Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Felix Schneider

born in Heilbronn, Germany

June 2018

Towards Spike–based Expectation Maximization in a Closed–Loop Setup on an Accelerated Neuromorphic Substrate

This Bachelor Thesis has been carried out by Felix Schneider at the
Kirchhoff Institute for Physics in Heidelberg
under the supervision of
Prof. Dr. Karlheinz Meier

Abstract

Learning experiments are in general time–consuming and computationally expensive if executed on conventional computing machines. In contrast to application–specific circuits on neuromorphic hardware like the BrainScaleS system, which can emulate Spiking Neural Networks fast and efficient. A possible model for a learning experiment is called *Spike-based Expectation Maximization* (SEM) – A population of neurons tries to find the hidden cause of spike patterns in an unsupervised manner. It is possible to transfer this approach to Leaky-Integrate-and-Fire (LIF) neurons which make an implementation on state-of-the-art neuromorphic hardware like the BrainScaleS system possible. The most promising approach is to use a closed–loop operation mode which enables real–time communication between the neuromorphic part and a host computer, forming both systems into a hybrid system. We will present in this thesis necessary mechanisms for SEM experiments which were implemented and verified on the BrainScaleS system: Weight adaptation from the host computer during emulation and a homeostatic rate adaptation using Sea-of-Noise networks as an on–wafer spike source. These presented experiments are the first using the closed–loop operation mode on the current hardware version HICANN v4.

Zusammenfassung

Lernexperimente sind im allgemeinen zeitaufwendig und rechenintensiv, wenn sie auf herkömmlichen Rechenmaschinen ausgeführt werden. Im Gegensatz zu anwendungsspezifischen Schaltkreisen auf neuromorpher Hardware wie dem BrainScaleS System, welches feuernde neuronale Netze schnell und effizient emmulieren kann. Ein mögliches Modell für solch ein Lernexperiment heißt Spike-basierte Erwartungswertmaximierung (SEM) – eine Population von Neuronen versucht die darunterliegende Struktur in Spikemustern zu finden. Dieser Ansatz lässt sich auf Leaky-Integrate-and-Fire Neuronen übertragen, was eine Implementation auf dem BrainScaleS System möglich macht. Der meist versprechende Ansatz ist es dabei einen closed–loop Operationsmodus zu nutzen, welcher eine Echtzeit Kommunikation zwischen der neuromorphen Hardware und einem Kontrollcomputer ermöglicht und beide System zu einem hybriden System kombiniert. Wir werden in dieser Arbeit nötige Mechanismen für die Durchführung von SEM Experimentn auf dem BrainScaleS System präsentieren. Diese wurden auf diesem System implementiert und getestet: Eine Gewichtsanpassung ausgelöst von einem Kontrollcomputer sowie eine homeostatische Ratenanpassung mit einem Sea-of-Noise Netzwerk als eine auf dem System befindliche Spikequelle. Diese präsentierten Experimente stellen zudem die ersten closed–loop Experimente auf der aktuellen Hardware Version (HICANN v4) dar.

Contents

1. Introduction

With approximately 10^{11} neurons, is the human brain one of the most complex objects in the universe. The European Flagship project *Human Brain Project* has set itself a goal of a comprehensive understanding of the human brain. Experts from diverse disciplines including neuroscience, physics, philosophy and computer science work together to achieve the primary objectives of the project: Create and operate a scientific research infrastructure for brain research, gather and organise data describing the brain, simulate the brain, build multi–scale theory and models of the brain, develop brain–inspired computing & robotics (Amunts and Ebell [2017]).

The *Electronic Vision(s) Group* from the Kirchhoff–Institute in Heidelberg is a partner of this project in the subproject *Neuromorphic Computing*. The goals are to build neuromorphic hardware to model a substantial part of the brain and to survey novel computer paradigms. We use for this task very–large–scale integration (VLSI) to create analog circuits which implement the Adaptive Exponential integrate and fire neuron model (AdEx). The system is called *BrainScaleS* system and realises about four million neurons and 880 million synapses on 20 wafers, each with 384 chips which are called *HICANN* – High Input Count Analog Neural Network (Schemmel et al. [2010]). The time constants of the circuit components are several orders of magnitude lower than the time constants of their biological counterparts. This leads to an acceleration of the system by a speed–up factor of about 10^4 compared to biology, allowing a study of temporal changes in neuronal networks which are not feasible with conventional simulators.

A promising field of use for neuromorphic hardware are closed–loop setups. Conventional closed–loop setups are well established in robotics where motor controllers influence the behaviour of the robot based on the current state of its environment. These controllers have some drawbacks – they are often computationally intensive and are hard to tune. Bio–inspired controllers could solve these problems and also integrate learning features to make these platforms adaptable to new tasks (Perez-Peña et al. [2017]). In the recent past, it was shown that self–driving robots which receive sensory input from its surrounding through a spike–based visual system are feasible. The data was then processed by IBMs TrueNorth system (Fischl et al. [2017]).

The BrainScaleS system supports different operation modes. One of these modes is the *closed–loop* mode where we combine a host computer and the neuromorphic hardware to form a hybrid system . The host computer receives information about the behaviour of the neuronal network on the BrainScaleS system during emula-

tion. Based on these information the host can influence the emulation by sending spikes back to the system or adjust the synaptic weight of a connection between neurons. Because of the massive speed–up of the neuromorphic part, this process is challenging. The calculations on the host as well as the communications between the two systems have to keep up with the acceleration – forming it into a *real–time* system where the computations have to fulfil timing constraints. A possible application for this closed–loop operation mode is a *Spike–Based Expectation Maximization* (SEM) experiment: A group of neurons which are organised in different layers receive structured input in form of spike trains and try to find the hidden cause of these presented spike patterns.

We will give a short introduction into this topic and will present in detail the necessary mechanisms for such an experiment which where developed and characterised in this thesis. These mechanisms are a homeostatic rate adaptation of a neuron as well as a weight adaptation in the closed–loop operation mode. We will conclude with an outlook of a closed–loop implementation of a SEM experiment on the BrainScaleS system.

2. The Neuromorphic System

In this chapter the BrainScaleS system will be introduced. All measurements during this thesis have been carried out on the Neuromorphic Computing Platform NM–PM1 located in Heidelberg. This platform uses VLSI analog circuits on interconnected chips to emulate spiking neural networks. We will start with a description of a single neuron like they are implemented on hardware and afterwards with a short introduction to the BrainScaleS system itself.

2.1. Neuron Model

2.1.1. Leaky-Integrate-and-Fire model

Many different neuron models have been developed in the field of neuroscience. Each with a different degree on biological accuracy and computational complexity. Complex models like the *Hodgin–Huxley* model describe the underlying ionic mechanisms which lead to a action potential very well, on the other hand is this model described by four ordinary differential equations and it is computationally intensive to analyse larger populations of neurons with this model. One of the simplest neuron models with scientific relevance is the *Leaky-Integrate-and-Fire* model (LIF). The model can be described by an ordinary differential equation.

$$C_m \frac{du}{dt} = g_l \left(E_L - u\right) + I^{syn} + I^{ext} \tag{2.1}$$

Where u describes the membrane potential of a neuron, C_m the membrane capacity. The terms $\dot{u} \propto I^{syn/ext}$ represent the integration of the input and the $\dot{u} \propto -u$ the *leaky* term which discharges the neurons membrane. The discharge current I results from the conductivity g_L and drives the membrane potential towards the resting potential E_L. The model itself introduces no spiking behaviour, this can be included with a threshold rule.

$$u(t_{spike}) = \vartheta \tag{2.2}$$

If the membrane potential is above a defined threshold, the neuron spikes. A spike is then represented by a delta peak $\delta(t - t_{spike})$ and the membrane potential is kept

at a resting potential ϱ for a refractory period τ_{ref}.

$$u(t_{spike} < t < t_{spike} + \tau_{raf}) = \varrho \tag{2.3}$$

A spike will then be sent to all post connected neurons as synaptic input. This input can be described by equation 2.4 where we distinguish between excitatory and inhibitory synaptic input.

$$I^{syn} = I^{exc./inh.} = \sum_i g_i^{exc./inh.}(u - E_{exc./inh.}) \tag{2.4}$$

2.1.2. Adaptive Exponential Integrate-and-Fire model

The neurons implemented on the BrainScaleS system are an extended version of LIF neurons, namely *Adaptive Exponential Integrate-and-Fire* (AdEx) neurons. In this model the spike generation term 2.5 is added to the LIF equation 2.1.

$$I^{exp} = g_L \Delta_T \exp\left(\frac{u - E_T}{\Delta_T}\right) \tag{2.5}$$

Here E_T acts similar to the threshold ϑ introduced in 2.2. If u crosses E_T from below, I_{exp} dominates the membrane dynamics. But in this case we do not have a hard threshold which always leads to a spike. External input I^{ext} can counteract with a negative contribution and prohibit spiking. Furthermore we introduce the adaptive variable w. The dynamics of w itself are also described by an ordinary linear differential equation. The AdEx–model can then be described by:

$$C_m \frac{du}{dt} = g_L(E_l - u) + g_L \Delta_T \exp\left(\frac{u - E_T}{\Delta_T}\right) - w + I^{syn} + I^{ext} \tag{2.6}$$

$$\tau_w \frac{dw}{dt} = a(u - E_L) + b\tau_w \rho - w. \tag{2.7}$$

With the additional variable w, the AdEx–model is able to emulate firing patterns like tonic spiking, adaptation or regular bursting. The time constant τ_w defines how fast w decays to zero. The variable ρ is here the neurons own spike train. The strength of the influence of w on the membrane potential can be adjusted with the variables a and b. After a spike, the membrane potential is kept again at a resting potential ϱ for a refractory time τ_{ref} as introduced for the LIF – model in 2.3 and w is set to $w = w + b$. For every spike the neuron receives from other neurons, the current is increased. How much it is increased depends on the activation function.

In case of the BrainScaleS system this is an exponential activation as shown in 2.8.

$$I^{syn}(t) = -\sum_i g_i^{syn}(u(t) - E_{syn}) \tag{2.8}$$

$$g_i^{syn}(t) = w_{ij} \sum_{t^s} \exp\left(\frac{-t - t^s}{\tau_{syn}}\right) \Theta(t - t^s) \tag{2.9}$$

Where g_i^{syn} is the conductance of the i-th synapse, w_{ij} represents the weight between neuron i and j, $\Theta(t)$ is the Heaviside function and E_{syn} the reversal potential. A rigorous introduction into this topic and a detailed explanation of the effects on the spiking behaviour of the parameters can be found in Petrovici [2015].

2.2. The BrainScaleS System

2.2.1. The HICANN–Chip

To simulate a neuron or in particular a network of neurons on an usual computing architecture, the described differential equations 2.6 have to be solved numerically. This becomes on usual van–Neumann architectures especially for larger networks computationally expensive and time consuming. This is different on an architecture like the BrainScaleS System. The system is a mixed–signal neuromorphic platform with analog neuromorphic circuits implemented. The differential equation of the AdEx model can then be represented by these electrical circuits which will evolve according to the laws of physics. The central part of BrainScaleS system are the *High Input Count Analog Neural Network* (HICANN) chips. On each chip are 512 neuron circuits realized. Compared to biological neurons, the time constants of the circuit components are several orders of magnitude smaller. This leads to a speed–up of 10^4 compared to biological time. The analog parameters of this model can be stored in floating gate memory cells which are charged or discharged compared to a reference voltage between 0 V and 1.8 V. The neurons receive spike input from two blocks of synapses and are arranged in two rows of 256 neuron circuits. Each circuit can receive spike input through two synaptic inputs. Each of those synaptic inputs is connected to 220 synapses which transmit spike events. The synaptic strength of those connections can be varied with digital weights. These digital weights have a 4 bit resolution and can represent 16 different possible weights. Each with a different strength – a digital weight of 0 means no connection, 15 is the strongest connection. There also exist two analog readout lines per HICANN which can be used to record voltage traces of two different neurons with the analog to digtal converters (ADCs).

2.2.2. The Wafer–Scale Hardware System

To build larger networks, several HICANN chips are connected with each other. In conventional chip production single chips are cut out of a silicon wafer, in this case the whole wafer is kept in tact and all HICANNs are connected with each other on top of the wafer. This process is called *wafer–scale integration*. The whole BrainScaleS system consists of 20 wafers, each with 384 HICANN chips. Repeaters on the wafer can recover spike signals and send them to other neurons on the wafer. External spikes which are sent to a HICANN chip are controlled by *Field Programmable Gate Arrays* (FPGAs). One FPGA regulates eight HICANNs – 48 FPGAs per wafer manage the communication between host computer and wafer. Figure 2.1 shows a simplified overview of the system. The FPGAs receive information about the experiment like configuration data from the host computer and configures the wafer system correspondingly. The user defines the networks topology with the PyNN language. The PyNN API enables the use of a high–level of abstraction to describe the neuronal network for the experiment. A binary representation of this network (PyHMF container) will be used from the mapping tool marocco which maps the network onto hardware. The output after this mapping process is also a binary – the StHAL container. The access to the hardware is then done with the hardware abstraction layer backend HALbe. A more precise introduction into this topic can be found in Jeltsch [2014].

The BrainScaleS system supports three different operation modes (see Müller [2014]). The first is the batch mode where single experiments are emulated on the hardware. External spike input is calculated before emulation and buffered in the FPGAs which can store 312 Mega Events (MEv) where one spike corresponds to an event. In the interactive or iterative mode several sequential experiments are emulated. The system is configured in the first step but the hardware resources stays allocated the whole time because of dependencies of the sequential experiment steps. The last mode is the closed–loop operation mode. In this mode the system interacts with an external environment with a direct communication to the host. This last operation mode is used in this thesis and will be explained in more detail in Chapter 3.3.

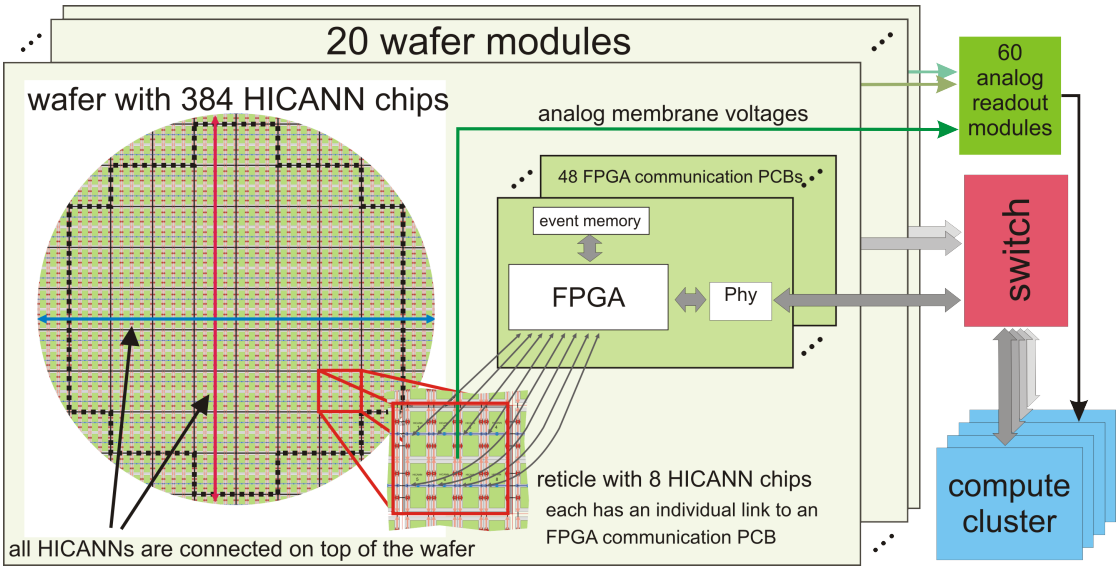


Figure 2.1.: Scheme of a simplified overview of the BrainScaleS system. On each wafer are 384 HICANN chips realized which are connected with each other. One FPGA handles eight HICANN chips, each with a connection to a FPGA communication PCB. The host computers are connected via a switch with the 48 FPGAs. Furthermore the membrane voltages of a neuron can be readout with one of the 60 analog readout modules.

3. Methods

This chapter will start with a short introduction into the topic *Spike–Based Expectation Maximization*. Afterwards the principles of real–time computing and closed–loop experiments will be explained and how these experiments can be performed on the BrainScaleS system. Finally, we will discuss necessary mechanisms for Spike Based Expectation Maximization in a closed–loop setup: homeostasis and Sea–of–Noise networks to introduce stochastic.

3.1. Spike Based Expectation Maximization

The goal in a Spike–Based Expectation Maximization (SEM) experiment, is to find the hidden cause in patterns which neurons receive as input in form of spike trains in an unsupervised manner. We will start with an explanation of the used model – A Restricted Boltzmann Machine. Subsequently, we will look into a method how this type of model can be trained and how it can be implement with spiking neural networks.

3.1.1. Boltzmann Machine

A Boltzmann Machine is a probabilistic model over binary random variables $\mathbf{Z}$=(Z_1, Z_2, ...). These Variables are represented by interconnected units, in our case neurons, with an intrinsic tendency to be active (Z_i=1) or inactive (Z_i=0). This intrinsic tendency is represented by a bias b_i. The interconnection between the units is realized with weights W_{ij}. The probability distribution for a state $\mathbf{z}$ with the normalization constant Z is given as

$$p(\mathbf{z}) = \frac{1}{Z} \cdot exp\left[\sum_{i,j} \frac{1}{2} z_i W_{ij} z_j + \sum_k b_k Z_k\right]. \tag{3.1}$$

$$Z = \sum_z exp\left[\sum_{i,j} \frac{1}{2} z_i W_{ij} z_j + \sum_k b_k Z_k\right] \tag{3.2}$$

If we define an energy E we can simplify our probability distribution

$$E(\mathbf{z}) = -\left[\sum_{i,j} \frac{1}{2} z_i W_{ij} z_j + \sum_k b_k Z_k\right] \tag{3.3}$$

$$p(\mathbf{z}) = \frac{1}{Z} e^{-E(\mathbf{z})}. \tag{3.4}$$

In a *Restricted Boltzmann Machine* (RBM), units are organised into a stack of layers, each having no internal connection and only neighbouring layers are connected to each other. For a two layer RBM we can write the energy function as shown in equation 3.5 below.

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i,j} v_i W_{ij} h_j - \sum_i a_i v_i - \sum_j b_j h_j \tag{3.5}$$

In the two layer case, the first layer is called the visible layer (v), the second layer the hidden layer (h). The bias of the visible units is called a_i here. For the expectation-maximization algorithm we also need the conditional probability, which is for v given h defined as

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1} p(v_i|\mathbf{h}) \tag{3.6}$$

3.1.2. Expectation Maximization

A method to find maximum likelihood solutions for models with hidden variables is the expectation-maximization algorithm. Given a certain distribution $f(x)$ and a data set, the likelihood function represents the likeliness of different parameters of $f(x)$. We want to maximize the likelihood function with respect to the parameters θ. This algorithm consists of two steps. The first step – the Expectation Step – uses the current parameters θ^{cur} which are initialized randomly to evaluate the a–posteriori distribution $p(\mathbf{z}|\mathbf{y}, \theta^{cur})$. In the maximization step we want to find a new set of parameters θ^{new}. This is done by finding the expectation of the complete-data log likelihood for some general θ. The new parameters θ^{new} are the parameters which maximizes the log likelihood.

$$\theta^{new} = \arg\max_{\theta} \sum_z p(\mathbf{z}|\mathbf{y}, \theta^{cur}) \ln p(\mathbf{y}, \mathbf{z}|\theta) \tag{3.7}$$

These two steps are repeated until convergence of the log likelihood or the parameters. This is the basic idea behind the EM–algorithm, for an introduction into this topic see Bishop [2006].

It can be shown that using a form of *spike timing dependent plasticity* (STDP) neurons can perform a stochastic version of expectation-maximization [Nessler et al. [2009]]. This is called *Spike Based Expectation Maximization*. The topology for such an experiment can be seen in Figure 3.1. Stochastic neurons (blue) receive input in form of spike trains from the input layer (red). The stochastic neurons are strongly inhibitory connected with each other which leads to a winner-take-all (WTA) circuit. In a WTA circuit only the neuron with the highest activity spikes, other neurons are suppressed due to strong inhibitory connections within this layer.

The cause layer of the described model defines a Boltzmann Machine, therefore the prior is as described in Section 3.1.1 given by Equation 3.1.1 The input variables y_i are independent given the state of the latent variables:

$$p(\mathbf{y}|\mathbf{z}_i, \theta) = \prod_i p(y_i|\mathbf{z}_i, \theta). \tag{3.8}$$

As described before, only one neuron z_k should be active and explain the pattern which is shown to the network. This is achieved by strongly inhibitory weights W_{kl}, therefore the variables y_i given the state of z_k are independent. The full joint probability is given by

$$p(\mathbf{y}, \mathbf{z}|\theta) = \frac{h(\mathbf{y})}{Z} \exp[\frac{1}{2}\, \mathbf{z}^T \mathbf{W}\mathbf{z} + \mathbf{y}^T \mathbf{V}\mathbf{z} + \mathbf{b}^T \mathbf{z}]. \tag{3.9}$$

$h(y_i)$ is a helper function with terms which are not important for inference. For the maximization step we need an update rule for the weights V_{ij} between input layer and cause layer. We calculate the gradient of the average complete log-likelihood.

$$\frac{\partial}{\partial V_{ik}} \langle \ln p(\mathbf{y}, \mathbf{z}|\theta) \rangle_{p(z|y,\theta)}. \tag{3.10}$$

This leads to the following update rule:

$$\frac{d\, V_{ik}}{dt} = \eta \cdot z_k(t) \cdot (y_i(t) e^{-(V_{ik}+V_{i0})} - 1) \tag{3.11}$$

In a strict mathematically derivation of the update rule (see Breitwieser [2015]), we can see that the bias depends on these updated weights $b_k \propto V_{ik}$. Therefore all neurons have to adjust their effective bias and keep track of all corresponding synaptic weights V_{ik}. The patters have also to be normalized, to avoid advantages of some patterns. Another possibility is to maintain a target frequency of the cause layer neurons in a homeostatic manner. This can be achieved by applying homeostasis to the activity of the cause layer neurons in a closed–loop setup. Necessary mechanisms

which are needed for an implementation of a SEM experiment in a closed–loop setup on the BrainScaleS system are the following: Weight adaptation with weight calculations on the host computer, homeostatic adaptation of a neuron frequencies and a spike source to introduce stochasticity. We will discuss these points and how feasible a closed–loop SEM experiment on the BrainScaleS system is in Chapter 5. In Figure 3.1 we can see a background source for a spike-based homeostasis. By adjusting the weights between this background source and the cause layer neuron, we can maintain the frequency of these neurons. Additionally, poisson background sources are used to introduce stochasticity into the network.

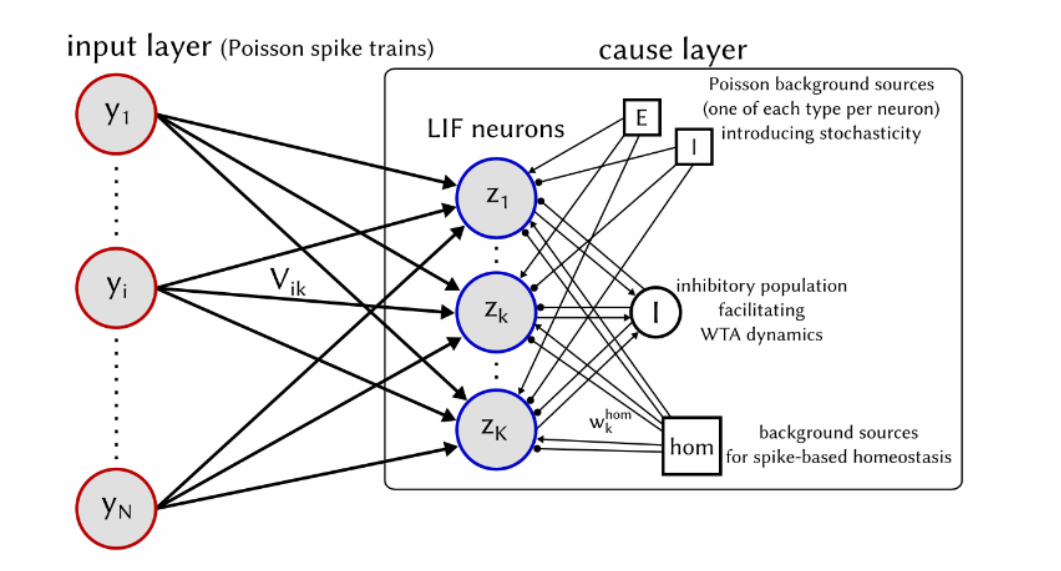


Figure 3.1.: Overview of the network structure, The network receives structured input in form of Poisson spike trains via the input neurons y_i. The visible layer is connected to the cause layer z_k with weights V_{ik}. The cause layer itself is strongly inhibitory connected with weights W_{kl} and turn the network into a winner-take-all network. [Scheme taken from Breitwieser [2015]]

3.2. Real–Time Computing

In his book *Real–Time Systems* (Kopetz [2011]) Hermann Kopetz describes a real–time computing system as follows.

> *"A real-time computer system is a computer system in which the correctness of the system behaviour depends not only on the logical results of the computations but also on the physical instant at which these results are produced."*

In such an environment the program has to respond within time constraints – also called *deadlines*. A deadline is the given time duration the system has to respond after an event was triggered. Dependent on the behaviour of the system if such a deadline is missed, we can distinguish between three categories. In a *Hard* real–time system producing the results after the deadline would lead to catastrophic consequences on the system. For example, a heart pacemaker has a hard deadline – missing a deadline could lead to loss of human life. The next category are *Firm* systems. In such a system missing a deadline would not lead to a total system failure but the produced results are useless for the system. The last category are systems with *Soft* deadlines. Missing a deadline will perhaps lead to performance degradation, but the information is still usable for the system.
In general we can divide the real–time system into subsystems. Figure 3.2 shows this decomposition. The operator or user interacts with the real–time computer system by sending and receiving data. The computer itself interacts with the object which should be controlled. Depending on what kind of object we want to control, this interaction can have different forms.

Important aspects of a real time system are timeliness, predictability, robustness and fault tolerance. As described before, not only the value of result is essential in such a system. The time domain in which the result was computed is significant as well. Predictability is the next important point – the system has to be analysable to predict what effect an action of the control system has on the controlled object. Furthermore, the control system has to be robust against peak–load conditions as well as fault tolerant towards single failures from the hardware or software.

On the BrainScaleS system real–time computing plays a minor role for ordinary experiments. In general, the host computer sends data like configuration information to the neuromorphic hardware where the experiment takes place. After the experiment is finished, the host computer receives the results which can then be analysed. Of course, this is a simplified view because we do not include the communications between the different parts of the neuromorphic system itself which

Figure 3.2.: In general we can divide a real–time system into smaller subsystems. This scheme shows the subsystems of such an real–time system. The operator interacts with a computer system to define rules how the object should be regulated. The computer itself interacts with the object which should be controlled. Depending on the information the computer receives it sends data back to control the object. Furthermore the computer sends information about the controlled object back to the operator.

can also be seen as a real–time system such as for FPGA programmes for data in- and output. However, real–time computing becomes a crucial part of the experiment when we operate in a closed–loop setup between a host computer and the BrainScaleS system. This topic will be discussed in the next section.

3.3. Closed–Loop operation mode of the BrainScaleS system

Most real–time computing systems act in a closed–loop setup to interact with an environment. There is no plain definition of a closed–loop system but in general there is an interaction between two or more components of the system with a forward data flow and a feedback. Typical examples for closed–loop systems can be found in the field of robotics. A robot receives sensory input of the surrounding environment. This data has to be processed and the robot has to react correspondingly.

3.3.1. Closed–Loop setup

If we combine the neuromorphic hardware and a host computer to form a hybrid system, we can use the BrainScaleS system in a closed–loop operation mode. In a conventional experiment the host computer is used to define the networks topology and the experiment, afterwards data like hardware configurations are sent to the system. After the emulation on the neuromorphic part, the host receives the results of the experiment in form of spike timing data. In the closed–loop operation mode the host computer can actively influence the experiment during emulation. The

hardware interacts with the environment and communicates directly with the host computer. Low latencies are important for these communications. A logical interface for real–time spike communication allows to send spikes from the host computer to the BrainScaleS system or readout the current spiking behaviour of a neuron. This interface is based on a collaboration between SpiNNaker and the developers of the BrainScaleS FPGA communication PCB firmware from TU Dresden. This leads to a closed–loop setup as described before. The host receives informations about the spiking behaviour and sends feedback to the BrainScaleS system which can be used to control for example the neurons frequency. Figure 3.3 shows a simplified overview of this hybrid system.

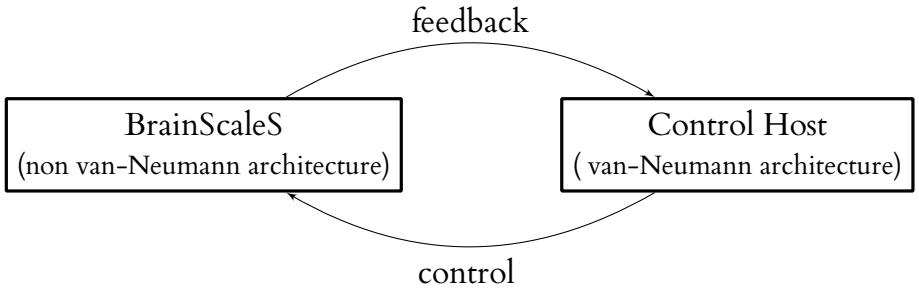


Figure 3.3.: Schematic view of the closed–loop mode of the BrainScaleS system. By combining the neuromorphic system and a conventional host computer , we form a hybrid system. The host will receive a feedback in form of spike information from the neuronal network. The host can send a feedback to the neuromorphic system and actively influence the emulation.

This interface uses the lowest level API to access the HICANNs (`hicannn-systems` and `halbe`) and is optimized for real–time communications. Therefore it avoids buffering and uses a small look–up table with 1024 entries to translate spike labels and send spikes between HICANN Chips and the host computer. Before we can use this interface, we have to define our network in PyNN. We run the mapping and receive a mapping and a hardware configuration file. These in PyNN defined settings can then be loaded within the closed–loop operation mode. The connection to a FPGA is encapsulated in a `fpga_handle` which can construct a realtime communication object `RealtimeComm` to start the real–time thread.

3.3.2. Spike Send / Readout Process

For sending spikes to a neuron on the hardware, we use the `RealtimeComm` object. We can send spikes to a neuron with `send_realtime_pulse(fpga_handle, SpinnInputAddress)`, where `SpinnInputAddress` is the address which is processed by the FPGA and translated to a hardware address with a routing table. The routing table can be created, after the marocco mapping file is loaded. The table can store 1024 entries of pulse addresses, each address consists of 16 bit which encode the corresponding neuron address. When we create a `RealtimeComm` object, we initiate transmission (`TX`) and receive (`RX`) buffers. We can insert data into the transmission buffer which will be extracted by a network API, furthermore data can be inserted into the receive buffer and be extracted by the user process. Additionally, `sockets` are created which define endpoints for the communication. There are several different socket types available, here `SOCK_DGRAM` was chosen. This socket uses the *User Data Protocol* (UDP) to send packets between host and neuromorphic part. Compared to the widely used *Transmission Control Protocol* (TCP) which is build on top of UDP, the used data protocol has some drawbacks. UDP does not promise that the packets will be received in the right order. Besides, if a packet is lost, the sender will not be told so because there is no acknowledgement of packets from the receiving side. The advantage of this protocol is that it avoids the latency which is caused by TCP error correction. In our case a lost packet is preferred to a delayed packet. In terms of real–time computing, this would correspond to a firm real–time system. A delayed packet would not drive the system to a total breakdown, but the spike information would be useless for the system.

For rate based operations like SEM we are not interested at the precise spike timing of a single spike. Rather we are interested in the current spike rate of a neuron.

$$\nu = \frac{n_{sp}}{T} \tag{3.12}$$

Where n_{sp} is the amount of spikes in a time period T. Several spikes should occur within this period to get reliable results. Typical choices for T are $100ms - 500ms$ (Gerstner et al. [2014]), but this choice depends on the one hand on the kind of neuron type and its input and on the other hand on the necessary resolution of rate updates for the performed task. Based on the information of the current rate, the host computer can react in form of sending spikes or adjusting weights (see Section 3.3.3). Therefore we use two different threads during the experiment. The first thread waits an adjustable `accumulation_time` which corresponds to T and reads out the `RX` buffer to get the amount of spikes since the last readout. Divided by the elapsed time we receive the current rate of the neuron. The second thread sends continuously spike packets from the host to the hardware neuron. With an adjustable wait time between each sent spike, we can adapt the amount of spikes

we are sending to the neuron. Based on measurements which were carried out before this thesis [see Schneider [2018]], we know that due to software latencies the minimal wait time between two spikes is limited to 2 μs which corresponds to 500 kHz firing rate in hardware time or to 50 Hz in biological time. These limitations are possible constraints for a SEM experiment which will be discussed in Section 5.

3.3.3. Weight Adaptation Process

The strength of the connection between neurons is described by its synaptic weight w^{syn}. On the hardware this weight is proportional to an analog parameter g_{max} and a 4-bit digital weight as described in 2.2.1. From the host this digital weight can be adjusted for every synapse in the closed–loop mode and will be further analysed during this thesis. Therefore we extract which rows of the synapse array are used by the projection we want to adjust. These information are contained in the `marocco` results. Similar to the spike send process, the weight adaptation is based on information about the current frequency of a neuron from the spike receiving thread. In a second thread the new weight is calculated and the `set_weights_row(HICANN, row, weight_row)` function is called which performs the weight update. The calculation of the new weight depends on the corresponding task. In a closed–loop SEM experiment this could be a calculation of a new weight update between visible and hidden layer. In Section 4.3 we will show how this process can be used for rate adaptation of a neuron. For the spike send and readout process we could use the SpiNNaker interface which is optimized for real–time operations. To change a weight on hardware we have to use the HostARQ protocol. This protocol aims at a reliable communication between host computer and HICANN chip but it not optimized for real–time closed–loop operations. Therefore a characterisation of this process is necessary to estimate if weight updates in the closed–loop operation mode are feasible.

3.4. Homeostasis

3.4.1. Biological Concept

Homeostasis is a biological concept defining a state of equilibrium. Originally describing the self regulating internal environment in which cells live. Figure 3.4 shows the typical homoestatic process. A variable which should be maintained is observed by a receptor. The receptor sends the information about the current state of the variable to a controller which decides based on these informations how to react. If the current state of the variable differs from the target value, the controller activates an effector to actively regulate the variable towards the target value. Typi-

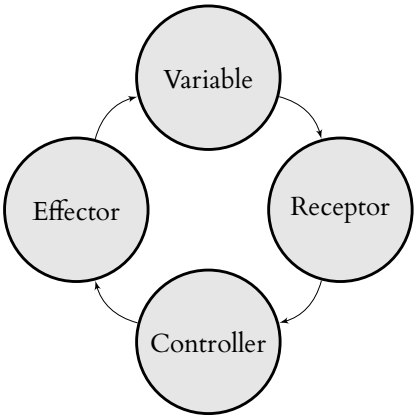


Figure 3.4.: Schematic view of the underlying process of homeostasis. A receptor acts as a sensor and detects if the value of the variable which should be maintained differs from the target value. This information will be transferred to a controller which activates an effector that is able to control the variable.

cal homeostatic processes in the human body are the maintainance of the body heat or the regulation of blood pressure. In case of the human body heat the body temperature is the variable which should be maintained. The central nervesystem acts as an receptor and is able to detect temperatures in different regions of the body. The brain as a controller compares the detected temperatures with its target temperatures and we start to sweat if the current temperature is to high or to chatter if it is to low.

3.4.2. Homeostatic Rate Adaptation

The concept of homeostasis can be applied to the rate of spiking neurons to maintain a defined target frequency. On the BrainScaleS system we will use the closed–loop operation mode to solve this task. In Figure 3.5 the setup for this implementation is shown. The neuron we want to control is shown at the top and receives disturbing input from other spike sources. Because of these disturbances the firing rate of the neuron can deviate from the target rate. This deviation can be compensated with an excitatory and an inhibitory spike source which are shown below the neuron. The host computer will act as an controller and compares the current rate of the neuron with its target rate. Based on this error which is defined by $error = rate_{target} - rate_{current}$, we need to adjust the spike input to the neuron from the excitatory / inhibitory spike source to drive the neurons rate towards the target rate. As described in section 3.3.2, it is possible to send spikes from the host to neurons on the hardware in the closed–loop mode. In Schneider [2018] we showed how sending spikes from

the host could be used to regulate a neurons rate in a homeostatic manner. We will show a different approach in section 4.3.2 using an on–wafer spike source.

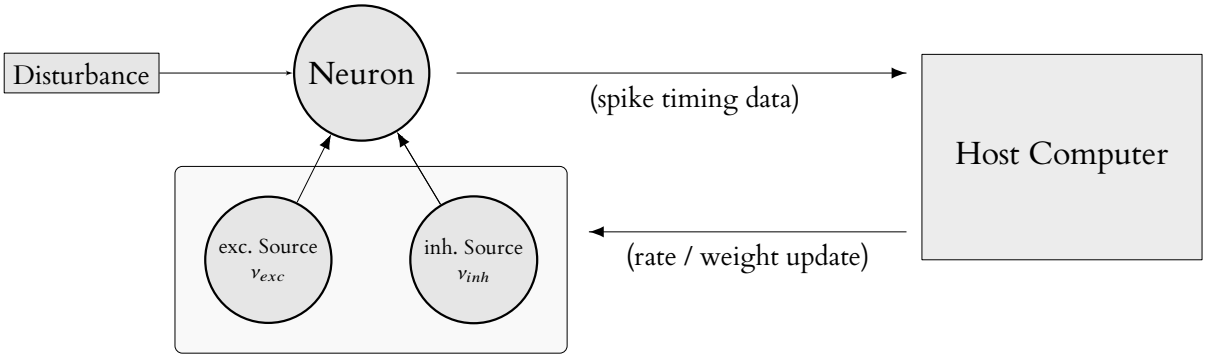


Figure 3.5.: Schematic topology of the homeostatic setup. The neuron we want to apply homeostasis on receives disturbing input in form of spikes from other neurons. The host compares the current rate with the target rate and drives the neurons firing rate towards the target rate by adapting the frequency of the connected inhibitory and excitatory spike sources correspondingly.

3.5. Sea–of–Noise Networks

For SEM experiments we need to introduce stochasticity into our network. One approach could be to use poisson spike sources and connect them with the corresponding neurons. On the BrainScaleS system the FPGA background generators are able to produce poisson distributed events on 1 to 8 channels simultaneously. Also regular event input on a single channel is possible. However, only one HICANN at a time can receive such distributed events from the background generator.

Another solution are Sea–of–Noise networks (SoN). These networks use the fact that a Neuron with a resting potential E_{rest} above the threshold V_{thresh} leads to a continuous spiking neuron. The neuron will spike in a regular fashion with a frequency that depends mainly on the chosen refractory period. This regular spiking will not introduce stochasticity, but if we connect several of these neurons inhibitory with each other, we can generate pseudo random spike trains. A neuron which should receive stochastic input can then be connected with several randomly chosen neurons of this SoN population. Additionally, such a network can be used as an on–wafer spike source as we need it for the homeostatic rate adaptation. We will present and compare in section 4.3.2 different approaches using these networks. The topology of such a network is shown in Figure 3.6. The nodes on the left side represent neurons in the SoN network. The resting potential of all these neurons is set above

the threshold potential. This would lead to a continuously spiking neurons with the maximum possible frequency, based on the chosen neuron parameters. Because of the random inhibitory pre–connected partners of each neuron within this network, this frequency will be limited. By adjusting the size of this network N, the inhibitory weight between those neurons w^{inh} and the amount of randomly chosen pre–synaptic partners K per neuron, the behaviour of this network can be controlled. A neuron which should receive random spike input can then be connected with some of these neurons from the SoN network. By adjusting the amount of connections and the weight of those connections from the SoN network to the neuron, we can vary the frequency of the input the neuron receives. We will use SoN networks as on–wafer spike source for our experiments, but a deeper analysis of these networks will not be part of this thesis but can be found in Pfeil et al. [2016].

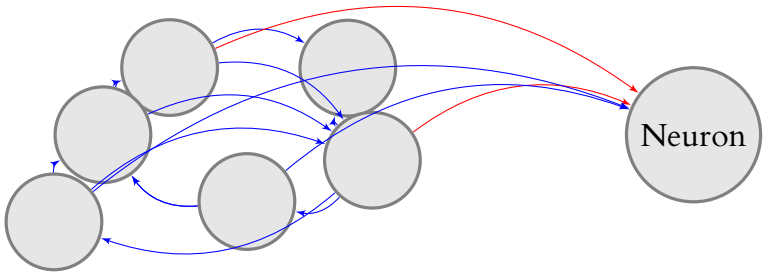


Figure 3.6.: For neurons in a *SoN*-network the threshold v_{thresh} is set below the resting potential v_{rest}. This leads to a bursting behaviour of the neurons. These neurons have a random inhibitory (blue) connection whith each other. Some of these neurons from the SoN population are excitatory (red) or inhibitory connected to the external neuron.

4. Measurements

We will present and test in this chapter different methods to regulate the firing rate of a neuron in a closed–loop setup on the BrainScaleS system. This can be achieved by connecting a neuron with a Sea-of-Noise (SoN) network and adjust the digital weights of those connections correspondingly.

As described in Section 3.3, the computations on the host computer have to keep up with the massive speed-up of the BrainScaleS system in the closed-loop operation mode. Therefore, special considerations have to be given to the software implementation. The following optimizations have been used to satisfy the necessary timing constraints. With the scheduling tool `schedtool` we can set CPU scheduling parameters of the Linux kernel. We use the real–time scheduling policy `SCHED_FIFO` for the processes during an experiment. A process using this policy pre-empt other tasks with minor priority. Additionally, the process was marked as non pageable and the necessary memory was pre–allocated. To avoid latencies from CPU switching, the process was pinned to one CPU.

It is important to note that the time domain of all measurements – if not stated otherwise – represents hardware time. To obtain biological time, the speed–up factor of 10^4 must be taken into account.

4.1. Weight Updates in the Closed–Loop Operation Mode

The implementation of the weight adaptation process was described previously in Section 3.3.3. Before we start with a measurement to characterise the weight setting process, we start with a demonstration of a weight update in the closed–loop operation mode. As described in Section 2.2.1, the digital weights represent the synaptic strength of the connection between neurons and have a 4 bit resolution which leads to 16 different possible weight settings. Figure 4.1 shows the mapping topology for this measurement.

Neuron n_2 is excitatory connected to a continuously spiking neuron n_1 ($V_{rest} > V_{thresh}$). Neuron n_2 will receive spikes from neuron n_1 and will start to fire spikes if the weight between both neurons is strong enough. Figure 4.2 shows the rate over time of neuron n_2. After 500 ms the host sends a signal to increase the weight be-

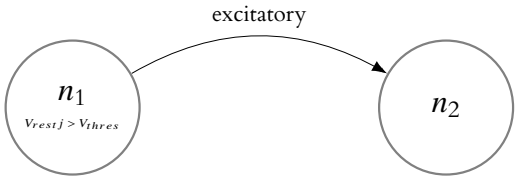


Figure 4.1.: The shown topology is used to determine the duration of the weight adaptation process. A continuously spiking neuron n_1 ($V_{rest} > V_{thresh}$) is connected to neuron n_2. By increasing the digital weight between both neurons, n_2 will receive spike input from n_1.

tween both neurons by one which is represented by grey vertical lines. Every grey vertical line represents the point in time when the host sends a signal to increase the weight by one. The first update which is shown in this plot represents an increase of the digital weight from 2 to 3. The connection between both neurons is in this case not strong enough to trigger neuron n_2 to start firing. After the digital weight was set to 4 after 500 μs, n_2 starts spiking with a frequency of 400 kHz. This update has the strongest impact on the neurons frequency. The following updates show clearly the non linear dependency between the weights and frequency. The behaviour of the neuron is as we would expect it, the noticeable thing about this measurement is that the weight update was triggered from the host. During emulation of those neurons on the BrainScaleS System, the host sent a signal to the corresponding FPGA which realizes the update of the digital weight.

4.2. Characterisation of the Weight Adaptation

As mentioned in Section 3.3.3, we use a different access channel for the weight setting process then for the spike sending process in the closed-loop operation mode. To send spikes between host and the BrainScaleS system, we used the SpiNNaker interface which is optimized for real–time communications (see Section 3.3). To set weights, we have to use the HostARQ protocol. Because this protocol is not optimized for real–time communication, we are interested in what time period this weight setting process takes place – the elapsed time between triggering a weight update on the host computer and the point in time when the weight update is realised on hardware. With this information we can estimate if a rate adaptation with weight updates is feasible. Therefore we use again the setup from 4.1, a continuously spiking neuron n_1 and a connected neuron n_2. To determine the described time period we use the moment when we call the function to update the weight as time stamp t_1. We set the digital weight to the maximum value of 15, therefore neuron n_2 will immediately start to spike after the weight is set. The point in time

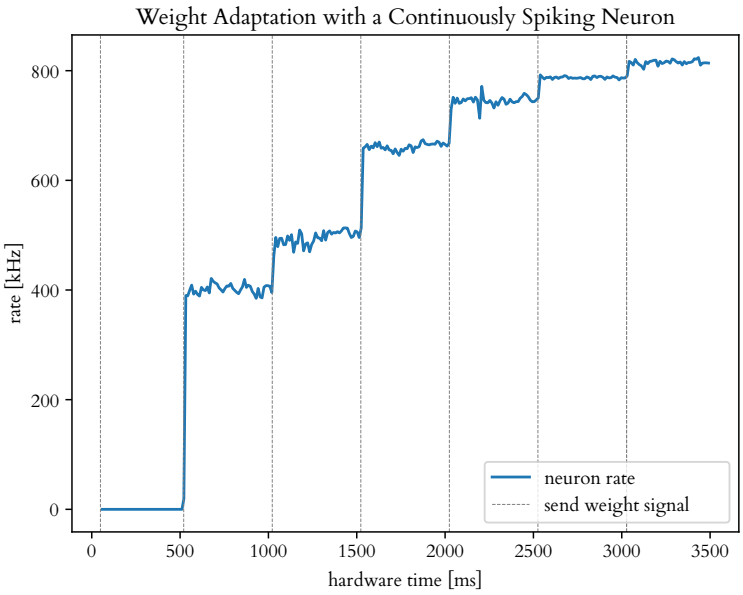
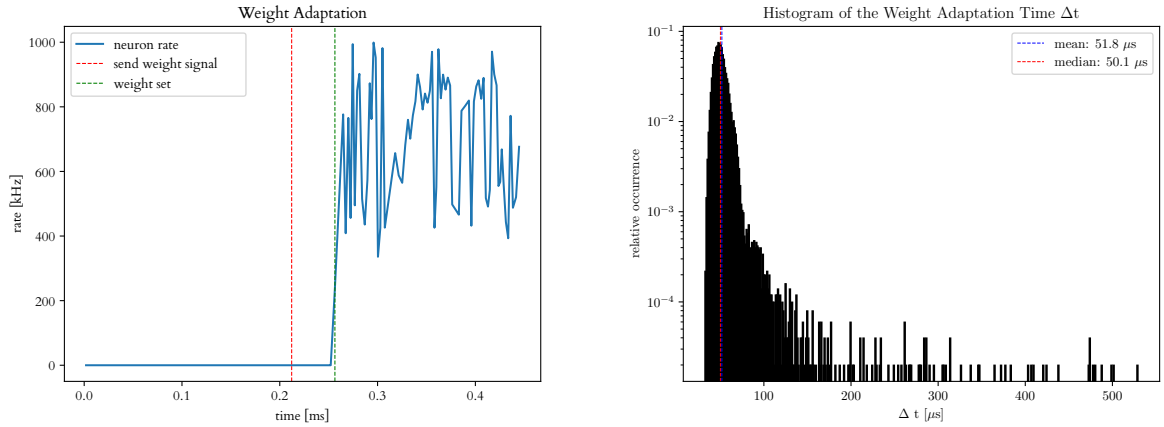


Figure 4.2.: This figure shows the effect of the weight adaptation. The setting is the same as described in Figure 4.1. By increasing the weights between n_1 and n_2, the frequency of neuron n_2 –which is shown in the plot – starts to increase.

when neuron n_2 starts spiking, is time stamp t_2. More precisely defines this interval $\Delta t = t_2 - t_1$ the round–trip time because it includes the communication from the host computer to the FPGA and the corresponding HICANN as well as the communication back to the host computer. However, in later experiments the weight updates will be calculated on the host computer based on the current behaviour of neurons on the hardware system. For this process the communication latencies between both systems have to be considered as well. The measured interval is shown in Figure 4.3a, the red vertical line represents the point in time when the signal for a weight update was sent from the host and the green line when the weight was set which is defined as the point in time when the neuron starts to spike. To determine the frequency of a neuron we accumulate spikes for a accumulation time $\Delta t_{accum.}$ which we set before an experiment. Divided by the amount of spikes of the neuron within this time interval, we receive the current rate. To determine if the neuron started spiking, the accumulation time was set to $\Delta t_{accum.} = 1\mu s$. Such short accumulation times lead to a strong variance in the determined frequencies which can be seen in the rate trace in Figure 4.3a. As we are only interested in the point in time when the neuron starts spiking, the accuracy of the determined frequency is unimportant for this measurement. However, a lower accumulation



(a) (b)

Figure 4.3.: We want to determine the time duration between sending a signal to set a weight from the host to the BrainScaleS system and when the weight change is realized on hardware. **(a)** shows the time interval we want to determine. A neuron n_2 is connected to a continuously spiking neuron n_1. By changing the weight between those neurons from 0 to 15 with a function call on the host (times stamp t_1), n_2 starts also to spike (time stamp t_2). **(b)** shows a histogram which visualizes the time duration Δt between the green and red dashed line in plot (a).

time increases the resolution of this measurement. The moment in time when the neuron starts to spike – when the rate ν of the neuron is the first time larger then zero after the signal for a weight change was sent – is set as *weight set* time point t_2. We measure this time interval $\Delta t = t_2 - t_1$ several times and visualize the data in a histogram in Figure 4.3b. The histogram shows a mean weight adaptation time of $\Delta t_{weight} = 52 \pm 14 \mu s$. Which corresponds to $\Delta t_{weight,bio} = 520 \pm 140 ms$ in biological time. This leads to a relative error of 27 %. We observe a hard boundary on the left side of the main peak. This is to be expected because the signal needs even for the best case a fixed amount of time which can not be undershot. The right side of the main peak shows a wide spread distribution which can be explained by software and communication latencies. The host computer has to communicate with the FPGA and the FPGA with the corresponding HICANN chip. Latencies are also caused by the error handling of the HostARQ protocol. As described previously, this protocol is not optimized for real–time operations. In rare cases this process can take even an order of magnitude longer then the average weight adaptation process. Especially for peak–load conditions on the host computer, the process be queued which leads to higher adaptation times as can be observed in this measurement.

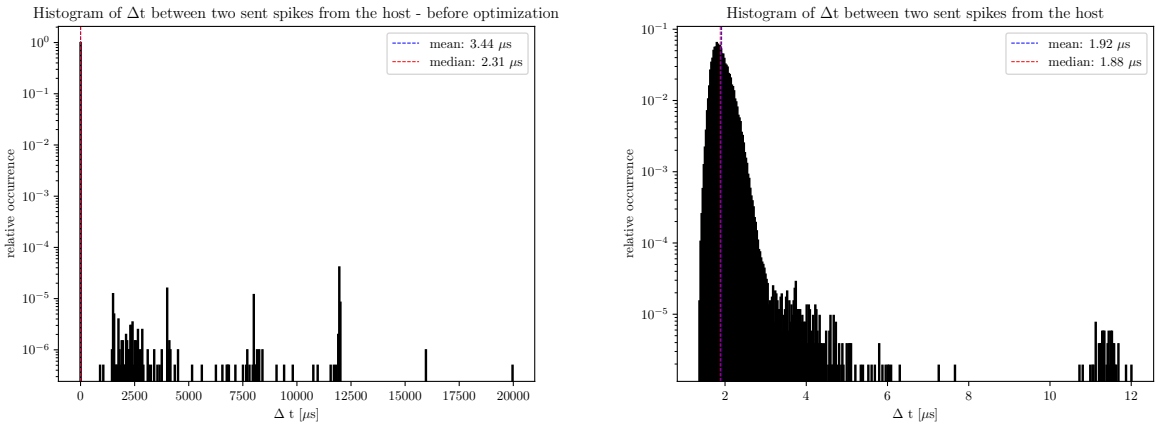
4.3. Rate Adaptation with Weights

We showed that rate adaptation of a neuron in a closed–loop setup is possible by sending spikes from the host to the neuromorphic part (see Schneider [2018]). Sending continuously spikes from the host to the hardware neuron should lead to a constant firing rate of the neuron. By adjusting a wait time between each sent spike, we can regulate the amount of spike which are sent from the host. However, software latencies set a upper limit for frequencies which can be reached with this procedure. Sending spikes as fast as possible to the BrainScaleS system – setting the wait time between to sent spikes to zero – and measure the time between two *send spike* calls, leads to a distribution as shown in Figure 4.4b. This measurement shows only software latencies, communications between host and BrainScaleS System are not included. The histogram in Figure 4.4b shows some smaller peaks several orders of magnitude away from the main peak around 2.31 μs. To use spikes which are sent from the host to maintain or adjust a spiking frequency we have to get rid of these outliers. After the described real–time optimizations at beginning of this chapter, we receive a distribution which is shown in Figure 4.4a. A slightly skewed Gaussian distribution with a hard boundary on the left side which can not be undershot. The right side shows a noisy behaviour which is a typical characteristic for measurements of this kind. Even with the used scheduling policies it is not always guaranteed that the corresponding process will be executed immediately. Other processes with a high priority can get the preference and will be executed earlier and the *send spike process* will be queued. An explicit reason for the small peak around 11 μs could not be found until now. The mean interval between two with maximum speed sent spikes is around $\Delta t = 2\mu s$. This corresponds to a frequency of 500 kHz. However, in practice only frequencies around 400 kHz showed a stable behaviour. For a more detailed explanation see Schneider [2018].

4.3.1. Using a continuously spiking Neuron

Another approach for a rate adaptation uses continuously spiking neurons as an on–wafer spike source during emulation. We use again the setup shown in Figure 4.1 for this measurement. We can adjust the weights between both neurons correspondingly to push the neurons rate towards a defined target rate. A possible solution to find the weight which drives the neuron towards a target rate is a simple linear adaptation. By increasing the digital weight at every update by one based on the current frequency of the neuron. In this measurement every 10 ms a weight updated takes place.

We define an error as the difference between target rate ν_{target} and the current rate of the neuron $\nu_{current}$. If the error is positive, we increase the digital weight, if it is negative we decrease it. Furthermore we define an error tolerance. If the absolute



(a) (b)

Figure 4.4.: We send spikes as fast as possible from the host to the BrainScaleS system and measure the time period between sending two spikes.**(a)** shows the distribution before the optimization. **(b)** This histogram shows the distribution of these time durations after the real–time optimizations. The plots and a detailed explanation can be found in Schneider [2018].

value of the error is below this tolerance, now weight update will be executed. Figure 4.5 shows this rate adaptation applied on a neuron. The red horizontal line represents the target rate which should be maintained. At the beginning the target rate is set to 700 kHz and switches after 200 ms to 500 kHz. The vertical dashed lines indicate the weight updates. We can see that the rate of the neuron oscillates around the target value. It would be possible to avoid those oscillations by increasing the accepted error tolerance. However, if we look at the impact of a single weight update in Figure 4.5 on the frequency of the neuron, we observe that the effect of a weight update is to rough – a rate adaptation is not feasible this way. We could decrease the frequency of the spiking neuron n_1 by increasing its refractory period τ_{ref} which would lead to finer adjustments per weight update. A lower highest frequency which can be reached would be the consequence.

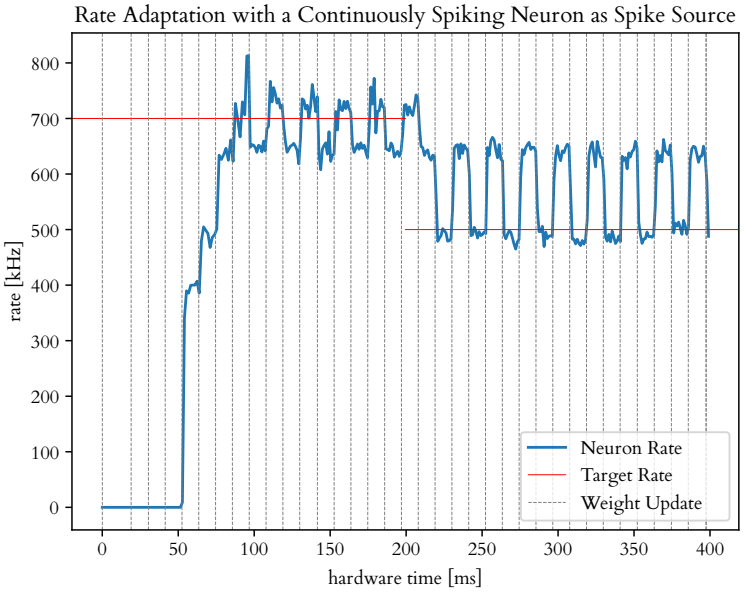


Figure 4.5.: The setup is the same as in Figure 4.1, the weights are updated based on the current frequency of the neuron to reach a target frequency which is displayed as a red horizontal line and switches after 200 ms from 700 kHz to 500 kHz. If the current rate is below the target frequency the weight will be increased and decreased otherwise.

4.3.2. Using a Sea–of–Noise network

A more plausible approach is to use a Sea–of–Noise (SoN) network as spike source. As described in Section 3.5, we can adjust the SoN network parameters like the amount of neurons N within this network, the number of pre connected partners K or the inhibitory weight w^{inh}. This leads to a lower spike frequency of the SoN neurons compared to continuously spiking neurons like in the last section and. This makes it possible to use a SoN network as an on–wafer spike source. Furthermore, we could decrease the amount of data which we would send from the host to the BrainScaleS system compared to the homeostasis implementation shown in Schneider [2018]. In the implementation shown in Schneider [2018] we send spikes from the host to the hardware to adjust a neurons frequency in a homeostatic manner. With this approach spikes would be generated on the hardware and only the weight updates would be sent from the host. In the following measurements we use a Sea–of–Noise network with 100 neurons, every neuron has 10 inhibitory connected partners with a digital weight of w^{inh}=1. This leads to a mean firing rate of the neurons within the SoN network of 120 ± 30 kHz. We measured the mean rate of each

neuron in this SoN network. The standard deviation of these determined mean rates defines the error of 30 kHz. It is important to note that this large error does not mean that a single neuron spikes with such a large variance. It just shows that the mean rates of the neurons within this population are widely distributed, which is caused by the random inhibitory connections. The neuron we want to regulate has 40 connections to randomly chosen neurons of this Sea–of–Noise network. The amount of 40 connections was chosen empirically and should not be seen as a general rule. The amount of necessary connections depends on the used SoN network as well on the necessary accuracy and the possible maximum rate which should be reached. To reach higher frequencies the SoN network has to be tuned, the amount of inhibitory pre-synaptic partners or the inhibitory weight can be decreased. Also the amount of connections between the neuron and the SoN network can be increased.

Linear Adaptation

By adjusting the digital weights of the connections between the SoN network and a neuron, we can regulate its firing frequency. In this first implementation we use a linear weight adaptation for the rate adjustment. At every weight update the weight of one of those 40 connections is increased by one until the target frequency is reached. Because of the 4 bit weight resolution per synapse, we get for 40 connections in total 600 different weight steps. We can imagine these weights as an array with the length defined by the amount of the connections. The i-th connection will then be set to the value of the i-th entry in this array. In this case every $50\mu s$ in hardware time a weight update takes place. The reason for this chosen update frequency will be explained in the following. Furthermore only weights which were changed compared to the previous weight setting will be updated. In case of a linear adaptation with a step size of 1 weight step, only one connection weight will be changed per update. Is the current frequency within a tolerance frame of the target rate, no weight update will be executed. Figure 4.6 shows the rate adaptation of a neuron with a target rate of 200 kHz. Every vertical dashed line represents a weight update, when the target rate is reached the density of weight updates decreases. The tolerance frame was set to ± 10 kHz around the target rate. We can see that the target rate is reached after around 170 ms and is maintained. Between 200 ms and 500 ms the mean rate of the neuron is 200 ± 11 kHz which corresponds to a frequency of 20.0 ± 1.1 Hz in the biological time domain.

Analysis of Different Update Frequencies

Figure 4.3 shows the rate traces of a neuron with a target rate of 400 kHz and different update frequencies. We can not increase the rate adaptation by increasing the weight update frequency further because an update every 50 μs defines a lower

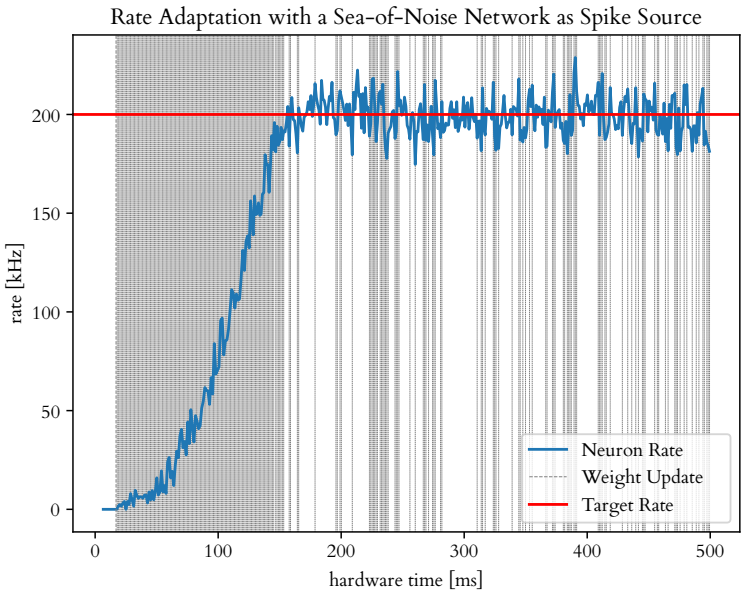


Figure 4.6.: This figure shows a rate adaptation with weights and a Sea-of-Noise network as spike source. The SoN network consists of 100 continuously spiking neurons, each with 10 inhibitory connected partners with a digital weight of $w^{inh} = 1$. Every weight update is represented with a grey vertical line. At every update step the weight of one of those 40 connections is increased or decreased by one until the target rate is reached.

bound. If we increase the update frequency further, the rate adaptation will not be accelerated as can be seen for the 10 μs trace in Figure 4.7 which is covered by the 50 μs trace. In Figure 4.3b we determined the duration for a weight update which was around 52 μs. This boundary can not be crossed even if we update different connections. The reason is that these connections are on the same chip and are updated by the same update controller. Therefore we are limited by the weight-set duration which we determined in Figure 4.3b It is hard to determine specific reasons for this latency and to decide if this process can be improved. There are different sources which contribute to this latency like the communication between Host, FPGA and HICANN as well as software latencies and were not analysed further during this thesis. Figure 4.8 shows an overview of rate traces for different target traces. In this case the weight updates take place again every 50 μs. This plot visualizes the linear weight updates with a step size of one weight per update. Furthermore, we can see again the non linear dependency between weights and the rate of the neuron.

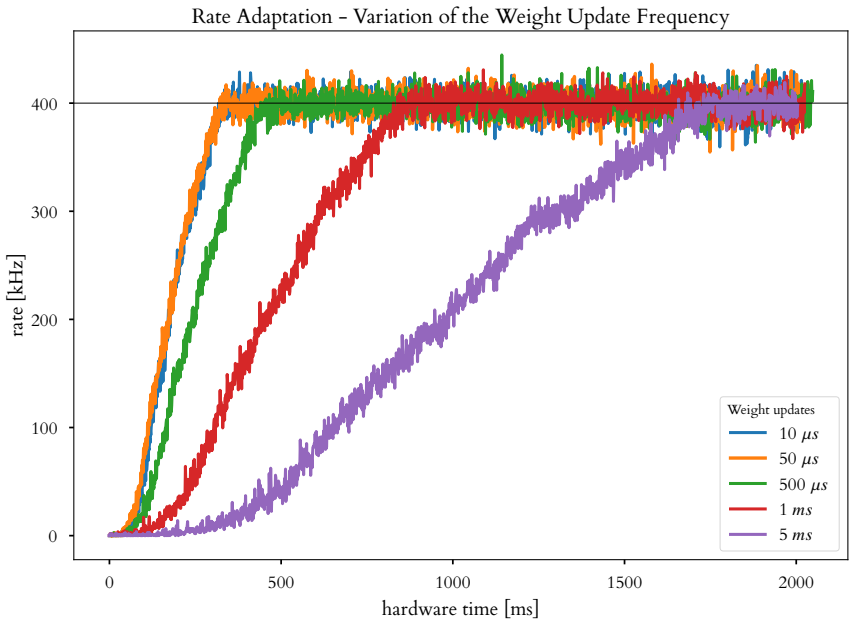


Figure 4.7.: This plot shows different traces of the rate of a neuron with a target rate of 400 kHz. By adjusting the rate of the weight updates we can control the speed of the rate adjustment.

Binary Search

There are different other update procedures possible to accelerate the rate adaptation. The amount of connections between SoN Network and the neuron is adjustable before an experiment, during an experiment this number is fixed. Because of that, the rate adaptation can be seen as a search for the right weight setting. For a fixed amount of different possible weight sets $\mathcal{W} = \{w_i | w_i \in \{0, .., 15\}\}$ we can use a binary search. The idea is to split the whole possible range in half and set the current weight in the middle. Based on the current frequency after this weight update we repeat this process with the upper or lower half of the possible set of weights until we reach the target frequency. With this approach the necessary amount of updates to find the right weight setting scales with $O(\log\ n)$ compared to $O(n)$ for a linear search. In Figure 4.9a we can see how this approach works. Independent from the target frequency all traces start after the first update in the middle of the possible weight range which correspond approximately to the middle of the possible rate range. Afterwards they split up and reach their target rate after around 200 μs. To avoid oscillations around the target rate, we switch to a linear weight adaptation if the current rate is within a tolerance frame. We can see that this methods slows

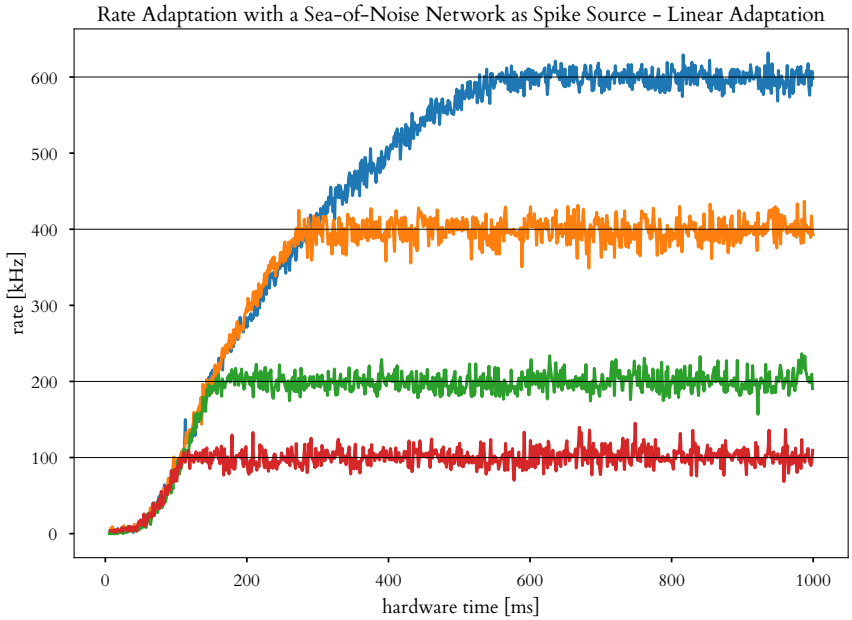
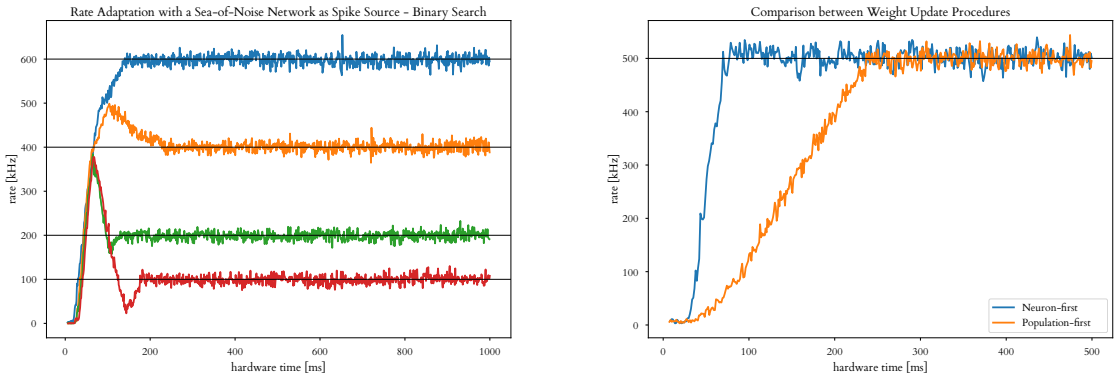


Figure 4.8.: This plot shows traces of a neurons frequency with different target rates. We used a linear adaptation with an adjustment of one digital weight every 50 μs.

down the process of adjusting the neurons rate for low target rates compared to a linear adaptation. On the other hand this method accelerates the adaptation for cases with large differences between current rate and target rate. For a linear adaptation we need to reach a target rate of 600 kHz around 550 μs, with a binary search this target rate was reached under 200 μs. The disadvantage of this method is the discontinuous rate trace. In a SEM experiment this could destroy the dynamics of the winner–takes–all network.



(a) (b)

Figure 4.9.: The rate adaptation can be accelerated with two different approaches which are shown in this Figure. **(a)** shows the implementation of a rate adaptation where we apply a binary search on the possible set of weights. The plot shows four traces with four different target rates which are represented by black horizontal lines. **(b)** shows a comparison of the different update procedures which are explained in Figure 4.1. We use an increasing step size for the weight updates and can see the impact of the two update procedures on the adaptation speed.

Adaptive Step Size

Another approach to accelerate the rate adaptation is to use different step sizes per weight update. For the linear adaptation approach, we updated the weights with a step size of one digital weight. Now we will increase the step size additive by one at every update if the sign of the current error is the same as in the previous update. The error is again defined as the difference between target rate and the current rate of the neuron. This leads to larger step sizes if the deviation from the target rate is tall and keeps the step size small for minor deviations. The blue graph in Figure 4.9b shows this approach with a target frequency of 500 kHz. This rate is reached after 250 ms which is around 100 ms faster then the simple linear adaptation with a step size of one. The limiting factor is again the duration for the execution and realization of a weight update on hardware. To accelerate this adaptation further, we need to decrease the amount of weight updates. In Figure 4.1 we can see two different weight update procedures. The i-th entry of the array represents the digital weight of the i-th connection. If we increase the weights at every update step by one digital weight, we adjust at every update step a different connection until every weight is set to one and start again with the first array entry. This is how

we updated the weights until now. We will call this the *Population-first* procedure because we connect first the whole population of neurons from the SoN network which are connected to the neuron we want to control before we increase the digital weights further. A problem arises if we use larger step sizes than one. If we need to increase the connection strength between the neuron and SoN network by an amount of ten digital weights in one update step, we had to set the digital weights of ten different connections. This can be avoided by using the *Neuron-first* procedure shown in Figure 4.1. First, we increase the weight of the connection of a single neuron from the SoN up to 15 before we start with the next connection. The advantage of this method is that we need less updates of hardware connections, we can increase the digital weight of a single connection by up to 15 weight steps and have to change only one weight on hardware. In the example in Figure 4.1 we start in both cases with all connections set to zero. We want to increase the connection strength between SoN network and the controlled neuron by an amount of 16 digital weights. The *Population-first*-procedure updates at every step a different connection and starts again with the first entry if the end of the array is reached. The *Neuron-first*-procedure increases the weight of a single connection to 15 and continues afterwards with the next connection. In this example we had to update 16 different weights on hardware with the Population-first approach, and only two connections with the Neuron-first procedure. It is important to note that it has not the same effect on the neurons frequency if we increase three different connections by one or one single connections bye three because of the non linear dependency between weights and rate. Nevertheless, we want to examine if this approach could be used to accelerate the rate adaptation. Figure 4.9b shows also a graph for a rate adaptation using the Neuron-first procedure. We can see that the target rate is with the Neuron-first procedure reached after 80 ms, which is around three times faster as with the Population-first procedure. This shows that using the Neuron-first procedure can accelerate the adaptation process if needed.

Maximum Step Size

When we use an increasing step size, it is possible to control the speed of the adaptation process by defining a maximum step size. In Figure 4.10 we can see this approach for different maximum step sizes. The target rate is set to 500 kHz and switches after 500 ms to 200 kHz. For a maximum step size of one we get similar results as for the linear adaptation in Figure 4.8 – a target rate of 500 kHz is reached after approximately 400 ms. By increasing the maximum step size, the adaptation is accelerated – for a maximum step size of 20 the target rate of 500 kHz is reached after around 80 ms.

	Population-first	Neuron-first
Start	[0, 0, 0, 0, ..., 0]	[0, 0, 0, 0, ..., 0]
1.Weight	[1, 0, 0, 0, ..., 0]	[1, 0, 0, 0, ..., 0]
2.Weight	[1, 1, 0, 0, ..., 0]	[2, 0, 0, 0, ..., 0]
3.Weight	[1, 1, 1, 0, ..., 0]	[3, 0, 0, 0, ..., 0]
⋮		
16.Weight	[1, 1,...,1, 0,...,0]	[15, 1, 0, 0, ...,0]

Table 4.1.: We present two different update procedures. The different weights can be visualized as an array where the i-th entry represents the digital weight of the i-th connection between Sea–of–Noise network and the neuron we want to control.

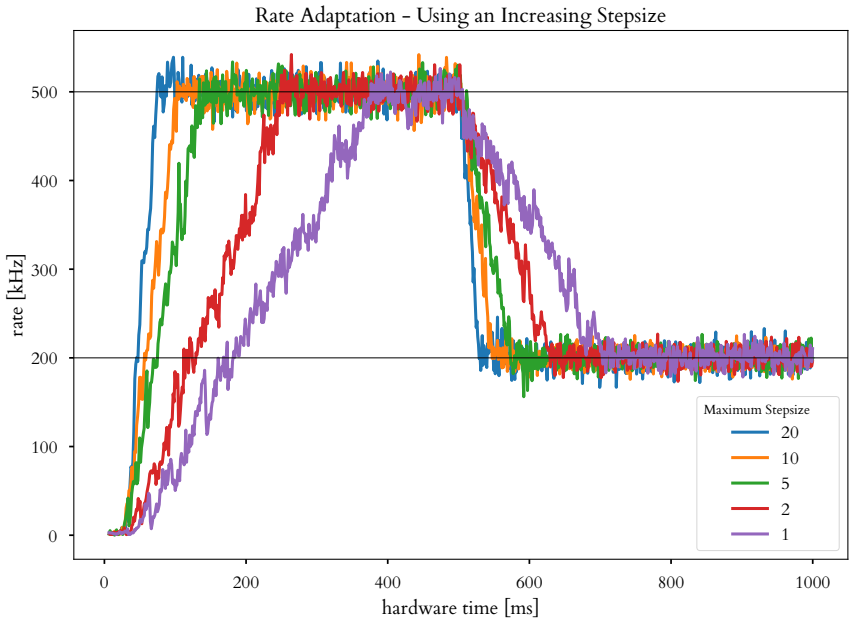


Figure 4.10.: Using an increasing step size with the Neuron-first procedure from Table 4.1. This acceleration can be controlled by defining a maximum step size which will not be passed. This plot shows five traces for different maximum step sizes. The target rate is set to 500 kHz and changes after 500 ms to 200 kHz.

Different Amount of Connections

We used for all measurements the same SoN network and also the same amount of 40 connections for comparison purposes. However, the amount of connections can easily be changed. Figure 4.11 shows rate traces of a neuron with different amounts of connections. The target rate was set to 800 kHz for all three traces. The blue graph with 20 connections is not able to reach the target rate. Even if all weights are set to the maximum value of 15, the rate is still below 400 kHz. With 40 or 60 connections this target rate can be reached. If we look at the beginning of the emulation and compare the two traces with 40 and 60 connections, we can see that even if all weights are set to zero, the neuron receives spike input if we use 60 connections. This is a problem as we would not be able to regulate the neuron on frequencies below 100 kHz. In general the neuron should not receive spikes if the weights are set to zero, with an adjustment of the neuron parameters this problem can be solved. It appears to be a calibration problem and has to be analysed further.

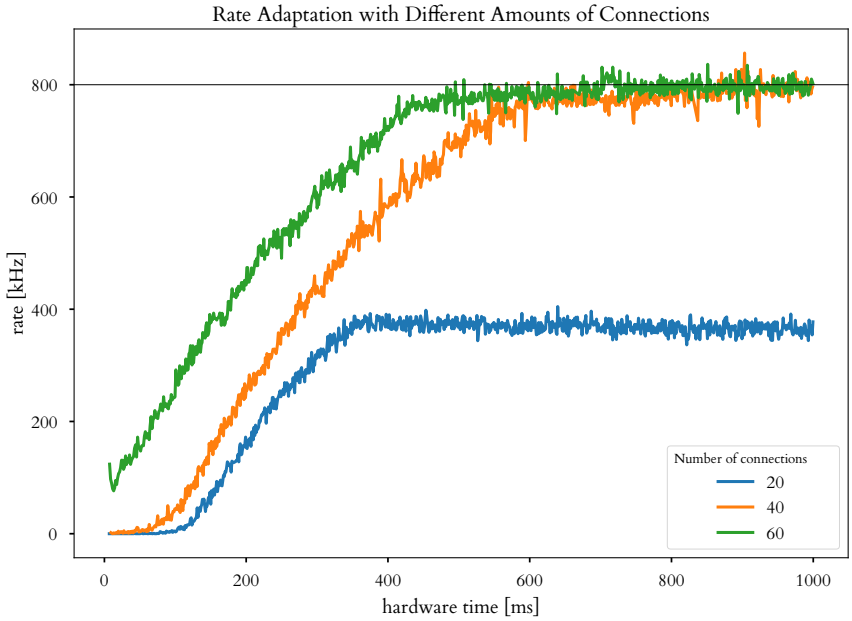


Figure 4.11.: The amount of connections between SoN network and the controlled neuron can be adjusted. This figure shows rate traces with a target rate of 800 kHz with a different amount of used connections.

Homeostasis Applied on a Neuron with Background Input

In the previous measurements, the neuron received only input from neurons of the SoN network. This input regulated the rate of the neuron towards the target rate. In a SEM experiment this would be different. The neuron would receive additional input which should be compensated by the homeostatic rate adaptation. If we apply homeostasis to a neuron, a defined target rate should be maintained independent from the additional input which the neuron receives. In Figure 4.12 we can see how this maintenance can be achieved. The plot shows two different traces of a neuron rate over time. The neuron receives spike input with a non-constant rate over time which is called background. If we apply homeostasis to the neurons frequency, this background will be compensated and a constant rate is maintained. We can see that after approximately 200 ms the background input decreases. If we apply homeostasis to the neurons rate, the rate decreases as well after 200 ms but increases again shortly after and is kept at the target rate. In this case we used a linear adaptation with a step size of one per weight update. As described previously, this process can be accelerated or slowed down if necessary. The background source was represented by spikes which were sent from the host computer during emulation.

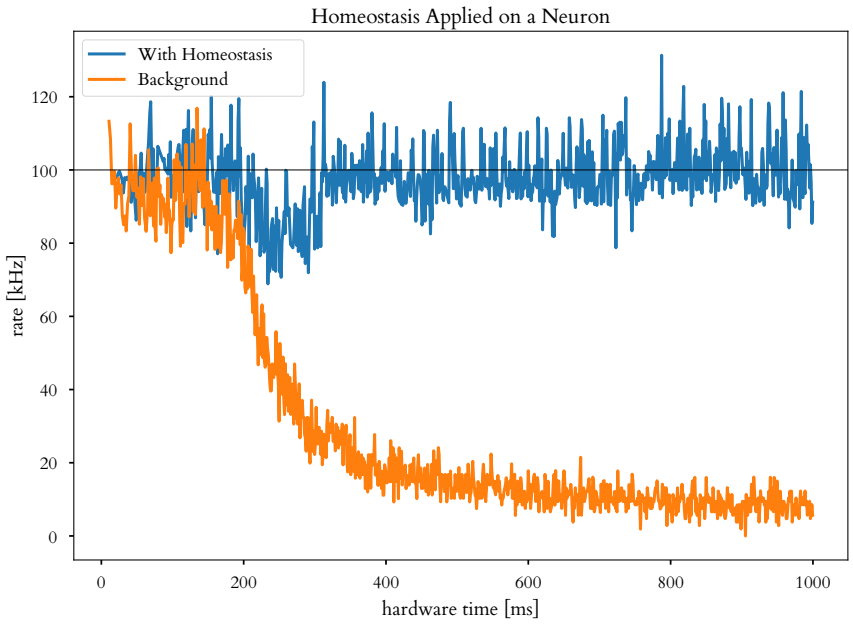


Figure 4.12.: A neuron receives a background input with a decreasing rate. Homeostasis applied to the neuron compensates the varying disturbances from the background input resulting in a constant rate.

We presented in this chapter different approaches how a rate adaptation of a neuron can be achieved using a SoN network as an on-wafer spike source during a closed–loop emulation on the BrainScaleS system. These different adaptation methods aimed at an acceleration of the rate adaptation. Nevertheless, care was take that the presented methods can be regulated in their adaptation speed as it is an crucial part for later SEM experiments to adjust the homeostatic rate adaptation correspondingly. Different other methods are conceivable which will be addressed in Section 6.

5. Outlook — Closed–Loop Spike–Based Expectation Maximization

This chapter will give an outlook towards spike–based expectation maximization experiments on the BrainScaleS system in a closed–loop operation mode. We will discuss how the methods which were implemented during this thesis can be used to conduct such an experiment.

5.1. Experimental Setup

As proof of concept, we will use a minimal network of three neurons encoding three different patterns of a 2 x 2 input layer. The three neurons of the cause layer will try to find the hidden cause behind patterns which are encoded in spike trains. The network will receive these spike trains via an input layer consisting of four neurons y_i. Three possible patterns are shown in Figure 5.1. Each of the four cells of these patterns is either active or inactive which will be represented by different mean rates of the corresponding spike train.

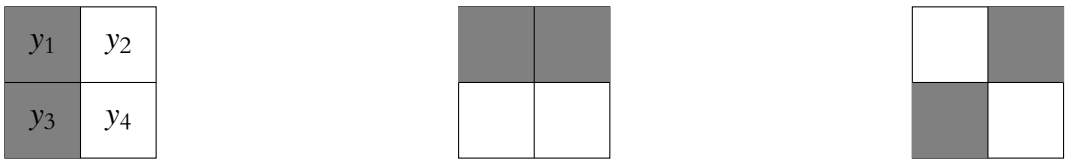


Figure 5.1.: For the SEM experiment, we need three different patterns which will be encoded in spike trains and passed to the network through an input layer of four neurons y_i. The four cells of the shown patterns can be active or inactive. This can be represented by different mean rates of the spike trains which are sent to the corresponding neurons of the input layer.

The input layer neurons are connected to the cause layer neurons via weights V_{ki} which are chosen randomly at the beginning and will be updated during emula-

tion as described in Section 3.1. The cause layer neurons are connected with each other via strong inhibitory weights W_{kl} to form a winner-takes-all structure. These weights will be kept fixed. To learn the patterns, we present them randomly to the network, if z_k fits better to this input than the other neurons $z_j (j \neq k)$, it will suppress the other neurons – V_{ki} will learn more of the current input than the other weights $V_{ji}, j \neq k$. After the weight update, the given neuron will be slightly specialised on input from the current pattern. Furthermore, we need to apply homeostasis on the activity of cause layer neurons. If the mean rate of one neuron increased, this neuron would suppress the other neurons because of the winner-takes-all structure. In this case, other neurons would not be able to learn as weight updates are only performed upon post–synaptic spikes. There are – roughly speaking – three processes which evolve in parallel during the experiment – each on a different time scale. On the fastest time scale is the presentation of the different patterns to the network. The homeostatic rate adaptation of the cause layer activity evolves slower as it hast to be maintained across all the different input patterns shown. Finally, the weight updates caused by the learning rule have to operate on the slowest time scale since otherwise the homeostatic control circuity would be unable to maintain an equilibrium of activity. The fine-tuning of these processes will be one of the difficulties of an implementation on the BrainScaleS system.

5.2. Possible Implementation on the BrainScaleS System

The experiment will be performed in the closed–loop operation mode of the BrainScaleS system. After the network is mapped onto hardware, we can present different patterns by sending spike packets from the host to the corresponding input neurons (see 3.3.2). The update rule for the weights V_{ki} (see Equation 3.1.2) depends on the rate of cause layer neuron z_k and the rate of input neuron y_i. The calculations for the weight update will be performed on the host computer. Therefore we can determine the rate of the corresponding neurons y_i and z_i as it was done with a single neuron for the rate adaptation in Section 4.3.2. Because of the winner–takes–all structure of the cause layer only one of the neurons in this layer will spike. Therefore only the weight updates for V_{ik} have to be calculated. How fast the weight adaptation process evolves depends on the chosen learning rate which has to be tuned to satisfy the timing constraints explained in the previous section. After the weight update for V_{ik} is calculated, we can change the digital weight of the corresponding connection. The maintenance of the cause layer neurons activity can be achieved with a Sea–of–Noise network similar to what was shown in Figure 4.12. The next steps, after the network is mapped on the hardware, are to implement the learning rule

and measure the resulting correlations. Finally, the learning rates have to be tuned.

The described network in this section learns the hidden cause of small 2 x 2 pictures. It is also possible to scale this experiment up and to increase the number of input neurons to learn more advanced patterns or to increase the amount of cause layer neurons to introduce more patterns. It is also important to mention, that learning such small pictures as shown in Figure 5.1 puts a huge burden on the single synapses. Another very promising approach was shown by Guo et al. [2017]. They stacked several SEM networks to form a *Hierarchical Bayesian Inference and Learning Spiking Neural Network*. With this approach, they were able to train their stacked SEM network on the MNIST dataset of handwritten digits. An implementation of this approach on the BrainScaleS system in the future is conceivable. The single SEM networks could be distributed over a whole wafer of the system. Problems could arise for the calculations of the update rule and the weight updates for the homeostatic rate adaptation which are done on the host computer. Because the computations have to keep up with the network emulation, it could be necessary to use more than one computer to perform the required updates.

6. Discussion

The goal of this thesis was to implement and test the necessary mechanisms for a Spike–Based Expectation Maximization experiment. These were: weight updates from the host with the required computations of the updates on the host, a method to introduce stochasticity into a network and a homeostatic rate adaptation. Therefore, we used the closed–loop operation mode of the BrainScaleS system to communicate in real–time between the host computer and the BrainScaleS system during emulation — forming both systems into a hybrid system.

First, we characterised the weight setting process. We showed that this process takes $52 \pm 14\ \mu s$ until the digital weight is set on hardware after the host triggered a weight change (see Section 4.2). We could not determine if this process can be improved because the reason for this delay is a combination of different communication latencies: between the host and the FPGA as well as between FPGA and the corresponding HICANN chip. Also, software latencies contribute to this delay. An analysis of these communication processes is necessary for the improvement of this weight setting process. The determined duration defines a round–trip time because it includes the communication from the neuromorphic system back to the host computer. To find the actual period for realising a weight update on hardware, the communication time back to the host has to be subtracted from the determined duration. In case of a SEM experiment, we have to calculate weight updates on the host based on information which the host receives from the neuromorphic hardware. Therefore, the communication time has to be considered as well. The problem of this communication overhead can be minimized with the developed HICANN–DLS prototype chip. This chip uses an on–chip plasticity processing unit which can be used to implement different plasticity rules.

A method to regulate the activity of a neuron was implemented (see Section 4.3). The controlled neuron is connected to a Sea-of-Noise network which acts as an on-wafer spike source. During emulation, we adjust the digital weights of connections from the SoN network to the neuron or order to control the frequency. We showed that the communication speed between both systems is fast enough to regulate the neuron in biological meaningful time domains. Different types of rate adaptation methods were implemented and presented in this thesis: linear adaptation, binary search and an adaptive step size. It was shown that the last two methods accelerated the rate adaptation compared to linear adaptation. There are other rate adaptation methods conceivable. Building on top of the adaptive step size we could use a mul-

tiplicatively increasing step size. Another possibility is a proportionality controller. There we would adjust the weights proportional to the current error. The problem with this method is it requires error normalisation. Otherwise, it would not be possible to convert the current error, given as frequency, into an adjustment of the digital weights. This normalisation would depend on the current neuron parameters, the amount of connections and the chosen Sea-of-Noise network. Therefore, a change in parameters requires adjustment of the normalisation. A simplified version could use a linear approximation between rate and weights and use the maximum possible rate which can be reached, divided by the highest possible weight setting as a conversion factor. The implemented methods which were shown in this thesis do not need normalisation and are rather straight–forward to use for different setups of Sea-of-Noise networks or neuron parameters Furthermore, we took care that it is possible to increase or decrease the speed of the rate adaptation for the different methods which is essential for homeostatic rate adaptation in SEM–like experiments. A limiting factor when increasing the speed of rate adaptation was the number of hardware connections which have to be changed at every update step. We showed with two different weight update procedures – *Population first* and *Neuron first* – that we can accelerate this process with the second procedure (see Section 4.3.2). However, Neuron-first procedure showed a rougher rate trace with a larger variance. The reason for this is that we use fewer neurons which are connected to the regulated neuron. This leads to larger statistical fluctuations. A solution could be a compromise between both procedures. A possibility would be to limit the number of different synapses which will be changed per update. If the amount of digital weights which we want to increase is larger than a defined quantity N, the digital weights will be equally distributed over the N connections with the lowest current weight.

During this thesis, we demonstrated different weight update methods with the same Sea-of-Noise network configurations. A next step should be to analyse the behaviour with larger Sea-of-Noise networks. If the network parameters are chosen sensible, neurons within larger SoN networks can spike with a lower variance. This would be useful for the rate adaptation process, as a lower variance could enable more precise rate adjustments.

Finally, we presented a minimal version of a SEM experiment and how it could be implemented on the BrainScaleS system. All necessary mechanisms for such an experiment were verified on this system or, if not existing, implemented during this thesis. In a next step towards a closed–loop SEM experiment, these mechanisms have to be combined. Furthermore, the corresponding learning rates have to be tuned to satisfy the required timing constraints which are caused by the different time scales in which the involved processes operate, as described in Section 5.2.

The shown experiments in this thesis were the first using the closed–loop oper-

ation mode on the current hardware version (HICANN v4) and mark a first step towards closed–loop SEM experiments on the BrainScaleS system. However, SEM experiments are just one possible use case for this operation mode. Especially the field of robotics opens a promising application area for biologically inspired closed–loop controllers where the sensory input could be processed on neuromorphic hardware like the BrainScaleS system. An ambitious but outstanding project would be to use several individual robots. The sensory input of each robot could then be processed on one of the wafer of the BrainScaleS system. Furthermore, they could interact with each other over the of–wafer network. It would also be possible to simulate this in a virtual environment, several agents could be used to solve for example a maze. The individual agents could then learn from each other to improve the behaviour of the whole group of agents.

Bibliography

Prof. Katrin Amunts and Christoph Ebell. *Overview Brochure of the Human Brain Project*. Human Brain Project Coordination Office, 2017.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Oliver Breitwieser. Towards a neuromorphic implementation of spike-based expectation maximization. Masterarbeit, Universität Heidelberg, 2015.

Kate D Fischl, Gaspar Tognetti, Daniel R. Mendat, Garrick Orchard, John Rattray, Christos Sapsanis, Laura F. Campbell, Laxaviera Elphage, Tobias E. Niebur, Alejandro Pasciaroni, Valerie E. Rennoll, Heather Romney, Shamaria Walker, Philippe O. Pouliquen, and Andreas G. Andreou. *Neuromorphic self-driving robot with retinomorphic vision and spike-based processing/closed-loop control*. IEEE, 2017.

Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, New York, NY, USA, 2014. ISBN 1107635195, 9781107635197.

Shangqi Guo, Zhaofei Yu, Fei Deng, Xiaolin Hu, and Feng Chen. *Hierarchical Bayesian Inference and Learning in Spiking Neural Networks*. IEEE, 2017.

S. Jeltsch. *A Scalable Workflow for a Configurable Neuromorphic Platform*. PhD thesis, Universität Heidelberg, 2014.

Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2011. ISBN 1441982361, 9781441982360.

Eric C. Müller. *Novel Operation Modes of Accelerated Neuromorphic Hardware*. PhD thesis, Universität Heidelberg, 2014.

Bernhard Nessler, Michael Pfeiffer, and Wolfgang Maass. *STDP enables spiking neurons to detect hidden causes of their input s*. Curran Associates, Inc., 2009.

Fernando Perez-Peña, J. Antonio Leñero-Bardallo, Alejandro Linares-Barranco, and Elisabetta Chicca. *Towards bioinspired close-loop local motor control: A simulated approach supporting neuromorphic implementations*. IEEE, 2017.

Mihai A. Petrovici. *Form vs. Function - Theory and Models for Neuronal Substrates*. PhD thesis, Universität Heidelberg, 2015.

Thomas Pfeil, Jakob Jordan, Tom Tetzlaff, Andreas Grübl, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Effect of heterogeneity on decorrelation mechanisms in spiking neural networks: A neuromorphic-hardware study. *Phys. Rev. X*, 6:021023, May 2016. doi: 10.1103/PhysRevX.6.021023. URL `http://link.aps.org/doi/10.1103/PhysRevX.6.021023`.

J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010.

Felix Schneider. Implementation of a closed–loop homeostasis on the brainscales system. Internship report, Universität Heidelberg, 2018.

A. Appendix

A.1. Acronyms

API	Application Programming Interface
AdEx	Adaptive-Exponential Integrate-and-Fire
FPGA	Field Programmable Gate Array
HICANN	High Input Count Analog Neural Network
SEM	Spike-Based Expectation Maximization
VLSI	Very-Large-Scale Integration

A.2. Parameters

A.2.1. Measurements with a Continuously Spiking Neuron

Table A.1.: Parameters for the measurements of the following Figures: 4.2, 4.3b, 4.5

Parameters	Values
Wafer	21
HICANNS	300 / 324
Neuron parameters	"tau_refrac": 0.1, "tau_m": 2.0, "v_thresh": -10.0, "e_rev_E": 60.0, "v_reset": -70.0, "cm": 0.2, "e_rev_I": -100.0, "tau_syn_E": 5.0, "tau_syn_I": 5.0, "v_rest": -20.0
Firing Neuron parameters	"tau_refrac": 0.1, "tau_m": 2.0, "v_thresh": -20.0, "e_rev_E": 60.0, "v_reset": -35.0, "cm": 0.2, "e_rev_I": -100.0, "tau_syn_E": 5.0, "tau_syn_I": 5.0, "v_rest": -10.0

A.2.2. Measurements with a Sea–of–Noise network

Table A.2.: Parameters for the measurements of the following Figures: 4.6, 4.7, 4.8 4.9b, 4.10, 4.12

Parameters	Values
Wafer	21
HICANNS	300 / 324
Neuron parameters	"tau_refrac": 0.01 "tau_m": 0.1 "v_thresh": -20.0 "e_rev_E": 60.0 "v_reset": -40.0 "cm": 0.2 "e_rev_I": -100.0 "tau_syn_E": 1.0 "tau_syn_I": 1.0" v_rest": -30.0
Sea-of-Noise network	#Neuronen: 100 w^{inh} =1 presynaptic Partner: 10 Neuronparameters: "tau_refrac": 4 "tau_m": 2.0 "v_thresh": -20.0 "e_rev_E": 60.0 "v_rese t": -70 "cm": 0.2 "e_rev_I": -100.0 "tau_syn_E": 1 "tau_syn_I": 1 "v_rest": -10.0

Statement of Originality (Erklärung):

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 26 June 2018 _____