the framework, the experimenter only needs to define one connection matrix for every part of the network (e.g. for the reservoir or for the output layer). After the simulation, the user gets usable probes for every part of the network. In addition, `PeleNet` provides different distributions for initializing the connection weights, defining learning rules, creating standard plots, logging relevant computation steps and a collection of utils for calculating statistics and handling data. Moreover, the framework is a whole new abstraction layer on top of the `NxSDK`. Compartments, connections and probes

are defined implicitly and are controlled via parameters. Due to its modular and object oriented architecture, the framework can easily be extended with additional functionality. Here, I give a brief overview of the code structure and the main features. The code is available on Github [1] under the MIT license.

Design and implementation

Pele is the goddess of volcanoes and fire in the Hawaiian religion (Nimmo, 1986; Emerson, 2013). She has the control over lava and volcanoes and is inter alias in control of the volcano *Loihi*. The name of the `PeleNet` framework is an eponym of the goddess Pele.

The framework is build to allow experiments with reservoir networks on Loihi. As shown in Figure 1A, `PeleNet` currently supports reservoir networks that follow Dale's law. The reservoir contains a pool of excitatory and a pool of inhibitory neurons. Those neuron pools are connected within and in-between. Additionally an input, noise and an output is available. The spiking data can be read out from the excitatory, inhibitory and output neurons. Every experiment can contain one or multiple trials. Figure 3 shows an example with 10 trials. The spiking activity can optionally be reset after every trial. With this, it is possible to simulate much faster, since is is not necessary to initialize the network again after every trial.

Programmatically, `PeleNet` consists of two main parts. One part contains some helper functions and external libraries which are not available as a package. This part is imported from the `PeleNet` framework internally, the user does not need to import modules from this part. The other part consists of the `PeleNet` code itself which is imported and used by the user.

Libraries

The `lib` folder currently contains code to generate an anisotropic connectivity matrix and some helper functions and classes. The code for initializing an anisotropic connectivity matrix is public on Github [2]. The underlying principle was introduced in (Spreizer et al., 2019) and used in (Michaelis et al., 2020) for generating robust robotic trajectories, using the `PeleNet` framework. The `helper` folder contains custom exception functions for invalid parameters or invalid function arguments and a `Singleton` class to decorate classes in the `PeleNet` framework to make use of the singleton design pattern.

PeleNet structure

The central entity of `PeleNet` is the *experiment*, which inherits from an *abstract experiment*. In an experiment one or several *networks* can be defined and used. In the experiment, a *parameter* set is defined and overwrites values of the default parameter object. The *parameters* are passed to every *network*, when it is initialized by the *experiment*. In addition, every *network* contains a *plot* object, which has access to the data sets and probes of the simulation. Passing data to one of the *plot* methods is therefore in most cases not necessary. Hence, the arguments in den *plot* method shape the plots appropriately in size, limits, labels, colors, etc. Two singleton objects are globally available to all other objects. The *system* singleton contains a *datalog* object which logs all important steps in a log file. It also logs the parameter set which was used for a particular experiment and basic plots and optionally

[1] PeleNet on Github: `https://github.com/sagacitysite/pelenet`

[2] Code for generating anisotropic connection weights on Github: `https://github.com/babsey/spatio-temporal-activity-sequence/tree/6d4ab597c98c01a2a9aa037834a0115faee62587`

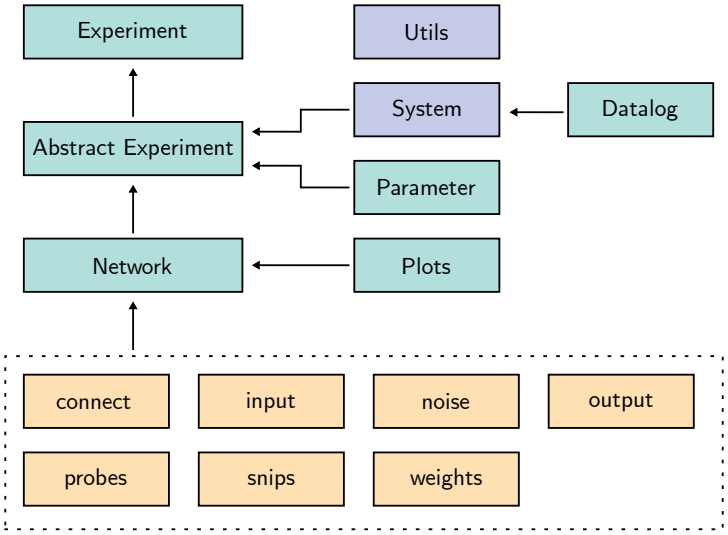


Figure 2: The code structure of the `PeleNet` framework. Classes have a green background, singletons are in purple and collections of methods for a class are yellow.

data sets from this experiment. The *utils* singleton provides a bunch of methods to handle and evaluate data, like dimensionality reduction, smoothing, calculating the spectral radius of the weight matrix and different kind of statistics. Figure 2 gives an overview of the dependencies between the classes in `PeleNet`.

Network

The major component of the `PeleNet` framework is the *network*. It contains a collection of methods, distributed over several files. Since this is the core of the framework, I will give some more details about the behavior in the following listing.

weights Methods in the `weights` file initialize the weight matrices for all parts of the network. The basic weight matrices are for connecting the excitatory and inhibitory parts of the reservoir. Weights can be initialized using constant values or a log-normal or normal distribution. In addition, a 2D topological anisotropic weight matrix can be initialized. All weight matrices are stored sparsely in a compressed sparse row (CSR) format.

connect Takes a weight matrix for every part of the network (e.g. one weight matrix for the whole reservoir), splits it in parts (called chunks in the framework), distributes these parts to the cores and interconnects them to each other. It is currently still necessary to define the number of neurons that should be used per core as a parameter. For an efficient distribution we need to consider two aspects. First, every core time-multiplexes up to 1024 neurons, the less neurons we simulate on every core, the faster the simulation will be. Second, the more cores are used, the more connection matrices are required, which slows down the initialization of the network. For an optimal run-time performance we should therefore reduce the number of neurons per core as much as possible and for an optimal initialization performance we should use as much neurons per core as possible. It is currently up to the user to choose best fitting values for this trade-off, but in most cases it is probably the fast simulation time which is desired. Later versions of the framework may allow the user to choose a preferred method and handles the distribution

of neurons automatically.

input Adds different types of inputs to the network. All of them base on the *spike generators* from the `NxSDK`. Currently topological inputs (in case of a 2D-network), noisy inputs (leave-n-neurons out in every trial), sequences of inputs and varying input positions per trial are supported. Topological inputs define a square of stimulated neurons in the excitatory layer of the reservoir at a defined position. Noisy inputs stimulate a specific number of reservoir neurons, but in every trial some few neurons are left out such that the input differs slightly in every trial. A sequence of inputs are multiple input regions which are stimulated in a row within one trial such that relations between them can be learned (e.g. with spike-timing-dependent plasticity). An example for a input sequence is shown in Figure 3. Finally, it is possible to define separate input regions, where one input region is randomly chosen in every trial.

noise Noise is currently generated by random inputs from *spike generators* which are connected to randomly chosen neurons in the network. Future implementations will probably make use of random changes in the current of a synapse or the membrane voltage of the neuron.

output Adds output neurons to the reservoir. Currently the only available output is a pooling layer, which was used in Michaelis et al. (2020). The pooling layer was used for a faster read out and performed regularization for the anisotropic network.

probes Contains several methods to define and process probes. First, probes are defined for every core and every connection matrix chunk. Second, after a successful simulation, probe data are post-processed and stacked together to complete and useful data sets. Note that the output of a connection weight probes is in CSR format again such that this matrix can directly be used as an initial connection matrix for another simulation.

snips Handles small `C`-scripts that run on the x86 cores of the Loihi chips (so called SNIPs). These scripts are located in the `pelenet/snips` folder. Currently a reset SNIP is available that resets the membrane voltages after a trial. It is important to note that the plasticity is currently not stopped while resetting the membrane voltages, which can cause problems when the *uk* variables ares used. But this feature is under development and will probably be added soon.

Experiments

An *experiment* inherits from an *abstract experiment* and is created in the `pelenet/experiments` folder. The *abstract experiment* inherits again from the `ABC` package, which allows defining abstract methods. The `defineParameters` method in the *abstract experiment* class is implemented as an abstract method that is then necessary in every *experiment*, otherwise an exception is thrown. The *abstract experiment* also provides some default functionality, which can optionally be overwritten. It initializes all necessary objects for the experiment and contains a default build process. Also the execution of the simulation follows a default behavior. In both cases, it is preferred to control the behavior of the *experiment* via parameters instead of overwriting methods. If the parameters do not cover the wanted behavior, it is suggested to make use of the available lifecycle methods. Available lifecycle methods are:

```
1  # Import abstract experiment
2  from ._abstract import Experiment
3
4  """
5  @desc: An experiment with a sequential input, trained over several trials
6  """
7  class SequenceExperiment(Experiment):
8
9      """
10     @desc: Define parameters for this experiment
11     """
12     def defineParameters(self):
13         return {
14             # Experiment
15             'seed': 1,  # Set fixed random seed
16             'trials': 10,  # Number of trials
17             'stepsPerTrial': 60,  # Number of simulation steps for every trial
18             # Neurons
19             'refractoryDelay': 2, # Refactory period
20             'voltageTau': 100,  # Voltage time constant
21             'currentTau': 5,  # Current time constant
22             'thresholdMant': 1200,  # Spiking threshold for membrane potential
23             # Network
24             'reservoirExSize': 400,  # Number of excitatory neurons
25             'reservoirConnPerNeuron': 35,  # Number of connections per neuron
26             'isLearningRule': True,  # Apply a learning rule
27             'learningRule': '2^-2*x1*y0 - 2^-2*y1*x0 + 2^-4*x1*y1*y0 - 2^-3*y0*w*w',
28             # Input
29             'inputIsSequence': True,  # Activates sequence input
30             'inputSequenceSize': 3,  # Number of input clusters in sequence
31             'inputSteps': 20,  # Number of steps a trace input is active
32             'inputGenSpikeProb': 0.8,  # Probability of spike for the input generator
33             'inputNumTargetNeurons': 40,  # Number of neurons activated by the input
34             # Probes
35             'isExSpikeProbe': True,  # Probe excitatory spikes
36             'isInSpikeProbe': True,  # Probe inhibitory spikes
37             'isWeightProbe': True  # Probe weight matrix at the end of every trial
38         }
```

Code Listing 1: Defining an experiment in the `pelenet/experiments` folder. The `defineParameters` method is required. Lifecycle methods are optional (not shown). In addition, the experiment can be extended by custom methods for e.g. data evaluation or visualization.

- `onInit`: Called after the experiment was initialized.
- `afterBuild`: Called after all network parts are connected (i.e. weight matrix, inputs, outputs, noise, probes).
- `afterRun`: Called after the simulation has finished and all data are post-processed.

Only if the parameters and the lifecycle methods are not sufficient to solve the intended behavior it is suggest to overwrite the `build` and `run` methods of the *abstract experiment*.

In practice, `Jupyter` notebooks are used to quickly evaluate the results of an *experiment*. It is suggested to use `Jupyter` notebooks only for visualization of the results and for prototyping. Code for the experiment should be included in the defined *experiment*. An example of a simple *experiment* is shown in Code Listing 1. In Code Listing 2 the experiment is used. The plotted spike train from the Code Listing 2 (line 25) is shown in Figure 3.

```python
# Load pelenet modules
from pelenet.experiments.sequence import SequenceExperiment

# Overwrite default parameters from pelenet/experiments/sequence.py
parameters = {
    # Experiment
    'seed': 2,  # Change random seed
}

# Initialize experiment
# The name is optional, it is extended to the folder in the log directory
# The parameters defined above are handed over to the experiment object
exp = SequenceExperiment(
    name='random-network-sequence-learning',
    parameters=parameters
)

# Build the network (weight matrix, inputs, probes, etc)
exp.build()

# Run the network simulation, afterwards the probes are post-processed to nice arrays
exp.run()

# Plot spike trains of the excitatory and inhibitory neurons
exp.net.plot.reservoirSpikeTrain(figsize=(12,6))
```

Code Listing 2: An example for running the experiment. For this Jupyter notebooks can be used. Parameters of the experiment can be overwritten to allow fast experimentation. At the end of the script an included plotting method is called to show spike trains. They are directly plotted in Jupyter and stored in the log folder related to this simulation.

Parameters

The parameter system is a powerful tool of `PeleNet` for defining an experiment. All `NxSDK` functionalities, which are included in `PeleNet`, are covered by the parameter set. It is not required to know anything about the `NxSDK` at all, if the functionality provided by `PeleNet` is sufficient for the user. The default parameters are split in three parts, parameters for the experiment, the system and derived parameters. The system parameters cover information about e.g. the used Loihi board, settings for the `matplotlib` library, logging and paths. The experiment parameters contain e.g. neurons, connections, inputs, outputs, noise, probes and also a learning rule for the reservoir. Finally, the derived parameters are calculated from the system and experiment parameters and are useful for the framework or for the user. Some parameters are sanity checked to avoid serious issues with false parameters. These checks are constantly extended. Note that the parameters are well documented in the `pelenet/parameters` file to understand their meaning, but it is not suggested to overwrite the parameters there. For overwriting parameters the `defineParameters` method is available in the *experiment*. All parameter values which are defined in this method overwrite the default parameter set. Derived parameters are calculated after the parameters are defined in an *experiment*.

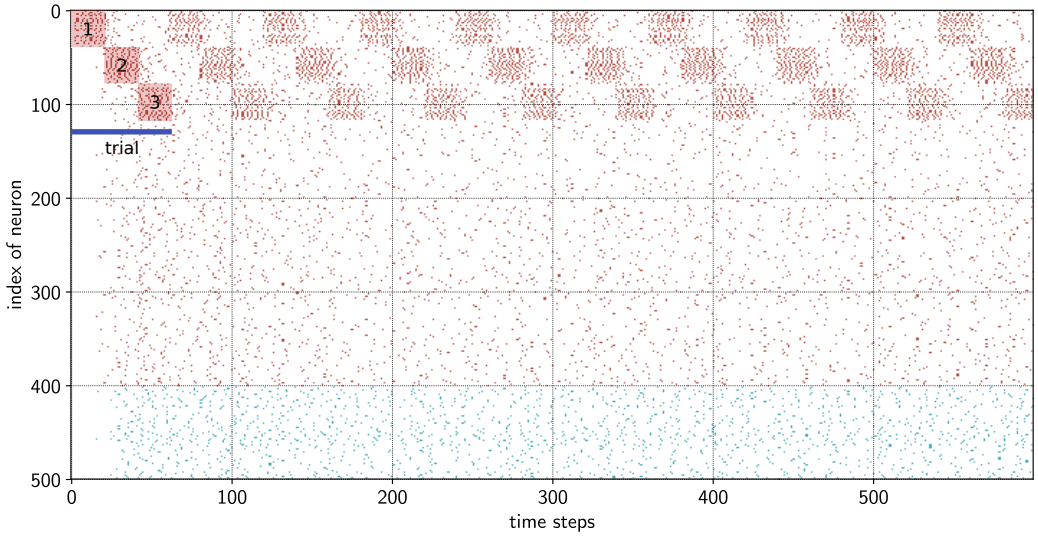


Figure 3: Example for a spike raster plot that shows inputs and trials. The experiment was performed with 10 trials and a input sequence with three inputs.

Discussion

Currently the neuromorphic hardware community grows fast and it is probably only a matter of time until new or updated hardware systems will emerge. The field of applications is very broad for such hardware, including the estimation of linear models, like LASSO (Shapero et al., 2014; Davies et al., 2018), non-parametric classification with k-nearest neighbor (Frady et al., 2020), deep spiking neural network (Massa et al., 2020) or reservoir networks (Michaelis et al., 2020). High-level libraries and frameworks are needed to cover these different specialized application areas. While for deep spiking neural networks the `SNN Toolbox` (Rueckauer et al., 2017; Rueckauer and Liu, 2018) is available for ANN to SNN conversions (also for Loihi) or `SLAYER` is available for training SNNs via backpropagation, reservoir network frameworks are still rare for neuromorphic computing.

The here presented `PeleNet` framework simplifies the implementation of reservoir networks on the neuromorphic hardware Loihi. The framework is an abstraction layer on top of the `NxSDK` from Intel. `PeleNet` allows an efficient distribution of the network over an arbitrary number of Loihi cores and chips. Probes are combined to data sets, which can directly be used for further evaluations. Parameters define the experiments. The "parameter approach" allows an easy initialization process and keeps the experiments clear. Finally, the plot systems already includes a bunch of default plots for reservoir computing.

Beside its already existing features, the framework is still in an early stage. Some functionalities of the Loihi chip are not provided yet. This includes inter alia current and voltage noise, usage of the tag for learning rules and synaptic delays. The learning rule covers the excitatory neurons in the reservoir, future version will also be able to apply a learning rule to output neurons. In addition, the framework is not yet available as a `pip` python package and still requires a manual installation of dependencies. Therefore one of the next steps will be to provide a package release of `PeleNet`. It is also intended to add an optimization functionality which runs several experiments in order to find optimal parameters, according to a criterion. The parameter system is already designed to support future optimization scripts. Finally, it is planned to add unit and integration tests to make sure that simulations are performed correctly. Currently four `Jupyter` notebooks are available as functional tests.

The aim of `PeleNet` is to speed up the implementation of reservoir computing experiments on Loihi. In a more broad perspective it has even the potential to push the field of reservoir computing implementations on neuromorphic hardware in general. Despite its early development stage, I am confident that the framework is already useful for computational studies.

License and code availability

The framework `PeleNet` is published under the MIT License and is therefore freely available without warranty. The code is available on Github[3]. Contributions are highly appreciated.

Acknowledgement

I sincerely thank Dr. Christian Tetzlaff for providing me with funding for my doctoral thesis and his supervision. In addition, I want to thank Intel for providing the Kapoho Bay to our lab and ongoing support. Finally I want to thank Andrew B. Lehr, since he is always willing to discuss problems constructively and influenced the implementation of the anisotropic network significantly.

References

Backus, J. (1978). Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99.

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE.

Emerson, N. B. (2013). *Pele and Hiiaka: a myth from Hawaii*. Tuttle Publishing.

Frady, E. P., Orchard, G., Florey, D., Imam, N., Liu, R., Mishra, J., Tse, J., Wild, A., Sommer, F. T., and Davies, M. (2020). Neuromorphic nearest neighbor search using intel's pohoiki springs. In *Proceedings of the Neuro-Inspired Computational Elements Workshop*, NICE '20, New York, NY, USA. Association for Computing Machinery.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

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

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., and Wang, H. (2018). Programming spiking neural networks on intel's loihi. *Computer*, 51(3):52–61.

[3]PeleNet on Github: `https://github.com/sagacitysite/pelenet`

Lukoševičius, M., Jaeger, H., and Schrauwen, B. (2012). Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Massa, R., Marchisio, A., Martina, M., and Shafique, M. (2020). An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor. *arXiv preprint arXiv:2006.09985*.

Michaelis, C., Lehr, A. B., and Tetzlaff, C. (2020). Robust robotic control on the neuromorphic research chip loihi. *arXiv preprint arXiv:2008.11642*.

Nimmo, H. A. (1986). Pele, ancient goddess of contemporary hawaii. *Pacific Studies*, 9(2):121.

Patino-Saucedo, A., Rostro-Gonzalez, H., Serrano-Gotarredona, T., and Linares-Barranco, B. (2020). Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the spinnaker neuromorphic platform. *Neural Networks*, 121:319–328.

Rajendran, B., Sebastian, A., Schmuker, M., Srinivasa, N., and Eleftheriou, E. (2019). Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, 36(6):97–110.

Rueckauer, B. and Liu, S.-C. (2018). Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682.

Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Guettler, M., Hartel, A., Hartmann, S., Husmann, D., Husmann, K., Jeltsch, S., et al. (2017). Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234. IEEE.

Schrauwen, B., Verstraeten, D., and Van Campenhout, J. (2007). An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pages 471–482.

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., and Plank, J. S. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*.

Shapero, S., Zhu, M., Hasler, J., and Rozell, C. (2014). Optimal sparse approximation with integrate and fire neurons. *International journal of neural systems*, 24(05):1440001.

Spreizer, S., Aertsen, A., and Kumar, A. (2019). From space to time: Spatial inhomogeneities lead to the emergence of spatiotemporal sequences in spiking neuronal networks. *PLoS computational biology*, 15(10):e1007432.

Tang, G., Shah, A., and Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *arXiv preprint arXiv:1903.02504*.

Young, A. R., Dean, M. E., Plank, J. S., and Rose, G. S. (2019). A review of spiking neuromorphic hardware communication systems. *IEEE Access*, 7:135606–135620.

Brian2Loihi: An emulator for the neuromorphic chip Loihi using the spiking neural network simulator Brian

Carlo Michaelis[1,2,*], Andrew B. Lehr[1,2,*], Winfried Oed[1,2,*], and Christian Tetzlaff[1,2]

[1]Department of Computational Neuroscience, University of Göttingen, Germany
[2]Bernstein Center for Computational Neuroscience, University of Göttingen, Germany
[*]These authors contributed equally.

Abstract

Developing intelligent neuromorphic solutions remains a challenging endeavour. It requires a solid conceptual understanding of the hardware's fundamental building blocks. Beyond this, accessible and user-friendly prototyping is crucial to speed up the design pipeline. We developed an open source Loihi emulator based on the neural network simulator Brian that can easily be incorporated into existing simulation workflows. We demonstrate errorless Loihi emulation in software for a single neuron and for a recurrently connected spiking neural network. On-chip learning is also reviewed and implemented, with reasonable discrepancy due to stochastic rounding. This work provides a coherent presentation of Loihi's computational unit and introduces a new, easy-to-use Loihi prototyping package with the aim to help streamline conceptualisation and deployment of new algorithms.

1 Introduction

Neuromorphic computing offers exciting new computational structures. Decentralised units inspired by neurons are implemented in hardware (reviewed by Rajendran et al., 2019; Schuman et al., 2017; Young et al., 2019). These can be connected up to one another, stimulated with inputs, and the resulting activity patterns can be read out from the chip as output. A variety of algorithms and applications have been developed in recent years, including robotic control (DeWolf et al., 2020; DeWolf et al., 2016; Michaelis et al., 2020; Stagsted et al., 2020), spiking variants of deep learning algorithms, attractor networks, nearest-neighbor or graph search algorithms (reviewed by Davies et al., 2021). Moreover, neuromorphic hardware may provide a suitable substrate for performing large scale simulations of the brain (S. Furber, 2016; Thakur et al., 2018). Neuromorphic chips specialised for particular computational tasks can either be provided as a neuromorphic computing cluster or be integrated into existing systems, akin to graphics processing units (GPU) in modern computers (Davies et al., 2021; S. B. Furber et al., 2014). With the right ideas, networks of spiking units implemented in neuromorphic hardware can provide the basis for powerful and efficient computation. Nevertheless, the development of new algorithms for spiking neural networks, applicable to neuromorphic hardware, is a challenge (Bouvier et al., 2019; Grüning & Bohte, 2014; Pfeiffer & Pfeil, 2018).

At this point, without much background knowledge of neuromorphic hardware, one can get started programming using the various software development kits available (e.g., Brüderle et al., 2011; Lin et al., 2018; Michaelis, 2020; E. Müller, Mauch, et al., 2020; E. Müller, Schmitt, et al., 2020; Rhodes et al., 2018; Rueckauer et al., 2021; Sawada et al., 2016; Spilger et al., 2020). Emulators for neuromorphic hardware

(S. B. Furber et al., 2014; Luo et al., 2018; Petrovici et al., 2014; Valancius et al., 2020) running on a standard computer or field programmable gate arrays (FPGA), make it possible to develop neuromorphic network architectures without even needing access to a neuromorphic chip (see e.g. NengoLoihi[1] and Dynap-SE[2]). This can speed up prototyping as the initialisation of networks, i.e. distributing neurons and synapses, as well as the readout of the system's state variables on neuromorphic chips takes some time. At the same time emulators transparently contain the main functionalities of the hardware in code and therefore provide insights into how it works. With this understanding, algorithms can be intelligently designed and complex network structures implemented.

In the following, we introduce an emulator for the digital neuromorphic chip `Loihi` (Davies et al., 2018) based on the widely used spiking neural network simulator `Brian` (Stimberg et al., 2019). We first dissect an individual computational unit from `Loihi`. The basic building block is a spiking unit inspired by a current based leaky integrate and fire (LIF) neuron model (see Gerstner et al., 2014). Connections between these units can be plastic, enabling the implementation of diverse on-chip learning rules. Analysing the computational unit allows us to create an exact emulation of the `Loihi` hardware on the computer. We extend this to a spiking neural network model and demonstrate that both `Loihi` and `Brian` implementations match perfectly. This exact match means one can do prototyping directly on the computer using `Brian` only, which adds another emulator in addition to the existing simulation backend in the Nengo Loihi library. This increases both availability and simplicity of algorithm design for `Loihi`, especially for those who are already used to working with `Brian`. In particular for the computational neuroscience community, this facilitates the translation of neuroscientific models to neuromorphic hardware. Finally, we review and implement synaptic plasticity and show that while individual weights show small deviations due to stochastic rounding, the statistics of a learning rule are preserved. Our aim is to facilitate the development of neuromorphic algorithms by delivering an open source emulator package that can easily be incorporated into existing workflows. In the process we provide a solid understanding of what the hardware computes, laying the appropriate foundation to design precise algorithms from the ground up.

2 Loihi's computational unit and its implementation

Developing a `Loihi` emulator requires precise understanding of how `Loihi` works. And to understand how something works, it is useful to "take it apart and put it back together again". While we will not physically take the `Loihi` chip apart, we can inspect the components of its computational units with "pen and paper". Then, by implementing each component on a computer we will test that, when put back together, the parts act like we expect them to. In the following we highlight how spiking units on `Loihi` approximate a variant of the well-known LIF model using first order Euler numerical integration with integer precision. This understanding enables us to emulate `Loihi`'s spiking units on the computer in a way that is straightforward to use and easy to understand. For a better intuition of how the various parameters on `Loihi` interact, we refer readers to our neuron design tool[3] for `Loihi`. Readers familiar with Davies et al. (2018) and numerical implementations of LIF neurons may prefer to skip to Section 2.3.

2.1 `Loihi`'s neuron model: a recap

The basic computational unit on `Loihi` is inspired by a spiking neuron (Davies et al., 2018). `Loihi` uses a variant of the leaky integrate and fire neuron model (Gerstner et al., 2014) (see Appendix 7.1). Each

[1] https://www.nengo.ai/nengo-loihi/
[2] https://code.ini.uzh.ch/yigit/NICE-workshop-2021
[3] anonymized

unit i of `Loihi` implements the dynamics of the voltage v_i

$$\frac{dv_i}{dt} = -\frac{1}{\tau_v} v_i(t) + I_i(t) - v_i^{th}\sigma_i(t), \tag{1}$$

where the first term controls the voltage decay, the second term is the input to the unit, and the third term resets the voltage to zero after a spike by subtracting the threshold. A spike is generated if $v_i > v_i^{th}$ and transmitted to other units to which unit i is connected. In particular, v models the voltage across the membrane of a neuron, τ_v is the time constant for the voltage decay, I is an input variable, v^{th} is the threshold voltage to spike, and $\sigma(t)$ is the so-called spike train which is meant to indicate whether the unit spiked at time t. For each unit i, $\sigma_i(t)$ can be written as a sum of Dirac delta distributions

$$\sigma_i(t) = \sum_k \delta(t - t_{i,k}), \tag{2}$$

where $t_{i,k}$ denotes the time of the k-th spike of unit i. Note that σ_i is not a function, but instead defines a *distribution* (i.e. *generalised function*), and is only meaningful under an integral sign. It is to be understood as the linear functional $\langle \sigma_i, f \rangle := \int \sigma_i(t) f(t)\, dt = \sum_k f(t_{i,k})$ for arbitrary, everywhere-defined function f (see Corollary 1 in Appendix 7.1.2).

Input to a unit can come from user defined external stimulation or from other units implemented on chip. Davies et al. (2018) describe the behavior of the input $I(t)$ with

$$I_i(t) = \sum_j J_{ij}(\alpha_I * \sigma_j)(t) + I_i^{\text{bias}}, \tag{3}$$

where J_{ij} is the weight from unit j to i, I_i^{bias} is a constant bias input, and the spike train σ_j of unit j is convolved with the synaptic filter impulse response α_I, given by

$$\alpha_I(t) = \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{4}$$

where τ_I is the time constant of the synaptic response and $H(t)$ the unit step function. Note that $\alpha_I(t)$ is defined differently here than in Davies et al. (2018) (see Appendix 7.1.3 for details). The convolution from Equation 3 is a notational convenience for defining the synaptic input induced by an incoming spike train, simply summing over the time-shifted synaptic response functions, namely $(\sigma_i * f)(t) = \langle \sigma_i, \tau_t \tilde{f} \rangle = \sum_k f(t - t_{i,k})$, where $\tau_t f(x) = f(x - t)$ and $\tilde{f}(x) = f(-x)$ (see Appendix 7.1.2).

2.2 Implementing `Loihi`'s spiking unit in software

From the theoretical model on which `Loihi` is based, we can derive the set of operations each unit implements with a few simple steps. Using a first order approximation for the differential equations gives the update equations for the voltage and synaptic input described in the `Loihi` documentation. Combined with a few other details regarding `Loihi`'s integer precision and the order of operations, we will have all we need to implement a `Loihi` spiking unit in software.

Synaptic input

From Equation 3 we see that the synaptic input can be written as a sum of exponentially decaying functions with amplitude J_{ij} beginning at the time of each spike $t_{j,k}$ (see Appendix 7.1.2). In particular we have

$$I_i(t) = \sum_j J_{ij} \sum_k \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}) + I_i^{\text{bias}}. \tag{5}$$

To understand the behavior of the synaptic input it is helpful to consider the effect of one spike arriving at a single synapse. Simplifying Equation 5 to just one neuron that receives just one input spike at time $t_1 = 0$, for $t \geq 0$ we get

$$I(t) = J \cdot \exp\left(-\frac{t}{\tau_I}\right) \tag{6}$$

and for $t < 0$, $I(t) = 0$. Each spike induces a step increase in the current which decays exponentially with time constant τ_I. Taking the derivative of both sides with respect to t gives

$$\frac{dI}{dt} = -\frac{1}{\tau_I} \cdot I(t), \tag{7}$$

$$I(0) = J. \tag{8}$$

Applying the forward Euler method to the differential equation for $\Delta t = 1$ and $t \geq 0$, $t \in \mathbb{N}$ we get

$$I[t] = I[t-1] - \frac{1}{\tau_I} \cdot I[t-1] + J \cdot s[t], \tag{9}$$

where $s[t]$ is zero unless there is an incoming spike on the synapse, in which case it is one. Here, $s[0] = 1$ and $s[t] = 0$ for $t > 0$. With this we have simply incorporated the initial condition into the update equation. Note that we have switched from a continuous (e.g. $I(t)$) to discrete (e.g. $I[t]$) time formulation, where $\Delta t = 1$ and t is unitless.

`Loihi` has a decay value δ^I, which is inversely proportional to τ_I, namely $\delta^I = 2^{12}/\tau_I$. Swapping τ_I by δ^I reveals

$$I[t] = I[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + J \cdot s[t]. \tag{10}$$

The weight J is defined via the mantissa $\tilde{w}_{ij}$ and exponent Θ (see Section 3.1) such that the equation describing the synaptic input becomes (with indices)

$$I_i[t] = I_i[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + 2^{6+\Theta} \cdot \sum_j \left(\tilde{w}_{ij} \cdot s_j[t]\right), \tag{11}$$

where $s_j[t] \in \{0, 1\}$ is the spike state of the j^{th} input neuron. Please note that Equation 11 is identical to the `Loihi` documentation.

From this we can conclude that the implementation of synaptic input on Loihi is equivalent to evolving the LIF synaptic input differential equation with the forward Euler numerical integration method (see Figure 1A1).

Voltage

It is straightforward to perform the same analysis as above for the voltage equation. We consider the subthreshold voltage dynamics for a single neuron and can therefore ignore the reset term $v_i^{th}\sigma_i(t)$ from Equation 1, leaving us with

$$\frac{dv}{dt} = -\frac{1}{\tau_v} v(t) + I(t). \tag{12}$$

Applying forward Euler gives

$$v[t] = v[t-1] - \frac{v[t-1]}{\tau_v} + I[t]. \tag{13}$$

Again, to compare with the `Loihi` documentation we need to swap the time constant τ_v by a voltage decay parameter, δ^v, which is inversely proportional to the time constant, the same as above for synaptic input. Plugging in $\tau_v = 2^{12}/\delta^v$ leads to

$$v[t] = v[t-1] \cdot (2^{12} - \delta^u) \cdot 2^{-12} + I[t]. \tag{14}$$

By introducing a bias term, the voltage update becomes

$$v_i[t] = v_i[t-1] \cdot (2^{12} - \delta^u) \cdot 2^{-12} + I_i[t] + I_i^{\text{bias}}. \tag{15}$$

Equation 15 agrees with the `Loihi` documentation. Like the synaptic input, the voltage implementation on `Loihi` is equivalent to updating the LIF voltage differential equation using forward Euler numerical integration (see Figure 1A2).

Integer precision

`Loihi` uses integer precision. So the mathematical operations in the update equations above are to be understood in terms of integer arithmetic. In particular, for the synaptic input and voltage equations the emulator uses *round away from zero*, which can be defined as

$$x_{\text{round}} \coloneqq \text{sign}(x) \cdot \lceil |x| \rceil. \tag{16}$$

where $\lceil \cdot \rceil$ is the ceiling function and $\text{sign}(\cdot)$ the sign function.

2.3 Summary

We now have all of the pieces required to understand and emulate a spiking unit from `Loihi`. Evolving the differential equations for the current-based LIF model with the forward Euler method and using the appropriate rounding (see Section 2.2) and update schedule (see Section 4.1 and Appendix 7.2.1) is enough to exactly reproduce `Loihi`'s behavior. This procedure is summarized in Algorithm 1 and an exact match between `Loihi` and an implementation for a single unit in `Brian` is shown in Figure 1A. Please note that during the refractory period `Loihi` uses the voltage trace to count elapsed time (see Figure 1A2, Appendix 7.2.2), while in the emulator the voltage is simply clamped to zero.

Algorithm 1: Loihi single neuron emulator

Result: Simulate one Loihi unit with one input synapse for t_{max} time steps and read out state variables (I, v) and spikes (σ).

```
# Define round away from zero
```
$\mathrm{rnd}(\cdot) \coloneqq \mathrm{sign}(\cdot)\lceil|\cdot|\rceil$
```
# Define input spike train
```
$S_t = \{0, 1\}\ \forall\ t \in \mathbb{N} \mid t \leq t_{max}$
```
# Define synaptic weight
```
$J := 2^{6+\Theta} \cdot \tilde{w},\ \Theta \in [-8, 7],\ \tilde{w} \in [-256, 255]$
```
# Define threshold
```
$v_{th} \coloneqq v_{mant} \cdot 2^6,\ v_{mant} \in [0, 131071]$
```
# Define voltage and current decay
```
$\tau_v = \delta^v / 2^{12},\ \delta^v \in [0, 4096]$

$\tau_I = \delta^I / 2^{12},\ \delta^I \in [0, 4096]$
```
# Initialise variables
```
$I_t, v_t, \sigma_t = 0\ \forall\ t \in \mathbb{N} \mid t \leq t_{max}$
```
# Loop over simulation steps
```
for t *from* 1 *to* t_{max} **do**
```
    # Spike input
```
    $s \leftarrow S_t$
```
    # Update and read synaptic input
```
    $I_t \leftarrow I_{t-1} - \mathrm{rnd}(\tau_I \cdot I_{t-1}) + J \cdot s$
```
    # Update and read voltage
```
    $v_t \leftarrow v_{t-1} - \mathrm{rnd}(\tau_v \cdot v_{t-1}) + I_t$
```
    # Check threshold
```
    if $v > v_{th}$ **then**
```
        # Read spike
```
        $\sigma_t \leftarrow 1$
```
        # Reset voltage
```
        $v_t \leftarrow 0$
    end
end

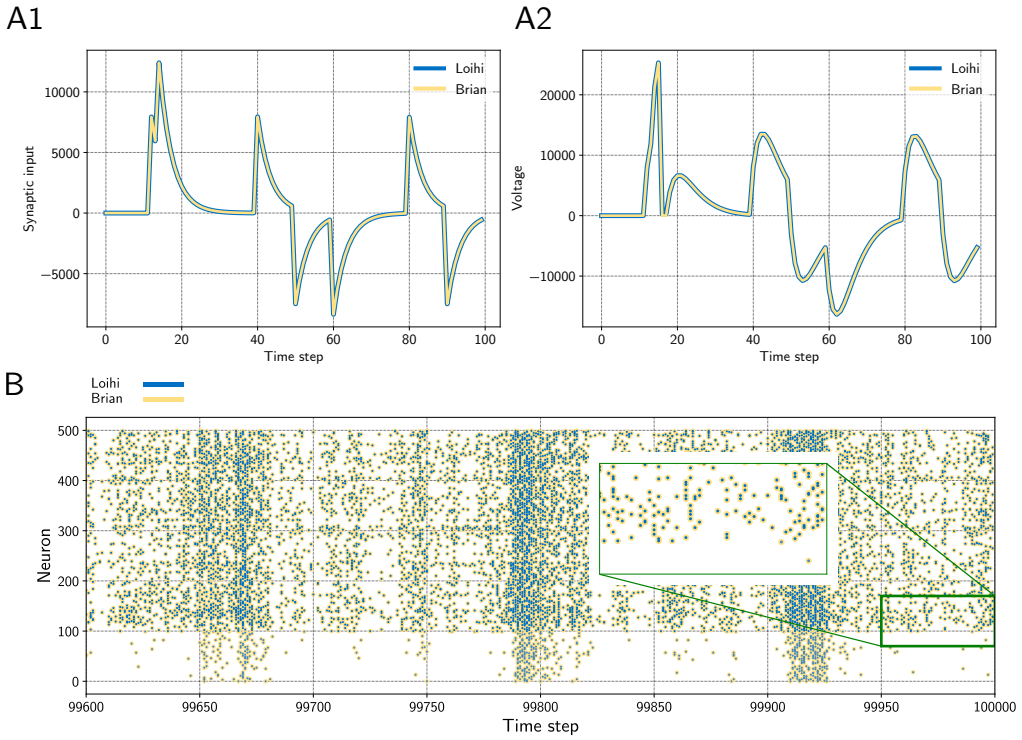


Figure 1: A Input trace of a single synapse and voltage trace of a neuron. The emulator matches `Loihi` in both cases perfectly. Note that `Loihi` uses the voltage register to count refractory time, which results in a functionally irrelevant difference after a spike, e.g time step 17 in A2 (see Appendix 7.2.2). **B** Network simulation with 400 excitatory (indices $100 - 500$) and 100 inhibitory (indices $0 - 100$) neurons. The network is driven by noise from an input population of 40 Poisson spike generators with a connection probability of 0.05. All spikes match exactly between the emulator and `Loihi` for all time steps. The figure shows the last 400 time steps from a simulation with 100 000 time steps.

3 Network and plasticity

We now have a working implementation of `Loihi`'s spiking unit. In the next step, we need to connect these units up into networks. And if the network should be able to learn online, connections between units should be plastic. In this section we review how weights are defined on `Loihi` and how learning rules are applied. This includes the calculation of pre- and post-synaptic traces. Based on this, we outline how these features are implemented in the emulator.

3.1 Synaptic weights

The synaptic weight consists of two parts, a weight mantissa $\tilde{w}$ and a weight exponent Θ and is of the form $\tilde{w} \cdot 2^{6+\Theta}$. However, in practice the calculation of the synaptic weight depends on bit shifts and its precision depends on a few parameters (see below). The weight exponent is a value between -8 and 7 that scales the weight mantissa exponentially. Depending on the sign mode of the weight (excitatory, inhibitory, or mixed), the mantissa is an integer in the range $\tilde{w} \in [0, 255]$, $\tilde{w} \in [-255, 0]$, or $\tilde{w} \in [-256, 254]$, respectively. The possible values of the mantissa depend on the number of bits available for storing the weight and whether the sign mode is *mixed* or not. In particular, precision is defined as

2^{n_s}, with

$$n_s = 8 - (n_{wb} - \sigma_{\text{mixed}}). \tag{17}$$

This can intuitively be understood with a few examples. If the weight bits for the weight mantissa are set to the default value of $n_{wb} = 8$ bits, it can store 256 values between 0 and 255, i.e. the precision is then $2^{8-(8-0)} = 2^0 = 1$. If $n_{wb} = 6$ bits is chosen, we instead have a precision of $2^{8-(6-0)} = 2^2 = 4$ meaning there are 64 possible values for the weight mantissa, $\tilde{w} \in \{0, 4, 8, 16, ..., 252\}$. If the sign mode is *mixed*, i.e. $\sigma_{mixed} = 1$, one bit is used to store the sign, which reduces the precision. Mixed mode enables both positive and negative weights, with weight mantissa between -256 and 254. Assuming $n_{wb} = 8$ in mixed mode, precision is $2^{8-(8-1)} = 2^1 = 2$ and $\tilde{w} \in \{-256, -254, ..., -4, -2, 0, 2, 4, ..., 254\}$.

3.1.1 Weight initialisation

While the user can define an arbitrary weight mantissa within the allowed range, during initialisation the value is rounded, given the precision, to the next possible value towards zero. This is achieved via bit shifting, that is the weight mantissa is shifted by

$$\tilde{w}^{\text{shifted}} = (\tilde{w} \gg n_s) \ll n_s, \tag{18}$$

where $\gg$ and $\ll$ are a right and left shift respectively. Afterwards the weight exponent is used to scale the weight according to

$$J^{\text{scaled}} = \tilde{w}^{\text{shifted}} \cdot 2^{6+\Theta}. \tag{19}$$

This value cannot be greater than 21 bits and is clipped if it exceeds this limit. Note that this only happens in one case for $\tilde{w} = -256$ and $\Theta = 7$. Finally the scaled value J^{scaled} is shifted again according to

$$J = (J^{\text{scaled}} \gg 6) \ll 6, \tag{20}$$

where J is the final weight.

We provide a table with all 4096 possible weights depending on the mantissa and the exponent in a Jupyter notebook[4].These values are provided for all three sign modes.

3.1.2 Plastic synapses

In the case of a *static* synapse, the initialised weight remains the same as long as the chip/emulator is running. Thus *static* synapses are fully described by the details above. For *plastic* synapses, the weight can change over time. This requires a method to ensure that changes to the weight adhere to its precision.

For *plastic* synapses, *stochastic rounding* is applied to the mantissa during each weight update. Whether the weight mantissa is rounded up or down depends on its proximity to the nearest possible values above and below, i.e.

$$\text{RS}_{2^{n_s}}(x) = \begin{cases} \text{sign}(x) \cdot \lfloor |x| \rfloor_{2^{n_s}} & \text{with probability } (2^{n_s} - (|x| - \lfloor |x| \rfloor_{2^{n_s}}))/2^{n_s} \\ \text{sign}(x) \cdot (\lfloor |x| \rfloor_{2^{n_s}} + 2^{n_s}) & \text{with probability } (|x| - \lfloor |x| \rfloor_{2^{n_s}}))/2^{n_s} \end{cases} \tag{21}$$

where $\lfloor \cdot \rfloor_{2^{n_s}}$ denotes rounding down to the nearest multiple of 2^{n_s}. After the mantissa is rounded, it is scaled by the weight exponent and the right/left bit shifting is applied to the result to compute the actual weight J. How this is realised in the emulator is shown in Code Listing 3.

To test that our implementation of the weight update for *plastic* synapses matches `Loihi` for each possible number of weight bits, we compared the progression of the weights over time for a simple learning rule. The analysis is described in detail in Appendix 7.3.

[4] anonymized

3.2 Pre- and post-synaptic traces

Pre- and post-synaptic traces are used for defining learning rules. `Loihi` provides two pre-synaptic traces x_1, x_2 and three post-synaptic traces y_1, y_2, y_3. Pre-synaptic traces are increased by a constant value $\hat{x}_i$, for $i \in \{1, 2\}$, if the pre-synaptic neuron spikes. The post-synaptic traces are increased by $\hat{y}_j$ for $j \in \{1, 2, 3\}$, accordingly. So-called *dependency factors* are available, indicating events like $x_0 = 1$ if the pre-synaptic neuron spikes or $y_0 = 1$ if the post-synaptic neuron spikes. These factors can be combined with the trace variables by addition, subtraction, or multiplication.

A simple spike-time dependent plasticity (STDP) rule with an asymmetric learning window would, for example, look like $dw = x_1 \cdot y_0 - y_1 \cdot x_0$. This rule leads to a positive change in the weight ($dw > 0$) if the pre-synaptic neuron fires shortly before the post-synaptic neuron (i.e. positive trace $x_1 > 0$ when $y_0 = 1$) and to a negative change ($dw < 0$) if the post-synaptic neuron fires shortly before the pre-synaptic neuron (i.e. positive trace $y_1 > 0$ when $x_0 = 1$). Thus, the time window in which changes may occur depends on the shape of the traces (i.e. impulse strength $\hat{x}_i$, $\hat{y}_i$; and decay τ_{x_i}, τ_{y_j}, see below).

For a sequence of spikes $s[t] \in \{0, 1\}$, a trace is defined as

$$x_i[t] = \alpha \cdot x_i[t-1] + \hat{x}_i \cdot s[t], \tag{22}$$

where α is a decay factor (see Davies et al., 2018). This equation holds for presynaptic (x_i) and post-synaptic (y_i) traces. However, in practice, on `Loihi` one does not set α directly but instead decay time constants τ_{x_i} and τ_{y_j}.

In the implementation of the emulator we again assume a first order approximation for synaptic traces, akin to synaptic input and voltage. Under this assumption for the exponential decay, in Equation 22 we replace α by

$$\alpha(\tau_{x_i}) = 1 - \frac{1}{\tau_{x_i}}. \tag{23}$$

Using this approximation gives reasonable results across a number of different τ_{x_i} and τ_{y_i} values (see Figure 4). While this essentially suffices, it could be improved by introducing an additional parameter, e.g. β, and optimising $\alpha(\tau_{x_i}, \beta)$.

Note that we have integer precision again. But different from the *round away from zero* applied in the neuron model, here *stochastic rounding* is used. Since traces are positive values between 0 and 127 with precision 1, the definition above in Equation 21 simplifies to the following

$$\mathrm{RS}_{1,\geq 0}(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\ \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor \end{cases} \tag{24}$$

Since this rounding procedure is probabilistic and the details of the random number generator are unknown, rounding introduces discrepancies when emulating `Loihi` on the computer. Further improvements are possible if more details of the chip's rounding mechanism were to be considered.

3.3 Summary

At this point we are able to connect neurons with synapses and build networks of neurons (see Figure 1B). It was shown how the weights are handled, depending on the user defined number of weight bits or the sign mode. In addition, using the dynamics of the pre- and post synaptic traces, we can now define learning rules. Note that different from the neuron model, the synaptic traces cannot be reproduced exactly since the details of the random number generator, used for stochastic rounding, are unknown. However, Figure 2 shows that the synaptic traces emulated in `Brian` are very close to the original ones in `Loihi` and that the behavior of a standard asymmetric STDP rule can be reproduced with the emulator.

4 Loihi emulator based on Brian

Here we provide an overview over the emulator package and show some examples and results. This enables straightforward emulation of the basic features from `Loihi` as a sandbox for experimenters. Note that we have explicitly not included routing and mapping restrictions, like limitations for the number of neurons or the amount of synapses, as these depend on constraints such as the number of used `Loihi` chips.

4.1 The package

The emulator package is available on *PyPI*[5] and can be installed using the `pip` package manager. The emulator does not provide all functionality of the `Loihi` chip and software, but the main important aspects. An overview over all provided features is given in Table 1 in the appendix. It contains six classes that extend the corresponding `Brian` classes. The classes are briefly introduced in the following. Further details can be taken from the code[6].

Network

The `LoihiNetwork` class extends the `Brian` `Network` class. It provides the same attributes as the original `Brian` class. The main difference is that it initializes the default clock, the integration methods and updates the schedule when a `Network` instance is created. Note that it is necessary to make explicitly use of the `LoihiNetwork`. It is not possible to use `Brian`'s *magic network*.

Voltage and synaptic input are evolved with the forward Euler integration method, which was introduced in Section 2.2. Additionally a state updater was defined for the pre- and post-synaptic traces.

The default network update schedule for the computational order of the variables from `Brian` do not match the order of the computation on `Loihi`. The `Brian` update schedule is therefore altered when initialising the `LoihiNetwork`, more details are given in Appendix 7.2.1.

Neuron group

The `LoihiNeuronGroup` extends `Brian`'s `NeuronGroup` class. Parameters of the `LoihiNeuronGroup` class are mostly different from the `Brian` class and are related to `Loihi`. When an instance is created, the given parameters are first checked to match requirements from `Loihi`. Finally, the differential equations to describe the neural system are shown in Code Listing 1. Since `Brian` does not provide a *round away from zero* functionality, we need to define it manually as an equation.

Synapses

The `LoihiSynapses` class extends the `Synapses` class from `Brian`. Again, most of the `Brian` parameters are not supported and instead `Loihi` parameters are available. When instantiating a `LoihiSynapses` object, the needed pre- and post-synaptic traces are included as equations (shown in Code Listing 2) as theoretically introduced in Section 3.2. Moreover, it is verified that the defined learning rule matches the

[5] https://pypi.org/project/brian2-loihi/
[6] anonymized

```
lif_equations = '''
    rnd_v = sign(v)*ceil(abs(v*1_tau_v)) : 1
    rnd_I = sign(I)*ceil(abs(I*1_tau_I)) : 1
    dv/dt = -rnd_v/ms + I/ms: 1 (unless refractory)
    dI/dt = -rnd_I/ms : 1
'''
```

Code Listing 1: Neuron model equations of the voltage and the synaptic input for `Brian`. It contains a *round away from zero* rounding.

```
x1decay_equations = '''
    x1_new = x1 * (1 - (1.0/tau_x1)) : 1
    x1_int = int(x1_new) : 1
    x1_frac = x1_new - x1_int : 1
    x1_add_or_not = int(x1_frac > rand()) : 1 (constant over dt)
    x1_rnd = x1_int + x1_add_or_not : 1
    dx1/dt = x1_rnd / ms : 1 (clock-driven)
'''
```

Code Listing 2: Synaptic decay equation for `Brian`. Only the decay for $x1$ is shown, the decay for $x2$, $y1$, $y2$, $y3$ is applied analogously. It contains an approximation of the exponential decay and stochastic rounding.

available variables and operations supported by `Loihi`. The equations for the weight update is shown in Code Listing 3.

Since we have no access to the underlying mechanism and we cannot reproduce the pseudo-stochastic mechanisms exactly, we have to find a stochastic rounding that matches `Loihi` in distribution. Note that on `Loihi` the same network configuration leads to reproducible results (i.e. same rounding). Thus to compare the behavior of `Loihi` and the emulator, we simulate over a number of network settings and compare the distribution of the traces. Figure 2B shows the match between the distributions. Note that with this, our implementation is always slightly different from the `Loihi` simulation, due to slight differences in rounding. In Figure 2C we show that these variations are constant and not diverging. In addition, Figure 2D shows that the principle behavior of a learning rule is preserved.

State monitor & Spike monitor

The `LoihiStateMonitor` class extends the `StateMonitor` class from `Brian`, while the `LoihiSpikeMonitor` class extends the `SpikeMonitor` class. Both classes support the most important parameters from their subclasses and update the schedule for the timing of the probes. This schedule update avoids shifts in the monitored variables, compared to `Loihi`.

Spike generator group

The `LoihiSpikeGeneratorGroup` extends the `SpikeGeneratorGroup` class from `Brian`. This class only reduces the available parameters to avoid that users unintentionally change variables which would cause an unwanted emulation behavior.

```
weight_equations = '''
    u0 = 1 : 1
    u1 = int(t/ms % 2**1 == 0) : 1
    ...
    u9 = int(t/ms % 2**9 == 0) : 1

    dw_rounded = int(sign(dw)*ceil(abs(dw))) : 1
    quotient = int(dw_rounded / precision) : 1
    remainder = abs(dw_rounded) % precision : 1
    prob = remainder / precision : 1
    add_or_not = sign(dw_rounded) * int(prob > rand()) : 1 (constant over dt)
    dw_rounded_to_precision = (quotient + add_or_not) * precision : 1
    w_updated = w + dw_rounded_to_precision : 1
    w_clipped = clip(w_updated, w_low, w_high) : 1
    dw/dt = w_clipped / ms : 1 (clock-driven)

    w_act_scaled = w_clipped * 2**(6 + w_exp) : 1
    w_act_scaled_shifted = int(floor(w_act_scaled / 2**6)) * 2**6 : 1
    w_act_clipped = clip(w_act_scaled_shifted, -limit, limit) : 1
    dw_act/dt = w_act_clipped / ms : 1 (clock-driven)

    dx0/dt = 0 / ms : 1 (clock-driven)
    dy0/dt = 0 / ms : 1 (clock-driven)
'''
```

Code Listing 3: Weight equations for `Brian`. The first part creates variables that allow terms of the plasticity rule to be evaluated only at the 2^k time step. *dw* contains the user defined learning rule. The updated weight mantissa is adapted depending on the number of weight bits, which determines the precision. The weight mantissa is rounded with *stochastic rounding*. After clipping, the weight mantissa is updated and the actual weight is calculated.

4.2 Results

To demonstrate that the `Loihi` emulator works as expected, we provide three examples covering a single neuron, a recurrently connected spiking neural network, and the application of a learning rule. All three examples are available as `jupyter` notebooks[7].

Neuron model

In a first test, we simulated a single neuron. The neuron receives randomly timed excitatory and inhibitory input spikes. Figure 1A1 shows the synaptic responses induced by the input spikes for the simulation using the `Loihi` chip and the `Brian` emulator. The corresponding voltage traces are shown in Figure 1A2. As expected, the synaptic input as well as the voltage match perfectly between both hardware types.

Network

In a second approach we applied a recurrently connected network of 400 excitatory and 100 inhibitory neurons with log-normal weights. The network gets noisy background input from 40 Poisson generators that are connected to the network with a probability of 0.05. As already shown by others, this setup leads to a highly chaotic behavior (Brunel, 2000; London et al., 2010; Sompolinsky et al., 1988; Van Vreeswijk & Sompolinsky, 1996). Despite the chaotic dynamics, spikes, voltages and synaptic inputs match perfectly

[7] anonymized

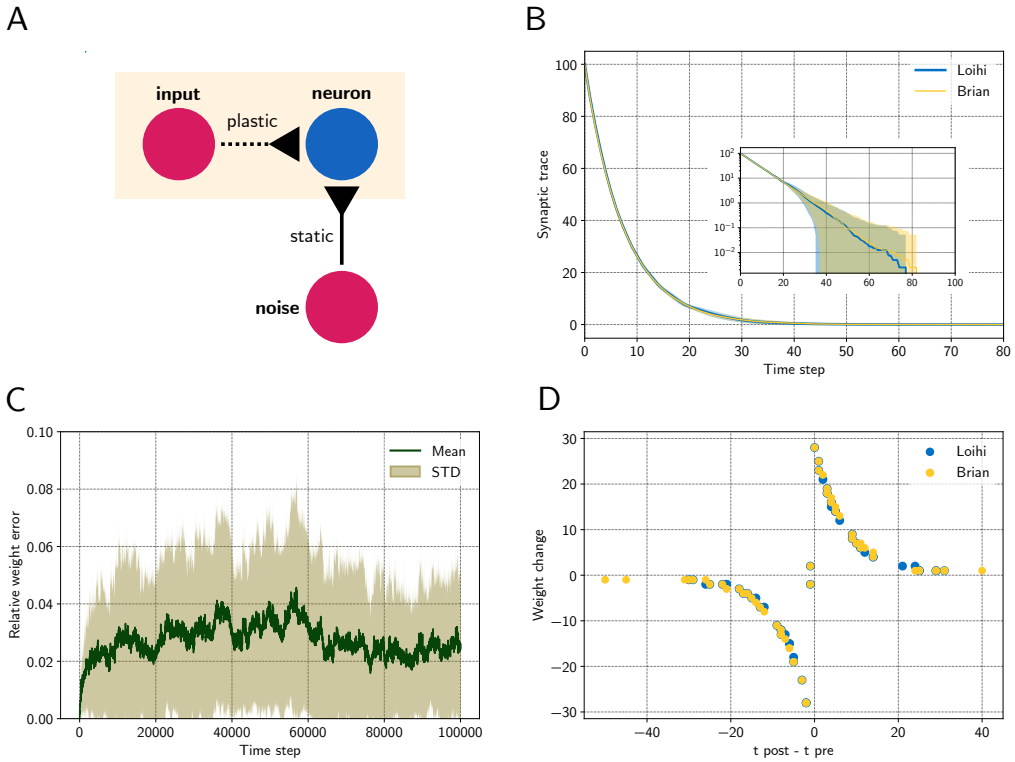


Figure 2: Comparing a STDP learning rule performed with the emulator and with `Loihi`. **A** Sketch showing the setup. **B** Synaptic trace for many trials showing the arithmetic mean and standard deviation. The inset shows the same data in a logarithmic scale. Note that every data point smaller than 10^0 shows the probability of rounding values between 0 and 1 up or down. **C** Relative difference $|\tilde{w}_L - \tilde{w}_B|/\tilde{w}_{\max}$ for the plastic weight between the emulator, $\tilde{w}_B$, and the `Loihi` implementation, $\tilde{w}_L$, for 50 simulations, $\tilde{w}_{\max} = 255$. **D** STDP weight change in respect to pre- and post-synaptic spike times, data shown for time steps $0 - 2000$ for visualisation purposes.

for all neurons and over the whole time. The spiking pattern of the network is shown in Figure 1B. All yellow (`Brian`) and blue (`Loihi`) dots match perfectly.

Learning

In the last experiment, we applied a simple STDP learning rule, as introduced in Equation 25, at a single plastic synapse. The experiment is sketched in Figure 2A. One spike generator, denoted *input*, has a plastic connection to a neuron with a very low weight ($\tilde{w} = 128$, $\Theta = -6$), such that it has a negligible effect on the post-synaptic neuron. Another spike generator, denoted *noise*, has a large but static weight ($\tilde{w} = 254$, $\Theta = 0$) to reliably induce post-synaptic spikes. Figure 2B compares the distribution of traces between the emulator and `Loihi`. For this 400 trials were simulated.

We chose an asymmetric learning window for the STDP rule. The learning rule uses one pre-synaptic trace x_1 ($\hat{x}_1 = 120$, $\tau_{x_1} = 8$) and one post-synaptic trace y_1 ($\hat{y}_1 = 120$, $\tau_{y_1} = 8$). In addition the dependency factors $x_0 \in 0, 1$ and $y_0 \in 0, 1$ are used, which indicate a pre- and post-synaptic spike respectively. Using these components, the learning rule is defined as

$$dw = 2^{-2} \cdot x_1 \cdot y_0 - 2^{-2} \cdot x_0 \cdot y_1. \tag{25}$$

Due to the stochastic rounding of the traces, differences in the weight changes occur, which are shown in Figure 2C. Fortunately, the differences in the weight changes remain on a constant level and do not diverge, even over long simulation times, e.g. 100 000 steps. Despite these variations, the STDP learning window of the emulator reproduces the behavior of the `Loihi` learning window, as shown in Figure 2D.

5 Discussion

This study was motivated by two goals. We hope to simplify the transfer of models to `Loihi` and therefore developed a `Loihi` emulator for `Brian`, featuring many functionalities of the `Loihi` chip. In the process of developing the emulator, we aimed to provide a deeper understanding of the functionality of the neuromorphic research chip `Loihi` by analysing its neuron and synapse model, as well as synaptic plasticity.

We hope that the analysis of `Loihi`'s spiking units has provided some insight into how `Loihi` computes. With the numerical integration method, numerical precision and related rounding method, as well as the update schedule, we were able to walk from the LIF neuron model down to the computations performed. For neurons and networks without plasticity we are able to emulate `Loihi` without error. Analysing and implementing synaptic plasticity showed that, due to stochastic rounding, it is not possible to exactly replicate trial by trial behavior when it comes to learning. However, on average the weight changes induced by a learning rule are preserved.

The main benefit of the `Brian2Loihi` emulator lies in lowering the hurdle for the experimenter. Especially in neuroscience, many scientists are accustomed to neuron simulators and in particular `Brian` is widely used. The emulator can be used for simple and fast prototyping, making a deep dive into new software frameworks and hardware systems unnecessary. In addition, hardware specific complications, like distributing neurons to cores, or constraints like potential limits on the number of available neurons or synapses, or on the speed or size of read-out, do not occur in the emulator. While this will surely improve with new generations of hardware and software in the upcoming years, they can already be ignored by using the emulator.

At this point it is important to note that not all `Loihi` features are included in the emulator, yet. In particular, the homeostasis mechanism, rewards, and tags for the learning rule are not included. In Table 1 we provide a comparison of all functionalities from `Loihi` with those available in the current state of the emulator. Development of this emulator is an open source project and we expect improvements and additions with time.

An important vision for the future is to flexibly connect front-end development environments (e.g. `Brian`, NEST, Keras, TensorFlow) with various back-ends, like neuromorphic platforms (e.g. `Loihi`, SpiNNaker, BrainScaleS, Dynap-SE) or emulators for these platforms. PyNN (Davison et al., 2009) is such an approach to unify different front-ends and back-ends in a more general way. Nengo (Bekolay et al., 2014), as another approach, does not provide the use of other simulators, but allows several back-ends and focuses on higher level applications (DeWolf et al., 2020). NxTF (Rueckauer et al., 2021) is an API and compiler aimed at simplifying the efficient deployment of deep convolutional spiking neural networks on `Loihi` using an interface derived from Keras. We think that ideally, one could continue to work in their preferred front-end environment while a package maps their code to existing chips or computer-based emulators of these chips. We expect an interface along these lines will play an important role in the future of neuromorphic computing and want to contribute to this development with our `Brian2Loihi` emulator.

At least for now, with an emulator at hand, it is easier to prototype network models and assess whether

an implementation on `Loihi` is worth considering. When getting started with neuromorphic hardware, to e.g. scale up models or speed up simulations, researchers familiar with `Brian` can directly deploy models prepared with the emulator. We hope that with this, others may find a smooth entry into the quickly emerging field of neuromorphic computing.

6 Acknowledgements

The work received funds by the Intel Corporation via a gift without restrictions. ABL currently holds a Natural Sciences and Engineering Research Council of Canada PGSD-3 scholarship. We would like to thank Jonas Neuhöfer, Sebastian Schmitt, Andreas Wild, and Terrence C. Stewart for valuable discussions and input.

7 Appendix

7.1 Loihi neuron model

Computational units on `Loihi` communicate via spikes. They can be connected up to form networks, each unit both sending and receiving spikes from some subset of the other units. Like neurons in the brain, a unit emits a spike if its internal variable reaches a certain threshold. The spike is then transmitted to all units with a direct incoming connection from the one that spiked. This induces a change in the receiving units' internal variable. At every time step, the internal variable of all units' decays towards zero, counteracting any input received. And after spiking, the internal variable is reset to zero. In terms of the brain, each computational unit on `Loihi` implements a simple model of a spiking neuron, in particular a variant of the leaky integrate and fire neuron model, which is based on a simple resistor-capacitor (RC) circuit. Readers are encouraged to consult the first chapter of Gerstner et al. (2014) for a more detailed treatment.

7.1.1 Voltage

We refer to the standard leaky integrate and fire neuron, as it is defined in Gerstner et al. (2014). In this model, the difference in electric potential between the interior and the exterior of a neuron, the so-called membrane potential, evolves according to

$$\tau_v \frac{dv}{dt} = -[v(t) - v_{rest}] + RI(t), \tag{26}$$

where v is the voltage across the membrane, τ_v is the membrane time constant of the neuron, v_{rest} is the resting potential, I is the input current and R is the resistance of the membrane. Whenever the membrane potential reaches threshold v_i^{th} it is reset to v_{rest}.

Davies et al. (2018) present the following variant of the standard LIF model which forms the basis for `Loihi`'s computational units

$$\frac{dv_i}{dt} = -\frac{1}{\tau_v} v_i(t) + I_i(t) - v_i^{th}\sigma_i(t), \tag{27}$$

where v is the voltage across the membrane, τ_v is the time constant for voltage decay, I is in this case an input variable, v_i^{th} is the threshold voltage to spike, and $\sigma(t)$ indicates whether the neuron fired a spike at time t.

There are a few differences that are worth noting. In the `Loihi` variant, the resting potential is zero. The membrane time constant τ_v applies only to the voltage decay and not to the input variable I. In effect, the resistance and the time constant are implicit in the input variable I as connection weight.

Further, resetting after a spike is included directly in the differential equation. The final term subtracts the threshold voltage v_i^{th} at the time of a spike. This is a matter of notation and can be written as such, or with a separate reset condition $v_i \to 0$ applied at the time of each spike, as in Gerstner et al. (2014).

7.1.2 Derivation of synaptic response

To keep this paper self-contained, here we derive the synaptic response of a `Loihi` unit to an incoming spike train. For the reader's convenience, we repeat the definition from Equation 3 in the main text here

and then with a few steps obtain the result from Equation 5.

Definition. The *synaptic response* is given by

$$I_i(t) = \sum_j J_{ij}(\alpha_I * \sigma_j)(t) + I_i^{\text{bias}}, \tag{28}$$

where J_{ij} is the weight from unit j to i, I_i^{bias} is a constant bias input, and the spike train σ_j of unit j is convolved with the *synaptic filter impulse response* α_I, given by

$$\alpha_I(t) = \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{29}$$

where τ_I is the time constant of the synaptic response and $H(t)$ the unit step function. Note we define $\alpha_I(t)$ differently here than in Davies et al. (2018) (see Appendix 7.1.3 for details).

Definition. The *unit step function* $H : \mathbb{R} \to \mathbb{R}$ is given by

$$H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0. \end{cases} \tag{30}$$

Definition. The *Dirac delta* is a tempered distribution $\delta \in \mathcal{S}'(\mathbb{R})$, with $\delta : \mathcal{S}(\mathbb{R}) \to \mathbb{C},\ \varphi \mapsto \langle \delta, \varphi \rangle$ where

$$\langle \delta, \varphi \rangle := \int_{-\infty}^{\infty} \delta(x)\varphi(x)\, dx := \varphi(0) \tag{31}$$

for all Schwartz functions $\varphi \in \mathcal{S}(\mathbb{R})$. Here we extend the definition such that $\delta : f \to f(0)$ for arbitrary, everywhere-defined $f : \mathbb{R} \to \mathbb{R}$.

Definition. We define the translation of δ by a, denoted δ_a, as the distribution $\tau_a\delta : \mathcal{S}(\mathbb{R}) \to \mathbb{C}$ with

$$\tau_a\delta(\varphi) := \langle \delta_a, \varphi \rangle = \int_{-\infty}^{\infty} \delta(x - a)\varphi(x)\, dx \tag{32}$$

and again extend this notion to arbitrary, everywhere-defined $f : \mathbb{R} \to \mathbb{R}$.

Lemma 1. (*translation property*) $\tau_a\delta(f) = f(a)$, for $a \in \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$.

Proof. Let $f : \mathbb{R} \to \mathbb{R}$. Then

$$\tau_a\delta(f) = \int_{-\infty}^{\infty} \delta(x - a)f(x)\, dx = \int_{-\infty}^{\infty} \delta(x)f(x + a)\, dx = f(0 + a) = f(a) \tag{33}$$

■

Corollary 1. As a sum of Dirac deltas, σ_i can be understood as the following linear functional

$$\sigma_i := \sum_k \tau_{t_{i,k}}\delta \ :\ \varphi \mapsto \mathbb{C}, \tag{34}$$

with

$$\langle \sigma_i, \varphi \rangle := \langle \sum_k \delta_{t_{i,k}}, \varphi \rangle = \sum_k \langle \delta_{t_{i,k}}, \varphi \rangle = \sum_k \varphi(t_{i,k}), \qquad \varphi \in \mathcal{S}(\mathbb{R}), \tag{35}$$

and again we extend this notion from the space of tempered distributions to σ_i for arbitrary, everywhere-defined f.

Definition. The *convolution* between the Dirac delta distribution and a function is to be understood in the following sense

$$(\delta * f)(x) := \langle \delta, \tau_x\tilde{f} \rangle = \int_{-\infty}^{\infty} \delta(y)f(x - y)\, dy = \int_{-\infty}^{\infty} \delta(x - y)f(y)\, dy \tag{36}$$

where $\tilde{f}(x) = f(-x)$.

Lemma 2. $(\delta * f)(x) = f(x)$.

Proof. Using $\delta(x) = \delta(-x)$ (E) and the translation property of the Dirac delta function (T) from Lemma 1 we have

$$(\delta * f)(x) := \int_{-\infty}^{\infty} \delta(x-y) f(y)\, dy \overset{E}{=} \int_{-\infty}^{\infty} \delta(y-x) f(y)\, dy = \tau_x \delta(f) \overset{T}{=} f(x). \tag{37}$$

■

Claim. The synaptic input $I_i(t)$ for unit i is given by

$$I_i(t) = \sum_j J_{ij} \sum_k \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}) + I_i^{\text{bias}}.$$

Proof. Applying the definition of convolution (D), linearity of the integral operator (L), the translation property of the Dirac delta function (T), and using that $\delta(x) = \delta(-x)$ (E) we have

$$\begin{aligned}
(\alpha_I * \sigma_j)(t) &\overset{D}{=} \int_{-\infty}^{\infty} \alpha_I(s)\, \sigma_j(t-s)\, ds && (38)\\
&= \int_{-\infty}^{\infty} \alpha_I(s) \sum_k \delta(t - t_{j,k} - s)\, ds && (39)\\
&\overset{L}{=} \sum_k \int_{-\infty}^{\infty} \alpha_I(s) \delta(t - t_{j,k} - s)\, ds && (40)\\
&\overset{E}{=} \sum_k \int_{-\infty}^{\infty} \alpha_I(s) \delta(s - (t - t_{j,k}))\, ds && (41)\\
&= \sum_k \tau_{t-t_{j,k}} \delta(\alpha_I) && (42)\\
&\overset{T}{=} \sum_k \alpha_I(t - t_{j,k}) && (43)
\end{aligned}$$

With this, we can write the synaptic input (Equation 3) as

$$\begin{aligned}
I_i(t) &= \sum_j J_{ij} \sum_k \alpha_I(t - t_{i,k}) + I_i^{\text{bias}} && (44)\\
&= \sum_j J_{ij} \sum_k \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}) + I_i^{\text{bias}}. && (45)
\end{aligned}$$

■

We see the input can be written as a sum of exponentially decaying functions with amplitude J_{ij} beginning at the time of each spike $t_{j,k}$.

7.1.3 Definition of the synaptic filter impulse response

Davies et al. (2018) defined the *synaptic filter impulse response* as

$$\alpha_I^{orig}(t) = \frac{1}{\tau_I} \exp\left(-\frac{t}{\tau_I}\right) H(t). \tag{46}$$

Note that we have omitted the factor of $1/\tau_I$ in our definition, in particular we defined

$$\alpha_I(t) = \exp\left(-\frac{t}{\tau_I}\right) H(t). \tag{47}$$

We prefer this formulation as the results obtained match exactly with the `Loihi` documentation. If, however, the factor of $1/\tau_I$ is included, the factor is carried through to Equation 10. Namely it becomes

$$I[t] = I[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + \frac{J}{\tau_I} \cdot s[t] \tag{48}$$

where we see there is an extra factor of $1/\tau_I$ multiplied by the weight J. The definition from Davies et al. (2018) and the `NxSDK` documentation can be reconciled by replacing this extra factor of $1/\tau_I$ with a static factor 2^6 and then considering the weight to be $J = \tilde{w} \cdot 2^{\Theta}$ instead of $J = \tilde{w} \cdot 2^{6+\Theta}$.

7.2 Miscellaneous implementational details

7.2.1 Brian state update schedule

In `Brian` the `network` class is the main class of a simulation. All containing objects like neurons, synapses, monitors, poisson generators, are added to that `network` object. Each of these objects have a *when* attribute. The `network` class decides in which order containing objects are updated depending on their *when* attribute. For this decision a schedule is defined, given as a string list. The default schedule is `['start', 'groups', 'thresholds', 'synapses', 'resets', 'end']`.

We observed that `Loihi` implements a schedule where first the synapses are updated and afterwards the neuron groups. In `Brian` the evaluation is performed in opposite order, which results in a shift between `Loihi` and the emulator. We therefore changed the `Brian` schedule to `['start', 'synapses', 'groups', 'thresholds', 'resets', 'end']`, i.e. the synapse update is pulled in front of groups.

Additionally the time when the synaptic monitor is evaluated is different in `Loihi`. For the emulator, we also needed to adjust these. This is done by changing the monitors *when* flag from the default *start* to *synapses* for the synaptic input and all pre- and post-synaptic trace variables. For probing the voltage and weight the *when* attribute was changed to *end*. The same holds for probing spikes with the spike monitor. Moreover, the poisson generators *when* flag has to be changed from the default *thresholds* to *start* to ensure Poisson spikes are given at the beginning of the current time step and are propagated through the simulation schedule.

7.2.2 Voltage memory used to count refractory time

Note that `Loihi` sets the voltage of a neuron to a non zero value if the neuron has spiked. The memory for storing the voltage is used for counting while the neuron is in refractory state. This causes a deviation between the emulator and `Loihi` for the voltage, which is only due to technical reasons and has no functional effect.

7.3 Plastic weight update with stochastic rounding

If the weight is updated by a learning rule, the weight mantissa needs to be updated according to the given precision, as described in Section 3.1.2. The precision is determined by the available number of bits, which can be chosen by the user, and in addition depends on the sign mode. To test the implementation in the emulator, we compared its behavior to `Loihi` for each possible number of weight bits. In particular, for an excitatory plastic synapse we increased the weight mantissa by one at each time step (via learning rule $dw = u_0$) and measured the actual weight after the update (i.e. rounding and shifting). Our expectation was that for *stochastic rounding* to the nearest 2^{n_s}, the average number of time steps required until a weight change takes place should be equal to 2^{n_s}. This is because the probability of rounding up from a given weight mantissa, e.g. $\tilde{w} := k \cdot 2^{n_s}$, when 1 is added can be calculated from Equation 21 as

$$(|w| - \lfloor |w| \rfloor_{2^{n_s}}))/2^{n_s} = ((k \cdot 2^{n_s} + 1) + k \cdot 2^{n_s})/k \cdot 2^{n_s} = 1/2^{n_s}.$$

As expected, the results match `Loihi`'s behavior nicely, as seen in Figure 3, confirming the validity of our implementation.

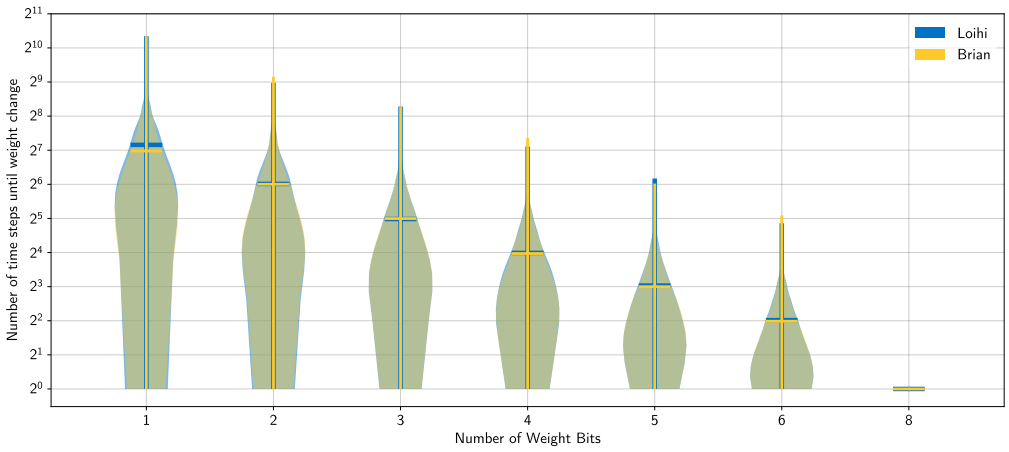


Figure 3: Distribution of the weight change for different number of weight bits. The weight mantissa is increased by 1 in every time step. Due to stochastic rounding, this change may then be rounded up or down. Shown is the distribution of the number of time steps until a weight change occurs. For each number of weight bits, 8000 weight changes were sampled for `Loihi` and the emulator. The emulator implementation matches `Loihi` well.

7.4 Pre- and post-synaptic decay deviations

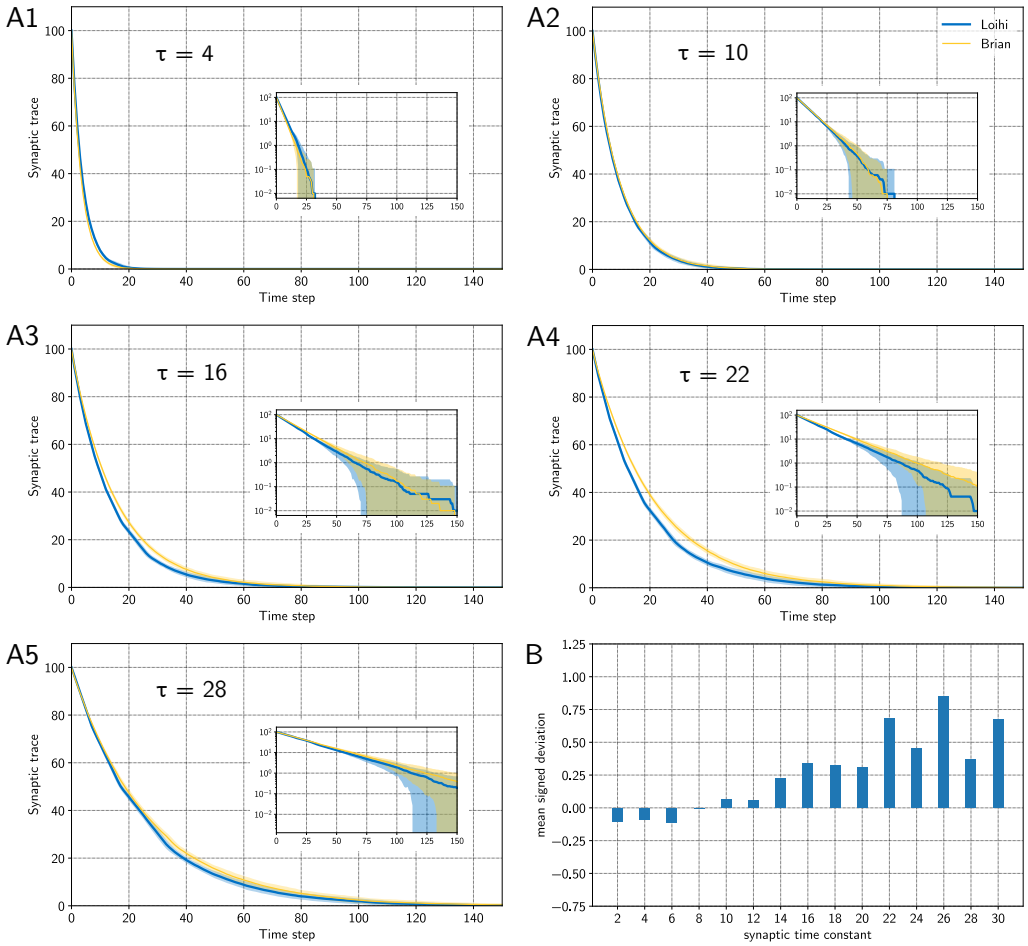


Figure 4: Deviations of the synaptic traces between `Loihi` and the emulator. **A** Synaptic traces for different synaptic time constants τ. Averaged over 100 trials each. The inlay shows the traces in a logarithmic scale. Blue indicates the trace from `Loihi`, yellow the trace from the emulator. **B** Mean signed deviation for different synaptic time constants *tau* over 100 trials each. For low *tau* values, the emulator is slightly below the `Loihi` reference, whereas it lies slightly above the `Loihi` traces for higher values.

7.5 Emulator features

Loihi	**Emulator**
neurons	
current impulse/decay	✓
voltage impulse/decay	✓
bias input	(✓)
homeostasis (threshold adaption)	-
random noise for current	-
random noise for voltage	(✓)
multi-compartment neurons	(✓)
connections	
weight mantissa/exponent	✓
weight precision	✓
synaptic delay	✓
box-synapse	-
learning	
presynaptic spike	✓
1^{st} presynaptic trace	✓
2^{nd} presynaptic trace	✓
postsynaptic spike	✓
1^{st} postsynaptic trace	✓
2^{nd} postsynaptic trace	✓
3^{nd} postsynaptic trace	✓
synaptic weight as variable	✓
reward spike	-
reward trace	-
tag	-
plastic synaptic delay	-
learning epoch	(✓)
probes	
probe variables	✓
probing conditions	(✓)

Table 1: Features of `Loihi` compared with the emulator (version 0.5.2). Check marks in brackets are not fully supported or can manually be included using core `Brian` functionality.

References

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., Choo, X., Voelker, A., & Eliasmith, C. (2014). Nengo: A python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*(48), 1–13. https://doi.org/10.3389/fninf.2013.00048

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., & Beigne, E. (2019). Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *15*(2), 1–35.

Brüderle, D., Petrovici, M. A., Vogginger, B., Ehrlich, M., Pfeil, T., Millner, S., Grübl, A., Wendt, K., Müller, E., Schwartz, M.-O., et al. (2011). A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological cybernetics*, *104*(4), 263–296.

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *Journal of Physiology-Paris*, *94*(5-6), 445–463.

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., … Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. https://doi.org/10.1109/MM.2018.112130359

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., & Risbud, S. R. (2021). Advancing neuromorphic computing with Loihi: A survey of results and outlook. *Proceedings of the IEEE*.

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., & Yger, P. (2009). PyNN: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, *2*, 11.

DeWolf, T., Jaworski, P., & Eliasmith, C. (2020). Nengo and low-power AI hardware for robust, embedded neurorobotics. *Frontiers in Neurorobotics*, *14*.

DeWolf, T., Stewart, T. C., Slotine, J.-J., & Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proceedings of the Royal Society B: Biological Sciences*, *283*(1843), 20162134.

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of neural engineering*, *13*(5), 051001.

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665.

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

Grüning, A., & Bohte, S. M. (2014). Spiking neural networks: Principles and challenges. *ESANN*.

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., & Wang, H. (2018). Programming spiking neural networks on Intel's Loihi. *Computer*, *51*(3), 52–61.

London, M., Roth, A., Beeren, L., Häusser, M., & Latham, P. E. (2010). Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, *466*(7302), 123–127.

Luo, T., Wang, X., Qu, C., Lee, M. K. F., Tang, W. T., Wong, W.-F., & Goh, R. S. M. (2018). An FPGA-based hardware emulator for neuromorphic chip with RRAM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(2), 438–450.

Michaelis, C. (2020). Pelenet: A reservoir computing framework for loihi. *arXiv preprint arXiv:2011.12338*.

Michaelis, C., Lehr, A. B., & Tetzlaff, C. (2020). Robust trajectory generation for robotic control on the neuromorphic research chip loihi. *Frontiers in neurorobotics*, *14*.

Müller, E., Mauch, C., Spilger, P., Breitwieser, O. J., Klähn, J., Stöckel, D., Wunderlich, T., & Schemmel, J. (2020). Extending BrainScaleS OS for BrainScaleS-2. *arXiv preprint arXiv:2003.13750*.

Müller, E., Schmitt, S., Mauch, C., Billaudelle, S., Grübl, A., Güttler, M., Husmann, D., Ilmberger, J., Jeltsch, S., Kaiser, J., et al. (2020). The operating system of the neuromorphic BrainScaleS-1 system. *arXiv preprint arXiv:2003.13749*.

Petrovici, M. A., Vogginger, B., Müller, P., Breitwieser, O., Lundqvist, M., Muller, L., Ehrlich, M., Destexhe, A., Lansner, A., Schüffny, R., et al. (2014). Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PloS one*, *9*(10), e108590.

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, *12*, 774.

Rajendran, B., Sebastian, A., Schmuker, M., Srinivasa, N., & Eleftheriou, E. (2019). Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, *36*(6), 97–110.

Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., Lester, D. R., Mikaitis, M., Plana, L. A., Rowley, A. G., et al. (2018). sPyNNaker: a software package for running PyNN simulations on SpiNNaker. *Frontiers in neuroscience*, *12*, 816.

Rueckauer, B., Bybee, C., Goettsche, R., Singh, Y., Mishra, J., & Wild, A. (2021). NxTF: An API and Compiler for Deep Spiking Neural Networks on Intel Loihi. *arXiv preprint arXiv:2101.04261*.

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., Alvarez-Icaza, R., Amir, A., Arthur, J. V., Andreopoulos, A., et al. (2016). Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications. *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 130–141.

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., & Plank, J. S. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*.

Sompolinsky, H., Crisanti, A., & Sommers, H.-J. (1988). Chaos in random neural networks. *Physical review letters*, *61*(3), 259.

Spilger, P., Müller, E., Emmel, A., Leibfried, A., Mauch, C., Pehle, C., Weis, J., Breitwieser, O., Billaudelle, S., Schmitt, S., et al. (2020). Hxtorch: Pytorch for brainscales-2. *Iot streams for data-driven predictive maintenance and iot, edge, and mobile for embedded machine learning* (pp. 189–200). Springer.

Stagsted, R., Vitale, A., Binz, J., Bonde Larsen, L., Sandamirskaya, Y., et al. (2020). Towards neuromorphic control: A spiking neural network based pid controller for uav.

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator (F. K. Skinner, Ed.). *eLife*, *8*, e47314. https://doi.org/10.7554/eLife.47314

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., Schemmel, J., Wang, R., Chicca, E., Olson Hasler, J., et al. (2018). Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, *12*, 891.

Valancius, S., Richter, E., Purdy, R., Rockowitz, K., Inouye, M., Mack, J., Kumbhare, N., Fair, K., Mixter, J., & Akoglu, A. (2020). FPGA based emulation environment for neuromorphic architectures.

Van Vreeswijk, C., & Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, *274*(5293), 1724–1726.

Young, A. R., Dean, M. E., Plank, J. S., & Rose, G. S. (2019). A review of spiking neuromorphic hardware communication systems. *IEEE Access*, *7*, 135606–135620.

Bibliography

Abbott, L. F., & Nelson, S. B. (2000). Synaptic plasticity: Taming the beast. *3*(S11), 1178–1183. doi:10.1038/81453

Abeles, M. (2009). Synfire chains. *4*(7), 1441. doi:10.4249/scholarpedia.1441

Abeles, M. (2011). Cell assemblies. *Scholarpedia*, *6*(7), 1505. doi:10.4249/scholarpedia.1505

Agnati, L. F., Guidolin, D., Guescini, M., Genedani, S., & Fuxe, K. (2010). Understanding wiring and volume transmission. *64*(1), 137–159. doi:10.1016/j.brainresrev.2010.03.003

Ahmed, K. [Khadeer]. (2021). Brain-inspired spiking neural networks. In *Biomimetics*. doi:10.5772/intechopen.93435

Ahmed, K. [Khaled], & Schuegraf, K. (2011). Transistor wars. *IEEE Spectrum*, *48*(11), 50–66. doi:10.1109/mspec.2011.6056626

Aibara, R., Mitsui, Y., & Ae, T. (1991). A CMOS chip design of binary neural network with delayed synapses. In *1991., IEEE international sympoisum on circuits and systems*. doi:10.1109/iscas.1991.176611

Aimone, J. B., Ho, Y., Parekh, O., Phillips, C. A., Pinar, A., Severa, W., & Wang, Y. (2020). Provable neuromorphic advantages for computing shortest paths. In *Proceedings of the 32nd ACM symposium on parallelism in algorithms and architectures*. doi:10.1145/3350755.3400258

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., ... Modha, D. S. (2015). TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557. doi:10.1109/tcad.2015.2474396

Albouy, G., Sterpenich, V., Balteau, E., Vandewalle, G., Desseilles, M., Dang-Vu, T., ... Maquet, P. (2008). Both the hippocampus and striatum are involved in consolidation of motor sequence memory. *Neuron*, *58*(2), 261–272. doi:10.1016/j.neuron.2008.02.008

Albouy, G., Sterpenich, V., Vandewalle, G., Darsaud, A., Gais, S., Rauchs, G., ... Maquet, P. (2013). Interaction between hippocampal and striatal systems predicts subsequent consolidation of motor sequence memory. *PLoS ONE*, *8*(3), e59490. doi:10.1371/journal.pone.0059490

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, *10*(1-2), 25–61. doi:10.1016/0025-5564(71)90051-4

Anderson, M. L. (2003). Embodied cognition: A field guide. *149*(1), 91–130. doi:10.1016/s0004-3702(03)00054-7

Andres, M., & Pesenti, M. (2015). *Finger-based representation of mental arithmetic.* Oxford University Press.

Araújo, R., Waniek, N., & Conradt, J. (2014). Development of a dynamically extendable SpiNNaker chip computing module. In *Artificial neural networks and machine learning – ICANN 2014* (pp. 821–828). doi:10.1007/978-3-319-11179-7_103

Arber, S., & Costa, R. M. (2018). Connecting neuronal circuits for movement. *Science*, *360*(6396), 1403–1404. doi:10.1126/science.aat5994

Aumann, T. D., & Prut, Y. (2015). Do sensorimotor β-oscillations maintain muscle synergy representations in primary motor cortex? *Trends in Neurosciences*, *38*(2), 77–85. doi:10.1016/j.tins.2014.12.002

Axmacher, N., Mormann, F., Fernandez, G., Cohen, M. X., Elger, C. E., & Fell, J. (2007). Sustained neural activity patterns during working memory in the human medial temporal lobe. *27*(29), 7807–7816. doi:10.1523/jneurosci.0962-07.2007

Backus, J. (1978). Can programming be liberated from the von neumann style?: A functional style and its algebra of programs. *Communications of the ACM*, *21*(8), 613–641. doi:10.1145/359576.359579

Bakken, T., Jorstad, N., Hu, Q., Lake, B., Tian, W., Kalmbach, B., ... Sorensen, S., et al. (2020). Evolution of cellular diversity in primary motor cortex of human, marmoset monkey, and mouse. *BioRxiv*.

Bamford, N. S., Wightman, R. M., & Sulzer, D. (2018). Dopamine's effects on corticostriatal synapses during reward-based behaviors. *97*(3), 494–510. doi:10.1016/j.neuron.2018.01.006

Baras, D., & Meir, R. (2007). Reinforcement Learning, Spike-Time-Dependent Plasticity, and the BCM Rule. *Neural Computation*, *19*(8), 2245–2279. doi:10.1162/neco.2007.19.8.2245. eprint: https://direct.mit.edu/neco/article-pdf/19/8/2245/817088/neco.2007.19.8.2245.pdf

Barinaga, M. (1996). Neuroscience: The cerebellum: Movement coordinator or much more? *272*(5261), 482–483. doi:10.1126/science.272.5261.482

Bastian, A. J. (2006). Learning to predict the future: The cerebellum adapts feedforward movement control. *Current Opinion in Neurobiology*, *16*(6), 645–649. doi:10.1016/j.conb.2006.08.016

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, *60*(6), 503–516. doi:10.1090/s0002-9904-1954-09848-8

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J.-M., ... Boahen, K. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*(5), 699–716. doi:10.1109/jproc.2014.2313565

Benyamini, M., & Zacksenhouse, M. (2015). Optimal feedback control successfully explains changes in neural modulations during experiments with brain-machine interfaces. *Frontiers in Systems Neuroscience*, *9*. doi:10.3389/fnsys.2015.00071

Berlucchi, G., & Buchtel, H. A. (2008). Neuronal plasticity: Historical roots and evolution of meaning. *Experimental Brain Research*, *192*(3), 307–319. doi:10.1007/s00221-008-1611-6

Bhutani, N., Sureshbabu, R., Farooqui, A. A., Behari, M., Goyal, V., & Murthy, A. (2013). Queuing of concurrent movement plans by basal ganglia. *Journal of Neuroscience*, *33*(24), 9985–9997. doi:10.1523/jneurosci.4934-12.2013

Bing, Z., Baumann, I., Jiang, Z., Huang, K., Cai, C., & Knoll, A. (2019). Supervised learning in SNN via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle. *Frontiers in Neurorobotics*, *13*. doi:10.3389/fnbot.2019.00018

Bliss, T. V. P., & Lømo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology*, *232*(2), 331–356. doi:10.1113/jphysiol.1973.sp010273

Blouw, P., Choo, X., Hunsberger, E., & Eliasmith, C. (2019). Benchmarking keyword spotting efficiency on neuromorphic hardware. doi:10.1145/3320288.3320304

Botvinick, M. M. (2008). Hierarchical models of behavior and prefrontal function. *Trends in Cognitive Sciences*, *12*(5), 201–208. doi:10.1016/j.tics.2008.02.009

Bower, J. M. (2013). *20 years of computational neuroscience*. Springer.

Braitenberg, V. (1978). Cell assemblies in the cerebral cortex. (pp. 171–188). doi:10.1007/978-3-642-93083-6_9

Broadbent, D. E. (1975). The magic number seven after fifteen years. *Studies in long term memory*, 3–18.

Brown, S. W. (1995). Time, change, and motion: The effects of stimulus movement on temporal perception. *57*(1), 105–116. doi:10.3758/bf03211853

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *Journal of Physiology-Paris*, *94*(5-6), 445–463. doi:10.1016/s0928-4257(00)01084-6

Buzsáki, G. (2010). Neural syntax: Cell assemblies, synapsembles, and readers. *68*(3), 362–385. doi:10.1016/j.neuron.2010.09.023

Carrillo-Reid, L., Lopez-Huerta, V. G., Garcia-Munoz, M., Theiss, S., & Arbuthnott, G. W. (2015). Cell assembly signatures defined by short-term synaptic plasticity in cortical networks. *25*(07), 1550026. doi:10.1142/s0129065715500264

Cavallari, S., Panzeri, S., & Mazzoni, A. (2014). Comparison of the dynamics of neural interactions between current-based and conductance-based integrate-and-fire recurrent networks. *Frontiers in Neural Circuits*, *8*. doi:10.3389/fncir.2014.00012

Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., & Rose, G. S. (2018). Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *8*(1), 125–136. doi:10.1109/jetcas.2017.2777181

Chen, I. (2005). Behaviorism. (pp. 127–147). doi:10.4018/978-1-59140-555-9.ch019

Chiappalone, M., Massobrio, P., & Martinoia, S. (2008). Network plasticity in cortical assemblies. *28*(1), 221–237. doi:10.1111/j.1460-9568.2008.06259.x

Choi, S., Yang, J., & Wang, G. (2020). Emerging memristive artificial synapses and neurons for energy-efficient neuromorphic computing. *Advanced Materials*, *32*(51), 2004659. doi:10.1002/adma.202004659

Chouinard, P. A., & Paus, T. (2006). The primary motor and premotor areas of the human cerebral cortex. *The Neuroscientist*, *12*(2), 143–152. doi:10.1177/1073858405284255

Chua, L., Sbitnev, V., & Kim, H. (2012). Hodgkin–huxley axon is made of memristors. *International Journal of Bifurcation and Chaos*, *22*(03), 1230011. doi:10.1142/s021812741230011x

Churchland, M. M., Cunningham, J. P., Kaufman, M. T., Ryu, S. I., & Shenoy, K. V. (2010). Cortical preparatory activity: Representation of movement or first cog in a dynamical machine? *68*(3), 387–400. doi:10.1016/j.neuron.2010.09.015

Clements, J. (1996). Transmitter timecourse in the synaptic cleft: Its role in central synaptic function. *19*(5), 163–171. doi:10.1016/s0166-2236(96)10024-2

Courtney, S. M., Ungerleider, L. G., Keil, K., & Haxby, J. V. (1997). Transient and sustained activity in a distributed neural system for human working memory. *386*(6625), 608–611. doi:10.1038/386608a0

Curran, T. (1997). Higher-order associative learning in amnesia: Evidence from the serial reaction time task. *Journal of Cognitive Neuroscience*, *9*(4), 522–533. doi:10.1162/jocn.1997.9.4.522

da Silva, J. A., Tecuapetla, F., Paixão, V., & Costa, R. M. (2018). Dopamine neuron activity before action initiation gates and invigorates future movements. *Nature*, *554*(7691), 244–248. doi:10.1038/nature25457

Dale, H. (1935). Pharmacology and nerve-endings. SAGE Publications.

Dan, Y., & Poo, M.-M. (2006). Spike timing-dependent plasticity: From synapse to perception. *86*(3), 1033–1048. doi:10.1152/physrev.00030.2005

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., ... Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *38*(1), 82–99. doi:10.1109/mm.2018.112130359

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., ... Risbud, S. R. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, *109*(5), 911–934. doi:10.1109/jproc.2021.3067593

Davison, A. P. (2008). PyNN: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, *2*. doi:10.3389/neuro.11.011.2008

Debanne, D., Inglebert, Y., & Russier, M. (2019). Plasticity of intrinsic neuronal excitability. *54*, 73–82. doi:10.1016/j.conb.2018.09.001

DeBole, M. V., Appuswamy, R., Carlson, P. J., Cassidy, A. S., Datta, P., Esser, S. K., ... Nayak, T. K. (2019). TrueNorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, *52*(5), 20–29. doi:10.1109/mc.2019.2903009

Deetjen, P., Speckmann, E.-J., & Hescheler, J. (2005). *Repetitorium physiologie*. Urban & Fischer.

Della-Maggiore, V. (2004). Stimulation of the posterior parietal cortex interferes with arm trajectory adjustments during the learning of new dynamics. *Journal of Neuroscience*, *24*(44), 9971–9976. doi:10.1523/jneurosci.2833-04.2004

Denk, C., Llobet-Blandino, F., Galluppi, F., Plana, L. A., Furber, S., & Conradt, J. (2013). Real-time interface board for closed-loop robotic tasks on the SpiNNaker neural computing system. In *Artificial neural networks and machine learning – ICANN 2013* (pp. 467–474). doi:10.1007/978-3-642-40728-4_59

Desai, N. S., Rutherford, L. C., & Turrigiano, G. G. (1999). Plasticity in the intrinsic excitability of cortical pyramidal neurons. *2*(6), 515–520. doi:10.1038/9165

Desmurget, M., & Turner, R. S. (2010). Motor sequences and the basal ganglia: Kinematics, not habits. *Journal of Neuroscience*, *30*(22), 7685–7690. doi:10.1523/jneurosci.0163-10.2010

Dhawale, A. K., Smith, M. A., & Ölveczky, B. P. (2017). The role of variability in motor learning. *Annual Review of Neuroscience*, *40*(1), 479–498. doi:10.1146/annurev-neuro-072116-031548

Diedrichsen, J., & Kornysheva, K. (2015). Motor skill learning between selection and execution. *Trends in Cognitive Sciences*, *19*(4), 227–233. doi:10.1016/j.tics.2015.02.003

Diedrichsen, J., Shadmehr, R., & Ivry, R. B. (2010). The coordination of movement: Optimal feedback control and beyond. *Trends in Cognitive Sciences*, *14*(1), 31–39. doi:10.1016/j.tics.2009.11.004

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271. doi:10.1007/bf01386390

Doyon, J., Gaudreau, D., Jr., R., Castonguay, M., Bédard, P., Bédard, F., & Bouchard, J.-P. (1997). Role of the striatum, cerebellum, and frontal lobes in the learning of a visuomotor sequence. *Brain and Cognition*, *34*(2), 218–245. doi:10.1006/brcg.1997.0899

Dum, R. P., Li, C., & Strick, P. L. (2002). Motor and nonmotor domains in the monkey dentate. *978*(1), 289–301. doi:10.1111/j.1749-6632.2002.tb07575.x

Dyson, F. W., Eddington, A. S., & Davidson, C. (1920). IX. a determination of the deflection of light by the sun's gravitational field, from observations made at the total eclipse of may 29, 1919. *220*(571-581), 291–333. doi:10.1098/rsta.1920.0009

Ebner, T. J., & Pasalar, S. (2008). Cerebellum predicts the future motor state. *The Cerebellum*, *7*(4), 583–588. doi:10.1007/s12311-008-0059-3

Eccles, J. C. (1976). From electrical to chemical transmission in the central nervous system: The closing address of the sir henry dale centennial symposium cambridge, 19 september 1975. *30*(2), 219–230. doi:10.1098/rsnr.1976.0015

Economo, M. N., Viswanathan, S., Tasic, B., Bas, E., Winnubst, J., Menon, V., ... Svoboda, K. (2018). Distinct descending motor cortex pathways and their roles in movement. *Nature*, *563*(7729), 79–84. doi:10.1038/s41586-018-0642-9

Efnusheva, D., Cholakoska, A., & Tentov, A. (2017). A survey of different approaches for overcoming the processor - memory bottleneck. *International Journal of Computer Science and Information Technology*, *9*(2), 151–163. doi:10.5121/ijcsit.2017.9214

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press.

Emerson, N. B. (2013). *Pele and hiiaka: A myth from hawaii*. Tuttle Publishing.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X. [Xiaowei], et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, *34*, pp. 226–231).

Fan, X., & Markram, H. (2019). A brief history of simulation neuroscience. *Frontiers in Neuroinformatics*, *13*. doi:10.3389/fninf.2019.00032

Finger, S. (2001). *Origins of neuroscience: A history of explorations into brain function.* Oxford University Press, USA.

Fletcher, P., Zafiris, O., Frith, C., Honey, R., Corlett, P., Zilles, K., & Fink, G. (2004). On the benefits of not trying: Brain activity and connectivity reflecting the interactions of explicit and implicit sequence learning. *Cerebral Cortex*, *15*(7), 1002–1015. doi:10.1093/cercor/bhh201

Florian, R. V. [Răzvan V.]. (2007). Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity. *Neural Computation*, *19*(6), 1468–1502. doi:10.1162/neco.2007.19.6.1468. eprint: https://direct.mit.edu/neco/article-pdf/19/6/1468/816938/neco.2007.19.6.1468.pdf

Florian, R. V. [Răzvan V.]. (2008). Tempotron-like learning with ReSuMe. In *Artificial neural networks - ICANN 2008* (pp. 368–375). doi:10.1007/978-3-540-87559-8_38

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, *5*(6), 345. doi:10.1145/367766.368168

Ford, L. R. (1956). *Network flow theory.* Santa Monica, CA: RAND Corporation.

Foster, J., Seth, S., Lokshin, M., & Sajaia, Z. (2013). *A unified approach to measuring poverty and inequality.* doi:10.1596/978-0-8213-8461-9

Frenkel, C., Lefebvre, M., Legat, J.-D., & Bol, D. (2018). A 0.086mm2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS. *IEEE Transactions on Biomedical Circuits and Systems*, 1–1. doi:10.1109/tbcas.2018.2880425

Friedman, R. M. (2005). Einstein and the nobel committee:authority vs. expertise. *36*(4), 129–133. doi:10.1051/epn:2005405

Friedmann, S., Schemmel, J., Grubl, A., Hartel, A., Hock, M., & Meier, K. (2017). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Transactions on Biomedical Circuits and Systems*, *11*(1), 128–142. doi:10.1109/tbcas.2016.2579164

Fu, T., Liu, X., Gao, H., Ward, J. E., Liu, X., Yin, B., ... Yao, J. (2020). Bioinspired bio-voltage memristors. *Nature Communications*, *11*(1). doi:10.1038/s41467-020-15759-y

Fuchs, E., & Flügge, G. (2014). Adult neuroplasticity: More than 40 years of research. *Neural Plasticity*, *2014*, 1–10. doi:10.1155/2014/541870

Fukai, T., Klinshov, V., & Teramae, J.-N. (2014). Cortical networks with lognormal synaptic connectivity and their implications in neuronal avalanches. (pp. 403–416). doi:10.1002/9783527651009.ch19

Furber, S. [Stephen], & Brown, A. (2009). Biologically-inspired massively-parallel architectures - computing beyond a million processors. In *2009 ninth international conference on application of concurrency to system design.* doi:10.1109/acsd.2009.17

Furber, S. [Steve]. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 051001. doi:10.1088/1741-2560/13/5/051001

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The SpiNNaker project. *Proceedings of the IEEE*, *102*(5), 652–665. doi:10.1109/jproc.2014.2304638

Galea, J. M., Vazquez, A., Pasricha, N., de Xivry, J.-J. O., & Celnik, P. (2010). Dissociating the roles of the cerebellum and motor cortex during adaptive learning: The motor cortex retains what the cerebellum learns. *Cerebral Cortex*, *21*(8), 1761–1770. doi:10.1093/cercor/bhq246

Gallese, V., & Lakoff, G. (2005). The brain's concepts: The role of the sensory-motor system in conceptual knowledge. *22*(3-4), 455–479. doi:10.1080/02643290442000310

Gautrais, J., & Thorpe, S. (1998). Rate coding versus temporal order coding: A theoretical approach. *Biosystems*, *48*(1-3), 57–65. doi:10.1016/s0303-2647(98)00050-1

Gentsch, A., Weber, A., Synofzik, M., Vosgerau, G., & Schütz-Bosbach, S. (2016). Towards a common framework of grounded action cognition: Relating motor control, perception and cognition. *146*, 81–89. doi:10.1016/j.cognition.2015.09.010

Gerstner, W., Kistler, W. M., Naud, R., & Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press.

Gewaltig, M.-O., & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, *2*(4), 1430.

Gheysen, F., Opstal, F. V., Roggeman, C., Waelvelde, H. V., & Fias, W. (2010). Hippocampal contribution to early and later stages of implicit motor sequence learning. *Experimental Brain Research*, *202*(4), 795–807. doi:10.1007/s00221-010-2186-6

Ghosh-Dastidar, S., & Adeli, H. (2009). Third generation neural networks: Spiking neural networks. In *Advances in intelligent and soft computing* (pp. 167–178). doi:10.1007/978-3-642-03156-4_17

Gilra, A., & Gerstner, W. (2017). Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network. *6*. doi:10.7554/elife.28295

Glackin, C., McDaid, L., Maguire, L., & Sayers, H. (2008). Implementing fuzzy reasoning on a spiking neural network. In *Artificial neural networks - ICANN 2008* (pp. 258–267). doi:10.1007/978-3-540-87559-8_27

Glickstein, M. (2006). Golgi and cajal: The neuron doctrine and the 100th anniversary of the 1906 nobel prize. *Current Biology*, *16*(5), R147–R151. doi:10.1016/j.cub.2006.02.053

Glimcher, P. W. (2011). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences*, *108*(Supplement 3), 15647–15654. doi:10.1073/pnas.1014269108. eprint: https://www.pnas.org/content/108/Supplement_3/15647.full.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2017). *Deep learning*.

Grafton, S. T., Hazeltine, E., & Ivry, R. B. (2002). Motor sequence learning with the nondominant left hand. *Experimental Brain Research*, *146*(3), 369–378. doi:10.1007/s00221-002-1181-y

Graziano, M. S. A., Yap, G. S., & Gross, C. G. (1994). Coding of visual space by premotor neurons. *266*(5187), 1054–1057. doi:10.1126/science.7973661

Grigoryeva, L., & Ortega, J.-P. (2018). Echo state networks are universal. *108*, 495–508. doi:10.1016/j.neunet.2018.08.025

Gütig, R., & Sompolinsky, H. [Haim]. (2006). The tempotron: A neuron that learns spike timing–based decisions. *Nature Neuroscience*, *9*(3), 420–428. doi:10.1038/nn1643

Guzowski, J. F., Knierim, J. J., & Moser, E. I. (2004). Ensemble dynamics of hippocampal regions CA3 and CA1. *44*(4), 581–584. doi:10.1016/j.neuron.2004.11.003

Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, *61*(4), 5–14. doi:10.1177/0008125619864925

Hamid, A. A., Frank, M. J., & Moore, C. I. (2021). Wave-like dopamine dynamics as a mechanism for spatiotemporal credit assignment. *184*(10), 2733–2749.e16. doi:10.1016/j.cell.2021.03.046

Hebb, D. O. (1949). *The organisation of behaviour: A neuropsychological theory*. Science Editions New York.

Hennequin, G., Vogels, T., & Gerstner, W. (2014). Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron*, *82*(6), 1394–1406. doi:10.1016/j.neuron.2014.04.045

Hennig, M. H. (2013). Theoretical models of synaptic short term plasticity. *7*. doi:10.3389/fncom.2013.00154

Henry, C. (2005). Basal metabolic rate studies in humans: Measurement and development of new equations. *Public Health Nutrition*, *8*(7a), 1133–1152. doi:10.1079/phn2005801

Henry, F. M., & Rogers, D. E. (1960). Increased response latency for complicated movements and a "memory drum" theory of neuromotor reaction. *Research Quarterly. American Association for Health, Physical Education and Recreation*, *31*(3), 448–458. doi:10.1080/10671188.1960.10762052

Herzfeld, D. J., Kojima, Y., Soetedjo, R., & Shadmehr, R. (2015). Encoding of action by the purkinje cells of the cerebellum. *Nature*, *526*(7573), 439–442. doi:10.1038/nature15693

Herzfeld, D. J., Kojima, Y., Soetedjo, R., & Shadmehr, R. (2018). Encoding of error and learning to correct that error by the purkinje cells of the cerebellum. *Nature Neuroscience*, *21*(5), 736–743. doi:10.1038/s41593-018-0136-y

Hikosaka, O., Sakai, K., Miyauchi, S., Takino, R., Sasaki, Y., & Putz, B. (1996). Activation of human presupplementary motor area in learning of sequential procedures: A functional MRI study. *Journal of Neurophysiology*, *76*(1), 617–621. doi:10.1152/jn.1996.76.1.617

Hiratani, N., & Fukai, T. (2014). Interplay between short- and long-term plasticity in cell-assembly formation. *9*(7), e101535. doi:10.1371/journal.pone.0101535

Hodges, A. (2014). *Alan turing: The enigma*. Princeton University Press.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544. doi:10.1113/jphysiol.1952.sp004764

Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. *The Annals of Statistics*, *36*(3). doi:10.1214/009053607000000677

Höppner, S., & Mayr, C. (2018). Spinnaker2-towards extremely efficient digital neuromorphics and multi-scale brain emulation. *Proc. NICE*.

Huberdeau, D. M., Krakauer, J. W., & Haith, A. M. (2015). Dual-process decomposition in human sensorimotor adaptation. *Current Opinion in Neurobiology*, *33*, 71–77. doi:10.1016/j.conb.2015.03.003

Huttenlocher, P. R. (2009). *Neural plasticity*. Harvard University Press.

Ito, M. (2000). Mechanisms of motor learning in the cerebellum. *Brain Research*, *886*(1-2), 237–245. doi:10.1016/s0006-8993(00)03142-5

Izhikevich, E. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*(6), 1569–1572. doi:10.1109/tnn.2003.820440

Izhikevich, E. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070. doi:10.1109/tnn.2004.832719

Izhikevich, E. M. (2007). Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex*, *17*(10), 2443–2452. doi:10.1093/cercor/bhl152. eprint: https://academic.oup.com/cercor/article-pdf/17/10/2443/894946/bhl152.pdf

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, *304*(5667), 78–80. doi:10.1126/science.1091277

Jain, V., Langham, M. C., & Wehrli, F. W. (2010). MRI estimation of global brain oxygen consumption rate. *Journal of Cerebral Blood Flow & Metabolism*, *30*(9), 1598–1607. doi:10.1038/jcbfm.2010.49

Jiang, X., Shen, S., Cadwell, C. R., Berens, P., Sinz, F., Ecker, A. S., ... Tolias, A. S. (2015). Principles of connectivity among morphologically defined cell types in adult neocortex. *Science*, *350*(6264), aac9462–aac9462. doi:10.1126/science.aac9462

Jin, X., Tecuapetla, F., & Costa, R. M. (2014). Basal ganglia subcircuits distinctively encode the parsing and concatenation of action sequences. *Nature Neuroscience*, *17*(3), 423–430. doi:10.1038/nn.3632

Kaas, J. H. (2020). *Evolutionary neuroscience*. Academic Press.

Kaiser, J., Billaudelle, S., Müller, E., Tetzlaff, C., Schemmel, J., & Schmitt, S. (2021). Emulating dendritic computing paradigms on analog neuromorphic hardware. *Neuroscience*. doi:10.1016/j.neuroscience.2021.08.013

Kalaska, J. F. (2009). From intention to action: Motor cortex and the control of reaching movements. In *Advances in experimental medicine and biology* (pp. 139–178). doi:10.1007/978-0-387-77064-2_8

Kandel, E. R., Schwartz, J. H., & Jessell, T. M. (2012). *Principles of neural science*. McGraw-hill New York.

Karas, I. R., & Atila, U. (2011). A genetic algorithm approach for finding the shortest driving time on mobile devices. *Scientific Research and Essays*, *6*(2), 394–405. doi:10.5897/SRE10.896

Kawai, R., Markman, T., Poddar, R., Ko, R., Fantana, A., Dhawale, A., ... Ölveczky, B. (2015). Motor cortex is required for learning but not for executing a motor skill. *Neuron*, *86*(3), 800–812. doi:10.1016/j.neuron.2015.03.024

Koechlin, E., & Jubault, T. (2006). Broca's area and the hierarchical organization of human behavior. *Neuron*, *50*(6), 963–974. doi:10.1016/j.neuron.2006.05.017

Koziol, L. F., Budding, D. E., & Chidekel, D. (2011). From movement to thought: Executive function, embodied cognition, and the cerebellum. *11*(2), 505–525. doi:10.1007/s12311-011-0321-y

Krakauer, J. W., Ghilardi, M.-F., Mentis, M., Barnes, A., Veytsman, M., Eidelberg, D., & Ghez, C. (2004). Differential cortical and subcortical activations in learning rotations and gains

for reaching: A PET study. *Journal of Neurophysiology*, *91*(2), 924–933. doi:10.1152/jn.00675.2003

Krakauer, J. W., Hadjiosif, A. M., Xu, J., Wong, A. L., & Haith, A. M. (2019). Motor learning, 613–663. doi:10.1002/cphy.c170043

Krestinskaya, O., Ibrayev, T., & James, A. P. (2018). Hierarchical temporal memory features with memristor logic circuits for pattern recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(6), 1143–1156. doi:10.1109/tcad.2017.2748024

Krestinskaya, O., James, A. P., & Chua, L. O. (2020). Neuromemristive circuits for edge computing: A review. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(1), 4–23. doi:10.1109/tnnls.2019.2899262

Kriener, B., Enger, H., Tetzlaff, T., Plesser, H. E., Gewaltig, M.-O., & Einevoll, G. T. (2014). Dynamics of self-sustained asynchronous-irregular activity in random networks of spiking neurons with strong synapses. *8*. doi:10.3389/fncom.2014.00136

Kulvicius, T., Herzog, S., Tamosiunaite, M., & Worgotter, F. (2021). Finding optimal paths using networks without learning–unifying classical approaches. *IEEE Transactions on Neural Networks and Learning Systems*, 1–11. doi:10.1109/tnnls.2021.3089023

Kumawat, S., Dudeja, C., & Kumar, P. (2021). An extensive review of shortest path problem solving algorithms. In *2021 5th international conference on intelligent computing and control systems (ICICCS)*. doi:10.1109/iciccs51141.2021.9432275

Kuzum, D., Yu, S., & Wong, H.-S. P. (2013). Synaptic electronics: Materials, devices and applications. *Nanotechnology*, *24*(38), 382001. doi:10.1088/0957-4484/24/38/382001

Ladenbauer, J., McKenzie, S., English, D. F., Hagens, O., & Ostojic, S. (2019). Inferring and validating mechanistic models of neural microcircuits based on spike-train data. *Nature Communications*, *10*(1). doi:10.1038/s41467-019-12572-0

Laje, R., & Buonomano, D. V. (2013). Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature Neuroscience*, *16*(7), 925–933. doi:10.1038/nn.3405

Lakoff, G., & Johnson, M. (2008). *Metaphors we live by*. University of Chicago press.

Lazar, A. (2009). *Self-organizing recurrent neural networks.* (Doctoral dissertation, Citeseer).

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*. doi:10.3389/fnins.2016.00508

Lefort, S., Tomm, C., Sarria, J.-C. F., & Petersen, C. C. (2009). The excitatory neuronal network of the c2 barrel column in mouse primary somatosensory cortex. *61*(2), 301–316. doi:10.1016/j.neuron.2008.12.020

Legenstein, R., Pecevski, D., & Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLOS Computational Biology*, *4*(10), 1–27. doi:10.1371/journal.pcbi.1000180

Levy, N., Horn, D., Meilijson, I., & Ruppin, E. (2001). Distributed synchrony in a cell assembly of spiking neurons. *14*(6-7), 815–824. doi:10.1016/s0893-6080(01)00044-2

Li, Y., Lu, H., Cheng, P.-l., Ge, S., Xu, H., Shi, S.-H., & Dan, Y. (2012). Clonally related visual cortical neurons show similar stimulus feature selectivity. *486*(7401), 118–121. doi:10.1038/nature11110

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., & Wang, H. (2018). Programming spiking neural networks on intel's loihi. *Computer*, *51*(3), 52–61. doi:10.1109/mc.2018.157113521

Liu, D., Yu, H., & Chai, Y. (2020). Low-power computing with neuromorphic engineering. *Advanced Intelligent Systems*, *3*(2), 2000150. doi:10.1002/aisy.202000150

Liu, S.-C., Delbruck, T., Indiveri, G., Whatley, A., & Douglas, R. (2014). *Event-based neuromorphic systems*. John Wiley & Sons.

London, M., Roth, A., Beeren, L., Häusser, M., & Latham, P. E. (2010). Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, *466*(7302), 123–127. doi:10.1038/nature09086

Long, L., & Fang, G. (2010). A review of biologically plausible neuron models for spiking neural networks. In *AIAA infotech@aerospace 2010*. doi:10.2514/6.2010-3540

Lukoševičius, M. (2012). A practical guide to applying echo state networks. (pp. 659–686). doi:10.1007/978-3-642-35289-8_36

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *3*(3), 127–149. doi:10.1016/j.cosrev.2009.03.005

Luo, Y., Zhang, H., Wang, Y., Wen, Y., & Zhang, X. (2018). ResumeNet: A learning-based framework for automatic resume quality assessment. In *2018 IEEE international conference on data mining (ICDM)*. doi:10.1109/icdm.2018.00046

Ma, D., Shen, J., Gu, Z., Zhang, M., Zhu, X., Xu, X., ... Pan, G. (2017). Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *Journal of Systems Architecture*, *77*, 43–51. doi:10.1016/j.sysarc.2017.01.003

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. doi:10.1016/s0893-6080(97)00011-7

Maass, W., & Markram, H. (2004). On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, *69*(4), 593–616. doi:10.1016/j.jcss.2004.04.001

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, *14*(11), 2531–2560. doi:10.1162/089976602760407955

Magzhan, K., & Jani, H. M. (2013). A review and evaluations of shortest path algorithms. *International journal of scientific & technology research*, *2*(6), 99–104.

Malecki, U., Stallforth, S., Heipertz, D., Lavie, N., & Duzel, E. (2009). Neural generators of sustained activity differ for stimulus-encoding and delay maintenance. *30*(5), 924–933. doi:10.1111/j.1460-9568.2009.06871.x

Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., ... Saria, A. (2011). Introducing the human brain project. *Procedia Computer Science*, *7*, 39–42. doi:10.1016/j.procs.2011.12.015

Marr, D., & Thach, W. T. [W. Thomas]. (1991). A theory of cerebellar cortex. In *From the retina to the neocortex* (pp. 11–50). doi:10.1007/978-1-4684-6775-8_3

Martin, A., & Nystrom, M. (2006). Asynchronous techniques for system-on-chip design. *Proceedings of the IEEE*, *94*(6), 1089–1120. doi:10.1109/jproc.2006.875789

Mazzoni, P., Hristova, A., & Krakauer, J. W. (2007). Why don't we move faster? parkinson's disease, movement vigor, and implicit motivation. *Journal of Neuroscience*, *27*(27), 7105–7116. doi:10.1523/jneurosci.0264-07.2007

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. doi:10.1007/bf02478259

McDougle, S. D., Bond, K. M., & Taylor, J. A. (2015). Explicit and implicit processes constitute the fast and slow processes of sensorimotor learning. *Journal of Neuroscience*, *35*(26), 9568–9579. doi:10.1523/jneurosci.5061-14.2015

Mead, C. (1980). Introduction to vlsi systems. *IEE Proceedings I-Solid-State and Electron Devices*, *128*(1), 18.

Mead, C., & Ismail, M. (Eds.). (1989). *Analog VLSI implementation of neural systems*. doi:10.1007/978-1-4613-1639-8

Medina, J. F. (2011). The multiple roles of purkinje cells in sensori-motor calibration: To predict, teach and command. *Current Opinion in Neurobiology*, *21*(4), 616–622. doi:10.1016/j.conb.2011.05.025

Meier, K. (2015). A mixed-signal universal neuromorphic computing system. In *2015 IEEE international electron devices meeting (IEDM)*. doi:10.1109/iedm.2015.7409627

Miall, R. C., Christensen, L. O. D., Cain, O., & Stanley, J. (2007). Disruption of state estimation in the human lateral cerebellum. *PLoS Biology*, *5*(11), e316. doi:10.1371/journal.pbio.0050316

Miall, R., & Wolpert, D. (1996). Forward models for physiological motor control. *Neural Networks*, *9*(8), 1265–1279. doi:10.1016/s0893-6080(96)00035-4

Michaelis, C. (2020). Pelenet: A reservoir computing framework for loihi. *arXiv preprint arXiv:2011.12338*.

Michaelis, C., Lehr, A. B., Oed, W., & Tetzlaff, C. (2021). Brian2loihi: An emulator for the neuromorphic chip loihi using the spiking neural network simulator brian. arXiv: 2109.12308 [cs.NE]

Michaelis, C., Lehr, A. B., & Tetzlaff, C. (2020). Robust trajectory generation for robotic control on the neuromorphic research chip loihi. *Frontiers in Neurorobotics*, *14*. doi:10.3389/fnbot.2020.589532

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, *63*(2), 81–97. doi:10.1037/h0043158

Milner, P. M. (1957). The cell assembly: Mark II. *64*(4), 242–252. doi:10.1037/h0042287

Minkowski, H. (1908). Die grundgleichungen für die elektromagnetischen vorgänge in bewegten körpern. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, *1908*, 53–111.

Mohan, H., Verhoog, M. B., Doreswamy, K. K., Eyal, G., Aardse, R., Lodder, B. N., ... de Kock, C. P. (2015). Dendritic and axonal architecture of individual pyramidal neurons across layers of adult human neocortex. *Cerebral Cortex*, *25*(12), 4839–4853. doi:10.1093/cercor/bhv188

Montessori, M. (1959). *The absorbent mind*. lulu.

Moradi, S., Qiao, N., Stefanini, F., & Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Transactions on Biomedical Circuits and Systems*, *12*(1), 106–122. doi:10.1109/tbcas.2017.2759700

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., & Masquelier, T. (2019). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *94*, 87–95. doi:10.1016/j.patcog.2019.05.015

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., & Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *29*(12), 6178–6190. doi:10.1109/tnnls.2018.2826721

Nakajima, T., Hosaka, R., Mushiake, H., & Tanji, J. (2009). Covert representation of second-next movement in the pre-supplementary motor area of monkeys. *Journal of Neurophysiology*, *101*(4), 1883–1889. doi:10.1152/jn.90636.2008

Nakamura, K., Sakai, K., & Hikosaka, O. (1998). Neuronal activity in medial frontal cortex during learning of sequential procedures. *Journal of Neurophysiology*, *80*(5), 2671–2687. doi:10.1152/jn.1998.80.5.2671

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., ... Boahen, K. (2019). Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, *107*(1), 144–164. doi:10.1109/jproc.2018.2881432

Neumann, P. E., & Gest, T. R. (2019). How many bones? every bone in my body. *33*(2), 187–191. doi:10.1002/ca.23425

Nguyen, D.-A., Tran, X.-T., & Iacopi, F. (2021). A review of algorithms and hardware implementations for spiking neural networks. *Journal of Low Power Electronics and Applications*, *11*(2), 23. doi:10.3390/jlpea11020023

Nimmo, H. A. (1986). Pele, ancient goddess of contemporary hawaii. *Pacific Studies*, *9*(2), 121.

Nissen, M. J., & Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, *19*(1), 1–32. doi:10.1016/0010-0285(87)90002-8

Overduin, S. A., d'Avella, A., Roh, J., & Bizzi, E. (2008). Modulation of muscle synergy recruitment in primate grasping. *Journal of Neuroscience*, *28*(4), 880–892. doi:10.1523/jneurosci.2869-07.2008

Overduin, S., d'Avella, A., Carmena, J., & Bizzi, E. (2012). Microstimulation activates a handful of muscle synergies. *Neuron*, *76*(6), 1071–1077. doi:10.1016/j.neuron.2012.10.018

Paillard, J. (1976). Réflexions sur l'usage du concept de plasticité en neurobiologie. *Journal de Psychologie Normale et Pathologique*, (1), 33–47.

Panda, P., & Roy, K. (2017). Learning to generate sequences with combination of hebbian and non-hebbian plasticity in recurrent spiking neural networks. *11*. doi:10.3389/fnins.2017.00693

Paninski, L. (2004). Superlinear population encoding of dynamic hand trajectory in primary motor cortex. *Journal of Neuroscience*, *24*(39), 8551–8561. doi:10.1523/jneurosci.0919-04.2004

Patriarchi, T., Cho, J. R., Merten, K., Howe, M. W., Marley, A., Xiong, W.-H., ... Tian, L. (2018). Ultrafast neuronal imaging of dopamine dynamics with designed genetically encoded sensors. *360*(6396), eaat4422. doi:10.1126/science.aat4422

Pavlov, I. P. (1927). Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex. Oxford University Press, London. A PHYSIOLOGICAL MODEL OF PHOBIC ANXIETY A.

Pawlak, V. (2010). Timing is not everything: Neuromodulation opens the STDP gate. *Frontiers in Synaptic Neuroscience*, *2*. doi:10.3389/fnsyn.2010.00146

Pehlevan, C., Ali, F., & Ölveczky, B. P. (2018). Flexibility in motor timing constrains the topology and dynamics of pattern generator circuits. *Nature Communications*, *9*(1). doi:10.1038/s41467-018-03261-5

Peracchia, C. (1977). Gap junction structure and function. *2*(2), 26–31. doi:10.1016/0968-0004(77)90251-1

Perlin, K. (1985). An image synthesizer. *19*(3), 287–296. doi:10.1145/325165.325247

Pesaran, B., Nelson, M. J., & Andersen, R. A. (2006). Dorsal premotor neurons encode the relative position of the hand, eye, and goal during reach planning. *51*(1), 125–134. doi:10.1016/j.neuron.2006.05.025

Pignatelli, M., & Bonci, A. (2015). Role of dopamine neurons in reward and aversion: A synaptic plasticity perspective. *86*(5), 1145–1157. doi:10.1016/j.neuron.2015.04.015

Ponulak, F. (2006). Supervised learning in spiking neural networks with resume method. *Phd, Poznan University of Technology*, *46*, 47.

Ponulak, F., & Hopfield, J. J. (2013). Rapid, parallel path planning by propagating wavefronts of spiking neural activity. *7*. doi:10.3389/fncom.2013.00098

Ponulak, F., & Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. *Neural Computation*, *22*(2), 467–510. doi:10.1162/neco.2009.11-08-901

Purves, D., Augustine, G., Fitzpatrick, D., Hall, W., Lamantia, A.-S., & White, L. (2013). *Neuroscience. 5th edition*. PMC3584490[pmcid]. YJBM. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3584490/

Pyle, R., & Rosenbaum, R. (2019). A reservoir computing model of reward-modulated motor learning and automaticity. *Neural Computation*, *31*(7), 1430–1461. doi:10.1162/neco_a_01198

Qureshi, A., Rizvi, F., Syed, A., Shahid, A., & Manzoor, H. (2014). The method of loci as a mnemonic device to facilitate learning in endocrinology leads to improvement in student performance as measured by assessments. *38*(2), 140–144. doi:10.1152/advan.00092.2013

Raichle, M. E., & Gusnard, D. A. (2002). Appraising the brain's energy budget. *Proceedings of the National Academy of Sciences*, *99*(16), 10237–10239. doi:10.1073/pnas.172399499

Rajendran, B., Sebastian, A., Schmuker, M., Srinivasa, N., & Eleftheriou, E. (2019). Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, *36*(6), 97–110. doi:10.1109/msp.2019.2933719

Rand, M. K., Hikosaka, O., Miyachi, S., Lu, X., & Miyashita, K. (1998). Characteristics of a long-term procedural skill in the monkey. *Experimental Brain Research*, *118*(3), 293–297. doi:10.1007/s002210050284

Rice, M., Patel, J., & Cragg, S. (2011). Dopamine release in the basal ganglia. *198*, 112–137. doi:10.1016/j.neuroscience.2011.08.066

Rosenbaum, D. A., Kenny, S. B., & Derr, M. A. (1983). Hierarchical control of rapid movement sequences. *Journal of Experimental Psychology: Human Perception and Performance*, *9*(1), 86–102. doi:10.1037/0096-1523.9.1.86

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. doi:10.1037/h0042519

Rossant, C., Goodman, D. F. M., Fontaine, B., Platkiewicz, J., Magnusson, A. K., & Brette, R. (2011). Fitting neuron models to spike trains. *Frontiers in Neuroscience*, *5*. doi:10.3389/fnins.2011.00009

Roy, B. (1959). Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences*, *249*(2), 216–218.

Ruchkin, D. S., Grafman, J., Cameron, K., & Berndt, R. S. (2003). Working memory retention systems: A state of activated long-term memory. *26*(6), 709–728. doi:10.1017/s0140525x03000165

Rullen, R. V., & Thorpe, S. J. (2001). Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex. *Neural Computation*, *13*(6), 1255–1283. doi:10.1162/08997660152002852

Sakai, K., Hikosaka, O., Miyauchi, S., Takino, R., Sasaki, Y., & Pütz, B. (1998). Transition of brain activation from frontal to parietal areas in visuomotor sequence learning. *The Journal of Neuroscience*, *18*(5), 1827–1840. doi:10.1523/jneurosci.18-05-01827.1998

Sakai, K., Kitaguchi, K., & Hikosaka, O. (2003). Chunking during human visuomotor sequence learning. *Experimental Brain Research*, *152*(2), 229–242. doi:10.1007/s00221-003-1548-8

Saladin, K. S. (2017). *Anatomy & physiology* (5th ed.). WCB/McGraw-Hill Education New York.

Salari, E., Freudenburg, Z. V., Vansteensel, M. J., & Ramsey, N. F. (2018). Spatial-temporal dynamics of the sensorimotor cortex: Sustained and transient activity. *26*(5), 1084–1092. doi:10.1109/tnsre.2018.2821058

Saleh, M., Takahashi, K., & Hatsopoulos, N. G. (2012). Encoding of coordinated reach and grasp trajectories in primary motor cortex. *Journal of Neuroscience*, *32*(4), 1220–1232. doi:10.1523/jneurosci.2438-11.2012

Savtchenko, L. P., & Rusakov, D. A. (2007). The optimal height of the synaptic cleft. *104*(6), 1823–1828. doi:10.1073/pnas.0606636104

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., ... Modha, D. S. (2016). TrueNorth ecosystem for brain-inspired computing: Scalable systems, software, and applications. In *SC16: International conference for high performance computing, networking, storage and analysis*. doi:10.1109/sc.2016.11

Saxena, V. (2021). Neuromorphic computing: From devices to integrated circuits. *Journal of Vacuum Science & Technology B*, *39*(1), 010801. doi:10.1116/6.0000591

Schandry, R. (2006). *Biologische psychologie*.

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., & Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE international symposium on circuits and systems*. doi:10.1109/iscas.2010.5536970

Schendan, H. E., Searl, M. M., Melrose, R. J., & Stern, C. E. (2003). An fMRI study of the role of the medial temporal lobe in implicit and explicit sequence learning. *Neuron*, *37*(6), 1013–1025. doi:10.1016/s0896-6273(03)00123-5

Schofield, W. N. (1985). Predicting basal metabolic rate, new standards and review of previous work. *Human nutrition. Clinical nutrition*, *39*, 5–41.

Schrauwen, B., Verstraeten, D., & Van Campenhout, J. (2007). An overview of reservoir computing: Theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007* (pp. 471–482).

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *42*(3), 1–21. doi:10.1145/3068335

Schultz, W., & Romo, R. (1992). Role of primate basal ganglia and frontal cortex in the internal generation of movements. *91*(3). doi:10.1007/bf00227834

Schuman, C. D., Birdwell, J. D., Dean, M., Plank, J., & Rose, G. (2016). Neuromorphic computing: A post-moore's law complementary architecture. In *International workshop on post-moore's era supercomputing (pmes)*, Salt Lake City, UT.

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., & Plank, J. S. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*.

Scott, S. H. (2004). Optimal feedback control and the neural basis of volitional motor control. *Nature Reviews Neuroscience*, *5*(7), 532–545. doi:10.1038/nrn1427

Seo, J.-s., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., ... Friedman, D. J. (2011). A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *2011 IEEE custom integrated circuits conference (CICC)*. doi:10.1109/cicc.2011.6055293

Shadmehr, R. (1997). Neural correlates of motor memory consolidation. *Science*, *277*(5327), 821–825. doi:10.1126/science.277.5327.821

Shapiro, L. (2019). *Embodied cognition*. Routledge.

Shen, J., Ma, D., Gu, Z., Zhang, M., Zhu, X., Xu, X., ... Pan, G. (2015). Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *Science China Information Sciences*, *59*(2), 1–5. doi:10.1007/s11432-015-5511-7

Sherrington, C. S. (1897). The central nervous system. *A text book of Physiology*, *929*.

Shmuelof, L., Huang, V. S., Haith, A. M., Delnicki, R. J., Mazzoni, P., & Krakauer, J. W. (2012). Overcoming motor "forgetting" through reinforcement of learned actions. *Journal of Neuroscience*, *32*(42), 14617–14621a. doi:10.1523/jneurosci.2184-12.2012

Singh, R. E., Iqbal, K., White, G., & Hutchinson, T. E. (2018). A systematic review on muscle synergies: From building blocks of motor behavior to a neurorehabilitation tool. *Applied Bionics and Biomechanics*, *2018*, 1–15. doi:10.1155/2018/3615368

Sjöström, J., & Gerstner, W. (2010). Spike-timing dependent plasticity. *5*(2), 1362. doi:10.4249/scholarpedia.1362

Smith, M. A., Ghazizadeh, A., & Shadmehr, R. (2006). Interacting adaptive processes with different timescales underlie short-term motor learning. *PLoS Biology*, *4*(6), e179. doi:10.1371/journal.pbio.0040179

Sockol, M. D., Raichlen, D. A., & Pontzer, H. (2007). Chimpanzee locomotor energetics and the origin of human bipedalism. *Proceedings of the National Academy of Sciences*, *104*(30), 12265–12269. doi:10.1073/pnas.0703267104

Sokolov, A. A., Miall, R. C., & Ivry, R. B. (2017). The cerebellum: Adaptive prediction for movement and cognition. *21*(5), 313–332. doi:10.1016/j.tics.2017.02.005

Sompolinsky, H. [H.], Crisanti, A., & Sommers, H. J. (1988). Chaos in random neural networks. *Physical Review Letters*, *61*(3), 259–262. doi:10.1103/physrevlett.61.259

Song, S., Miller, K. D., & Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *3*(9), 919–926. doi:10.1038/78829

Song, S., Sjöström, P. J., Reigl, M., Nelson, S., & Chklovskii, D. B. (2005). Highly nonrandom features of synaptic connectivity in local cortical circuits. *3*(3), e68. doi:10.1371/journal.pbio.0030068

Sossin, W. S., Sweet-Cordero, A., & Scheller, R. H. (1990). Dale's hypothesis revisited: Different neuropeptides derived from a common prohormone are targeted to different processes. *87*(12), 4845–4848. doi:10.1073/pnas.87.12.4845

Spalla, D., Cornacchia, I. M., & Treves, A. (2021). Continuous attractors for dynamic memories. *10*. doi:10.7554/elife.69499

Spicker, P. (2012). Why refer to poverty as a proportion of median income? *20*(2), 163–175. doi:10.1332/175982712x652069

Spreizer, S., Aertsen, A., & Kumar, A. (2019). From space to time: Spatial inhomogeneities lead to the emergence of spatiotemporal sequences in spiking neuronal networks. *15*(10), e1007432. doi:10.1371/journal.pcbi.1007432

Stainer, M. J., Carpenter, R., Brotchie, P., & Anderson, A. J. (2016). Sequences show rapid motor transfer and spatial translation in the oculomotor system. *Vision Research*, *124*, 1–6. doi:10.1016/j.visres.2016.06.002

Sternberg, S., Monsell, S., Knoll, R. L., & Wright, C. E. (1978). The latency and duration of rapid movement sequences: Comparisons of speech and typewriting. In *Information processing in motor control and learning* (pp. 117–152). doi:10.1016/b978-0-12-665960-3.50011-6

Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife*, *8*, e47314. doi:10.7554/eLife.47314

Sussillo, D., Churchland, M. M., Kaufman, M. T., & Shenoy, K. V. (2015). A neural network that finds a naturalistic solution for the production of muscle activity. *Nature Neuroscience*, *18*(7), 1025–1033. doi:10.1038/nn.4042

Svoboda, K., & Li, N. (2018). Neural mechanisms of movement planning: Motor cortex and beyond. *49*, 33–41. doi:10.1016/j.conb.2017.10.023

Sylva, K. (1997). Critical periods in childhood learning. *British Medical Bulletin*, *53*(1), 185–197. doi:10.1093/oxfordjournals.bmb.a011599

Taherkhani, A., Belatreche, A., Li, Y., Cosma, G., Maguire, L. P., & McGinnity, T. (2020). A review of learning in biologically plausible spiking neural networks. *Neural Networks*, *122*, 253–272. doi:10.1016/j.neunet.2019.09.036

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2015a). DL-ReSuMe: A delay learning-based remote supervised method for spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(12), 3137–3149. doi:10.1109/tnnls.2015.2404938

Taherkhani, A., Belatreche, A., Li, Y., & Maguire, L. P. (2015b). Multi-DL-ReSuMe: Multiple neurons delay learning remote supervised method. In *2015 international joint conference on neural networks (IJCNN)*. doi:10.1109/ijcnn.2015.7280743

Tanaka, H. (2016). Modeling the motor cortex: Optimality, recurrent neural networks, and spatial dynamics. *Neuroscience Research*, *104*, 64–71. doi:10.1016/j.neures.2015.10.012

Tanaka, H., & Sejnowski, T. J. (2013). Computing reaching dynamics in motor cortex with cartesian spatial coordinates. *Journal of Neurophysiology*, *109*(4), 1182–1201. doi:10.1152/jn.00279.2012

Tang, G., Shah, A., & Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional SLAM. doi:10.1109/iros40897.2019.8967864

Tansey, E. (1997). Not committing barbarisms: Sherrington and the synapse, 1897. *Brain Research Bulletin*, *44*(3), 211–212. doi:10.1016/s0361-9230(97)00312-2

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural Networks*, *111*, 47–63. doi:10.1016/j.neunet.2018.12.002

Taylor, J. A., Krakauer, J. W., & Ivry, R. B. (2014). Explicit and implicit contributions to learning in a sensorimotor adaptation task. *Journal of Neuroscience*, *34*(8), 3023–3032. doi:10.1523/jneurosci.3619-13.2014

Teramae, J.-n., & Fukai, T. (2014). Computational implications of lognormally distributed synaptic weights. *102*(4), 500–512. doi:10.1109/jproc.2014.2306254

Teramae, J.-n., Tsubo, Y., & Fukai, T. (2012). Optimal spike-based communication in excitable networks with strong-sparse and weak-dense links. *2*(1). doi:10.1038/srep00485

Tetzlaff, C., Dasgupta, S., Kulvicius, T., & Wörgötter, F. (2015). The use of hebbian cell assemblies for nonlinear computation. *5*(1). doi:10.1038/srep12866

Tetzlaff, C., Kolodziejski, C., Timme, M., Tsodyks, M., & Wörgötter, F. (2013). Synaptic scaling enables dynamically distinct short- and long-term memory formation. *9*(10), e1003307. doi:10.1371/journal.pcbi.1003307

Tetzlaff, C., Kolodziejski, C., Timme, M., & Wörgötter, F. (2011). Synaptic scaling in combination with many generic plasticity mechanisms stabilizes circuit connectivity. *5*. doi:10.3389/fncom.2011.00047

Thach, W. T. [W. T.]. (1996). On the specific role of the cerebellum in motor learning and cognition: Clues from PET activation and lesion studies in man. *19*(03), 411–433. doi:10.1017/s0140525x00081504

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., ... Etienne-Cummings, R. (2018). Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in Neuroscience*, *12*. doi:10.3389/fnins.2018.00891

Thura, D., & Cisek, P. (2017). The basal ganglia do not select reach targets but control the urgency of commitment. *Neuron*, *95*(5), 1160–1170.e5. doi:10.1016/j.neuron.2017.07.039

Todorov, E. (2004). Optimality principles in sensorimotor control. *Nature Neuroscience*, *7*(9), 907–915. doi:10.1038/nn1309

Todorov, E., & Jordan, M. I. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neuroscience*, *5*(11), 1226–1235. doi:10.1038/nn963

Treisman, M., Faulkner, A., & Naish, P. L. N. (1992). On the relation between time perception and the timing of motor action: Evidence for a temporal oscillator controlling the timing of movement. *45*(2), 235–263. doi:10.1080/14640749208401326

Truccolo, W., Friehs, G. M., Donoghue, J. P., & Hochberg, L. R. (2008). Primary motor cortex tuning to intended movement kinematics in humans with tetraplegia. *Journal of Neuroscience*, *28*(5), 1163–1178. doi:10.1523/jneurosci.4415-07.2008

Tschentscher, N., Hauk, O., Fischer, M. H., & Pulvermüller, F. (2012). You can count on the motor cortex: Finger counting habits modulate motor cortex activation evoked by numbers. *59*(4), 3139–3148. doi:10.1016/j.neuroimage.2011.11.037

Tseng, Y.-w., Diedrichsen, J., Krakauer, J. W., Shadmehr, R., & Bastian, A. J. (2007). Sensory prediction errors drive cerebellum-dependent adaptation of reaching. *Journal of Neurophysiology*, *98*(1), 54–62. doi:10.1152/jn.00266.2007

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, *2*(1), 230–265.

Turrigiano, G. G. (2008). The self-tuning neuron: Synaptic scaling of excitatory synapses. *135*(3), 422–435. doi:10.1016/j.cell.2008.10.008

Turrigiano, G. G., & Nelson, S. B. (2004). Homeostatic plasticity in the developing nervous system. *5*(2), 97–107. doi:10.1038/nrn1327

Vakil, E., Kahan, S., Huberman, M., & Osimani, A. (2000). Motor and non-motor sequence learning in patients with basal ganglia lesions: The case of serial reaction time (SRT). *Neuropsychologia*, *38*(1), 1–10. doi:10.1016/s0028-3932(99)00058-5

van Rossum, M. C. W., Bi, G. Q., & Turrigiano, G. G. (2000). Stable hebbian learning from spike timing-dependent plasticity. *20*(23), 8812–8821. doi:10.1523/jneurosci.20-23-08812.2000

van Vreeswijk, C., & Sompolinsky, H. [H.]. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, *274*(5293), 1724–1726. doi:10.1126/science.274.5293.1724

Verhaeghen, P., & Marcoen, A. (1996). On the mechanisms of plasticity in young and older adults after instruction in the method of loci: Evidence for an amplification model. *11*(1), 164–178. doi:10.1037/0882-7974.11.1.164

Verwey, W. B. (1996). Buffer loading and chunking in sequential keypressing. *Journal of Experimental Psychology: Human Perception and Performance*, *22*(3), 544–562. doi:10.1037/0096-1523.22.3.544

Vincent-Lamarre, P., Calderini, M., & Thivierge, J.-P. (2020). Learning long temporal sequences in spiking networks by multiplexing neural oscillations. *Frontiers in Computational Neuroscience*, *14*. doi:10.3389/fncom.2020.00078

Vittoz, E. A. (1996). The fundamentals of analog micropower design. *Tutorial*, *7*, 365–372.

von Neumann, J. (1993). First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, *15*(4), 27–75. doi:10.1109/85.238389

Waldeyer, W. (1891). Ueber einige neuere forschungen im gebiete der anatomie des centralnervensystems (fortsetzung aus no. 44.) *DMW - Deutsche Medizinische Wochenschrift*, *17*(45), 1244–1246. doi:10.1055/s-0029-1206842

Wang, J., Belatreche, A., Maguire, L. P., & McGinnity, T. M. (2011). A new learning algorithm for adaptive spiking neural networks. In *Neural information processing* (pp. 461–468). doi:10.1007/978-3-642-24955-6_55

Warshall, S. (1962). A theorem on boolean matrices. *Journal of the ACM*, *9*(1), 11–12. doi:10.1145/321105.321107

Wehr, M., & Zador, A. M. (2003). Balanced inhibition underlies tuning and sharpens spike timing in auditory cortex. *426*(6965), 442–446. doi:10.1038/nature02116

Will, B., Dalrymple-Alford, J., Wolff, M., & Cassel, J.-C. (2008). The concept of brain plasticity—paillard's systemic analysis and emphasis on structure and function (followed by the translation of a seminal paper by paillard on plasticity). *Behavioural Brain Research*, *192*(1), 2–7. doi:10.1016/j.bbr.2007.11.030

Xu-Wilson, M., Chen-Harris, H., Zee, D. S., & Shadmehr, R. (2009). Cerebellar contributions to adaptive control of saccades in humans. *Journal of Neuroscience*, *29*(41), 12930–12939. doi:10.1523/jneurosci.3115-09.2009

Wolpert, D. M. [Daniel M.], & Ghahramani, Z. (2000). Computational principles of movement neuroscience. *Nature Neuroscience*, *3*(S11), 1212–1217. doi:10.1038/81497

Wolpert, D. M. [Daniel M], Miall, R., & Kawato, M. (1998). Internal models in the cerebellum. *Trends in Cognitive Sciences*, *2*(9), 338–347. doi:10.1016/s1364-6613(98)01221-2

Wong, A. L., Haith, A. M., & Krakauer, J. W. (2014). Motor planning. *The Neuroscientist*, *21*(4), 385–398. doi:10.1177/1073858414541484

Wong, A. L., Lindquist, M. A., Haith, A. M., & Krakauer, J. W. (2015). Explicit knowledge enhances motor vigor and performance: Motivation versus practice in sequence tasks. *Journal of Neurophysiology*, *114*(1), 219–232. doi:10.1152/jn.00218.2015

Wu, H. G., Miyamoto, Y. R., Castro, L. N. G., Ölveczky, B. P., & Smith, M. A. (2014). Temporal structure of motor variability is dynamically regulated and predicts motor learning ability. *Nature Neuroscience*, *17*(2), 312–321. doi:10.1038/nn.3616

Xu, X. [Xiangmin], Olivas, N. D., Ikrar, T., Peng, T., Holmes, T. C., Nie, Q., & Shi, Y. (2016). Primary visual cortex shows laminar-specific and balanced circuit organization of excitatory and inhibitory synaptic connectivity. *The Journal of Physiology*, *594*(7), 1891–1910. doi:10.1113/jp271891

Yamazaki, T., & Igarashi, J. (2013). Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit. *Neural Networks*, *47*, 103–111. doi:10.1016/j.neunet.2013.01.019

Yantis, S., Schwarzbach, J., Serences, J. T., Carlson, R. L., Steinmetz, M. A., Pekar, J. J., & Courtney, S. M. (2002). Transient neural activity in human parietal cortex during spatial attention shifts. *5*(10), 995–1002. doi:10.1038/nn921

Yeom, H. G., Kim, J. S., & Chung, C. K. (2020). Brain mechanisms in motor control during reaching movements: Transition of functional connectivity according to movement states. *Scientific Reports*, *10*(1). doi:10.1038/s41598-020-57489-7

Yokoi, A., Arbuckle, S. A., & Diedrichsen, J. (2018). The role of human primary motor cortex in the production of skilled finger sequences. *The Journal of Neuroscience*, *38*(6), 1430–1442. doi:10.1523/jneurosci.2798-17.2017

Yokoi, A., & Diedrichsen, J. (2019). Neural organization of hierarchical motor sequence representations in the human neocortex. *Neuron*, *103*(6), 1178–1190.e7. doi:10.1016/j.neuron.2019.06.017

Young, A. R., Dean, M. E., Plank, J. S., & Rose, G. S. (2019). A review of spiking neuromorphic hardware communication systems. *IEEE Access*, *7*, 135606–135620. doi:10.1109/access.2019.2941772

Yttri, E. A., & Dudman, J. T. (2016). Opponent and bidirectional control of movement velocity in the basal ganglia. *Nature*, *533*(7603), 402–406. doi:10.1038/nature17639

Zenke, F., Agnes, E. J., & Gerstner, W. (2015). Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *6*(1). doi:10.1038/ncomms7922

Zenke, F., & Gerstner, W. (2017). Hebbian plasticity requires compensatory processes on multiple timescales. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *372*(1715), 20160259. doi:10.1098/rstb.2016.0259

Zhang, L. I., Tao, H. W., Holt, C. E., Harris, W. A., & Poo, M.-m. (1998). A critical window for cooperation and competition among developing retinotectal synapses. *395*(6697), 37–44. doi:10.1038/25665

Zhang, S., & Zhang, Y. (2018). A hybrid genetic and ant colony algorithm for finding the shortest path in dynamic traffic networks. *Automatic Control and Computer Sciences*, *52*(1), 67–76. doi:10.3103/s014641161801008x

Zhang, T., Jia, S., Cheng, X., & Xu, B. (2021). Tuning convolutional spiking neural network with biologically plausible reward propagation, 1–11. doi:10.1109/tnnls.2021.3085966

Zhang, W., & Linden, D. J. (2003). The other side of the engram: Experience-driven changes in neuronal intrinsic excitability. *4*(11), 885–900. doi:10.1038/nrn1248

Zhu, J., Zhang, T., Yang, Y., & Huang, R. (2020). A comprehensive review on emerging artificial neuromorphic devices. *Applied Physics Reviews*, *7*(1), 011312. doi:10.1063/1.5118217

Zollikofer, C. P. E., de León, M. S. P., Lieberman, D. E., Guy, F., Pilbeam, D., Likius, A., ... Brunet, M. (2005). Virtual cranial reconstruction of sahelanthropus tchadensis. *Nature*, *434*(7034), 755–759. doi:10.1038/nature03397

Zucker, R. S., & Regehr, W. G. (2002). Short-term synaptic plasticity. *64*(1), 355–405. doi:10.1146/annurev.physiol.64.092501.114547

Curriculum Vitae

Personal information

Name:	Carlo Michaelis
Date of birth:	1989-09-29
Place of birth:	Dieburg, Germany
Email:	carlo.michaelis@gmail.com

Education

2006 - 2009	High school, Landrat-Gruber-Schule Dieburg Advanced courses: information technology & electrotechnology
2010 - 2014	Psychology, Bachelor of Science, University of Leipzig Thesis: "Explizite Detektion von Abweichungen in komplexen tonalen Regeln" at Cognitive and Biological Psychology, University of Leipzig
2011 - 2015	Physics, Bachelor of Science, University of Leipzig Thesis: "Intrinsische Rauschunterdückung bei LCMV Beamformer und Minimum Norm Estimate bei Magnetenzephalographie" at MPI for Human Cognitive and Brain Sciences, Leipzig
2015 - 2018	Statistics, Master of Science, Humboldt University of Berlin Thesis: "Does intrinsic plasticity prefer regularity? A Markov chain Monte Carlo perspective on a self-organizing recurrent neural network" at Third Institute of Physics - Biophysics, University of Göttingen
2018 - 2021	PhD student, University of Göttingen Program: International Max Planck Research School for Physics of Biological and Complex Systems