Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Alexander Nock

born in Tuttlingen (Germany)

2021

Migration and Enhancement of the Advanced Lab Course on Neuromorphic Computing

This Bachelor Thesis has been carried out by
Alexander Nock
at the Kirchhoff Institute for Physics in Heidelberg
under the supervision of
Dr. Johannes Schemmel

Abstract

BrainScaleS-2 is an analog neuromorphic computing platform currently developed by the Electronic Vision(s) group in Heidelberg. The research approach behind this technology uses spiking neural networks for information processing in a similar way to the human brain. An introduction to this topic is provided by a series of experiments in the framework of the advanced lab course for physics. Since those are currently performed on a considerably older neuromorphic chip, the thesis aimed to migrate the existing experiments onto BrainScaleS-2. Given the modified specifications resulting from a new hardware, it was necessary to adapt the experiment code accordingly. The thesis evaluates the obtained results ensuring that the intended purpose of every task could be sustained after the migration. Motivated by demonstrating a more practical application, a new task introducing a sudoku solver was designed. The functionality of the corresponding neural network is explained and evaluated.

Zusammenfassung

BrainScaleS-2 ist eine analoge Plattform für Neuromorphes Rechnen, die gegenwärtig von der Electronic Vision(s) Gruppe in Heidelberg entwickelt wird. In ähnlicher Weise wie das menschliche Gehirn verwendet der Forschungsansatz hinter dieser Technologie gepulste neuronale Netze für die Informationsverarbeitung. Eine Einführung in dieses Thema wird mit einer Versuchsreihe im Rahmen des Fortgeschrittenen-Praktikums angeboten. Da diese gegenwärtig auf einem deutlich älterem neuromorphen Chip ausgeführt wird, hat sich die Bachelorarbeit zum Ziel gesetzt, die bestehenden Aufgaben auf BrainScaleS-2 zu migrieren. Die mit einer neuen Hardware verbundenen Änderungen der technischen Einzelheiten machen es notwendig, den Programmcode der Experimente entsprechend anzupassen. Die Bachelorarbeit wertet die gewonnenen Ergebnisse aus, um sicherzustellen, dass der Zweck jeder Aufgabe nach der Migration erhalten werden konnte. Mit der Motivation eine praxisnähere Anwendung zu demonstrieren, wurde ein Sudoku-Löser erstellt und in Form einer neuen Aufgabe eingführt. Die Funktionsweise des zugehörigen neuronalen Netzes wird erklärt und ausgewertet.

Contents

1 Introduction

Inspired by the functionality of the human brain, Artifical Neural Networks (ANNs) have been developed to process input with a network of neurons. The research on ANNs has shown that they can be used in a broad range of applications [1] and are outperforming conventional computers in specific tasks like image recognition [2] or speech processing [3]. When trying to emulate the functionality of a brain more realistically, the firing nature of the neurons has to be considered. This leads to Spiking Neural Networks (SNNs) for whose description it is necessary to solve a complex system of differential equations. Therefore, neuromorphic hardware has been designed which provides an alternative way of computing by directly emulating the biological behavior in integrated circuits. This does not only yield a higher processing speed for some systems [4] but also has the potential to be more energy efficient than conventional computers [5].
In Heidelberg, the Electronic Vision(s) group has been developing neuromorphic architecture for more than 15 years with BrainScaleS-2 being the latest generation thereof. For physics students the work group offers a two-day introduction to how neuromorphic hardware operates and what it is capable of. This is provided in form of an experiment in the advanced lab course for which an older neuromorphic chip is used at the moment. Given the newer and improved hardware, the migration of the lab course seemed expedient and therefore was addressed by this thesis. For every task, the main objective was to find a suitable operating point for the neurons after the code had been rewritten. Moreover, it was necessary to check whether the purpose of every task is still sustained on the new hardware. Thereby, the software scripts had to be easy to comprehend for students unfamiliar with neuromorphic hardware and potentially little programming experience. The first two tasks of the experiment on neuromorphic computing deal with observing and calibrating analog parameters of a single neuron circuit. They have already been migrated onto BrainScaleS-2 during my internship and therefore can be found in the according report [6]. This thesis will further follow the structure of the advanced lab course and migrate the remaining tasks which use different networks of interconnected neurons to obtain an application of neuromorphic computing. Thereby the different networks are defined by their respective connections and also the weights thereof. Lastly, a new task that uses a neural network to solve sudoku puzzles has been created and evaluated. It presents a conceivable application of neuromorphic computing for a widely known task.

2 Theory & Methods

2.1 Biological Background

With over 10^4 units per cubic millimeter, neurons form the elementary components of the human brain. The input from ca. 10^4 other neurons [7] arrives at their dendrites (fig. 1) and travels to the soma which is the neuron body and functions as the processing unit. Their output is then forwarded by the axon which is connected to the dendrites of the subsequent neurons by synapses. In the neuron itself the information is processed using the membrane potential which is given by the difference in the potential inside the soma and the more positively charged outside. With no input, the membrane potential is drawn towards a constant value which is called leakage potential. With excitatory or inhibitory input, ion-pumps are activated causing a rise or fall on the membrane, the postsynaptic potential (PSP), which then passively returns back to its initial value by diffusion processes. Thereby, the soma integrates all incoming signals to a superposition of exponential functions as shown in the right part of figure 1. If a certain threshold potential is crossed, the membrane potential increases significantly and sends the so-called action potential onto its axon. Then inside the neuron temporarily a reset potential below its resting potential is taken and after a refractory period of typically a few milliseconds the leakage potential is aimed again.

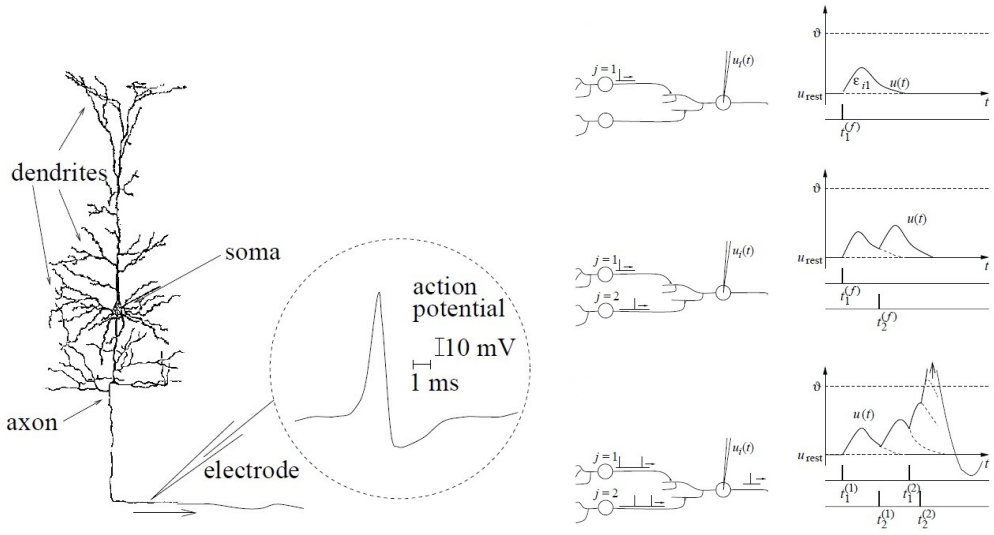

Figure 1: *Left:* Schematic of a single neuron in a drawing by Ramón y Cajal.
Right: The superposition of the input spikes (postsynaptic potentials) in the soma which eventually results in an output spike (figures taken from [8]).

These processes in all of the roughly 10^{11} neurons of our brain yield a fast way of parallel computing with only a comparable low need of energy. Therefore, this essentially different way of processing information (compared to the von Neumann architecture) is subject of software and hardware modelling. Furthermore, the way of information processing in the human brain is well understood on the level of neurons, but does not explain all phenomena observed on network level [9]. Hence large-scale brain paradigms are emerging and with modelling the interaction of numerous neurons, further insights in this field are strove for.

2.2 Networks and Models

The most elementary neural network is a simple perceptron [10] which consists of an input layer connected to an output layer and for instance could be used to build an AND gate. For slightly more advanced tasks like an XOR gate an extra layer of (hidden) neurons in between is needed which makes the network a multilayer perceptron. If there are connections to the same layer or the previous layers it is called a recurrent neural network (RNN). Otherwise, the information flows only in one direction which is captured by the name feedforward network. All these networks are also used in machine learning, but there they are static and don't emulate the spiking behavior of biological neurons. This is different for spiking neural networks (SNN) [8] which are used for the advanced lab course. There, the time component plays an important role and the information is forwarded by the output given as a series of spikes.

A quantitative description of the electric circuit that a spiking neuron can be seen as, is delivered by the model from Hodgkin and Huxley (1952) [11]. In order to describe the ion channels and their current flows, they use coupled differential equations which are computationally expensive to solve due to their non-linearity. However, there are also more basic models which are used to model SNNs on hardware. For the BrainScaleS-2 hardware this is the *adaptive exponential integrate-and-fire* (AdEx) [12] model which is based on the simpler *leaky integrate-and-fire* (LIF) model [8] and sufficiently explains the hardware for the advanced lab course.

2.3 Leaky Integrate-and-Fire (LIF) Model

In order to electronically describe the basic dynamics of neurons, the leaky integrate-and-fire model (LIF) with current based synapses [13] is used throughout the thesis. Mathematically this is expressed by the following differential equation for the membrane voltage V_m:

$$C_m \frac{dV_m(t)}{dt} = g_{leak}(E_{leak} - V_m(t)) + I_{exc}(t) + I_{inh}(t) \tag{1}$$

In biology, the membrane separates the differently charged inside and outside of the neuron and thereby acts as a capacitor. Considering that, it is crucial to have the capacitance C_m which can be charged or discharged. This happens with excitatory or inhibitory input given in form of the time-dependent synaptic currents I_{exc} and I_{inh}. Subsequently, the voltage V_m over the membrane capacitance C_m is pulled towards the leakage potential E_{leak} over time. This happens by current flowing through the resistance with leak conductance g_{leak}.

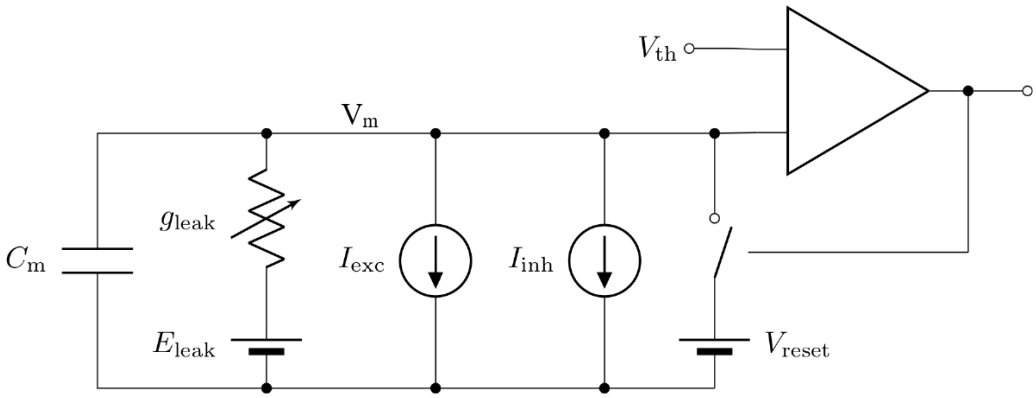

Figure 2: Circuit based on the leaky integrate-and-fire model with current based synaptic input (figure taken from [14]).

If the potential on the membrane crosses the threshold voltage V_{th}, the comparator outputs a voltage which potentially serves as a spike for the following circuits. Simultaneously, the membrane voltage switches to the reset voltage V_{reset} for the duration of the refractory time constant τ_{refrac}.

2.4 The HICANN-X Chip

Currently the advanced lab course is performed on the Spikey chip which has 384 neurons in total. In contrast, the platform mainly used by the working group is the new BrainScaleS-2 (BSS-2) architecture [15] which will be adopted for the lab course. It contains the HICANN-X chip with its total of 512 AdEx neurons. Those are distributed over four equally sized quadrants which have own analog parameter storage circuitry memorizing the parameters of their 128 neurons (cf. figure 3). All these parameters have a 10-bit precision when their digital values are produced in the parameter memory.

The connections between the neurons are managed by the synapse arrays where each of the 256 rows is connected to a synapse driver. Those are used to process input coming not only from other on-chip neurons but also from external sources. Every neuron is then connected to the end of a synapse array column, i.e. a total of 512 x 256 = 131 072 synapses are available where an individual weight can be chosen with a 6-bit resolution.

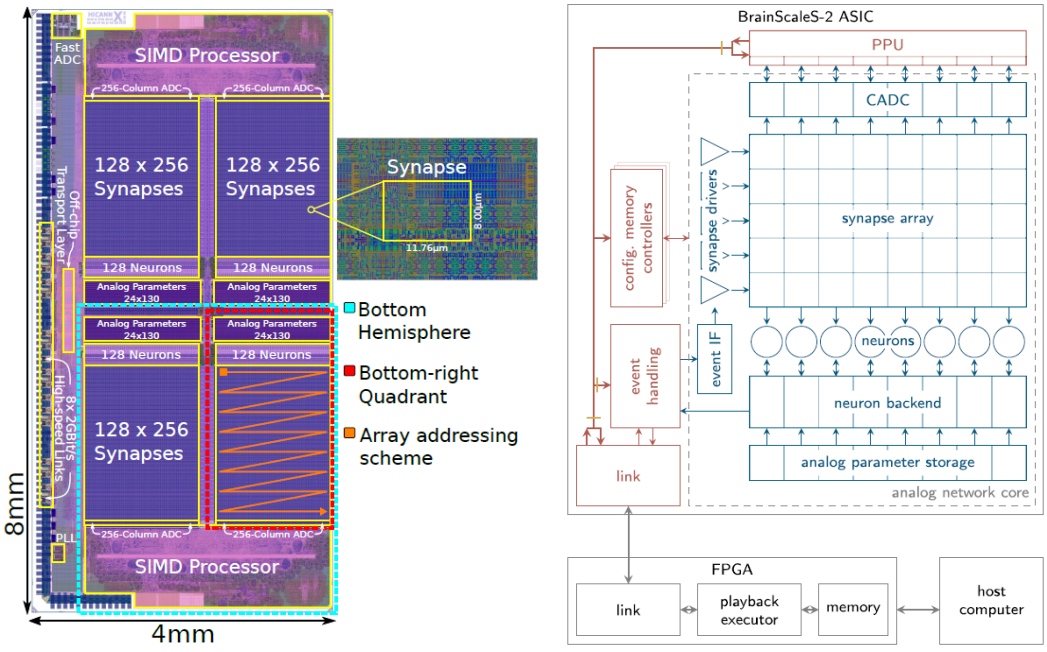

Figure 3: Schematic setup of the BrainScaleS-2 chip (*left*, taken from [16]) with the diagram of one quadrant and its communication to the host computer (*right*, taken from [17]).

The voltage on the membrane is measured by the on-chip membrane analog-to-digital converter (MADC) which explains why the voltages are obtained in digital values. This can be translated back to analog values with a reference voltage digital-to-analog converter (DAC) which needs to be connected to the MADC and sweeps through the measurable voltage range.
The main difference between the two chip generations are the Plasticity Processing Units (PPUs) that are exclusively on the BSS-2 setup. They are microprocessors working with SIMD (single instruction, multiple data) instructions which for example can be used to calculate the weight updates for several neuron connections simultaneously. This is required when a learning rule for the weights is used to take past firing behavior into account. The benefit compared to calculating the new weights on an external computer is that the weights can be updated during a hardware run which is necessary to prevent a high latency and results in a faster experiment execution.
Important to mention is that the synapses and neurons themselves show fixed-pattern noise due to their manufacturing imperfections which mainly result from variations in the size and doping of the transistors. That is why calibrations are applied which yield a significant improvement, but never can compensate all the differences. Therefore, the used network models are required to be robust against noise to a certain extent.

2.5 Software

For novice users like the students in the advanced lab course it cannot be expected to operate with an unknown system on a level close to the hardware. Therefore, it is necessary to provide an application programming interface (API) with a higher level of abstraction that also has numerous use cases for expert users. With this motivation, PyNN [18] has been implemented for BrainScaleS-2 [14] and will be used when writing the tasks for the advanced lab course. It mounts on top of several other layers of software that had to be developed beforehand to operate on the new hardware. PyNN is a Python package that can be used to build neural networks at a high-level of abstraction and works independently of the backend in place. Its user-friendliness is reflected by the intuitive way a neural network can be set up. The experimenter can select any neurons and group them together to populations. Those can be connected to other populations with so-called projections using different types of synapses. It is also possible to get more detailed access to the network's properties and change pa-

rameters within the neurons or choose among different types of synapses. Currently, the backend of PyNN on BrainScaleS-2 only allows to operate in units of the digital-to-analog converter (DAC) that is used when reading the values in the parameter memory. For the evaluation of the tasks these parameters will partly be transformed to hardware values. In the long-term, the aim is to set all parameters in biological values like it is realized in the PyNN version of Spikey.
The general control flow starts with the network that is set up in PyNN, including the assigned duration of the hardware run and all observables that are selected for recording. The backend of PyNN then creates the playback program which contains all configurations and instructions. This program is subsequently send from the host computer to a field-programmable gate array (FPGA) which handles the communication between the host and the chip. After the experiment is executed on the hardware, the FPGA receives the experiment results and provides them for a readout by the connected computer.

2.6 Advanced Lab Course

The advanced lab course is a mandatory module for all undergraduate physics students in Heidelberg. It offers numerous series of experiments covering nearly all areas of physics. The one migrated in this thesis is on the topic of neuromorphic computing (being based on [19]) and is designed for the typical duration of two days. It is divided into seven tasks each being split in several subtasks. Some of the subtasks are optional for faster students or those who choose to write a report on the experiment. To each task, software scripts are provided which are meant to be modified by the students in the course of the experiment and therefore should be understood by people unfamiliar with the topic. For the Spikey chip, an oscilloscope is used to directly observe the membrane voltage in the laboratory. On the prospective setup (fig. 4) this could be done as well, but is no longer required because the previously described MADC on BrainScaleS-2 provides the same functionality. In the picture, the xBoard surrounding the HICANN-X chip manages the supply voltage and also carries the analogue and digital periphery. It is connected to the cube setup which is intended for the communication to the host computer via 1Gbit Ethernet per FPGA.

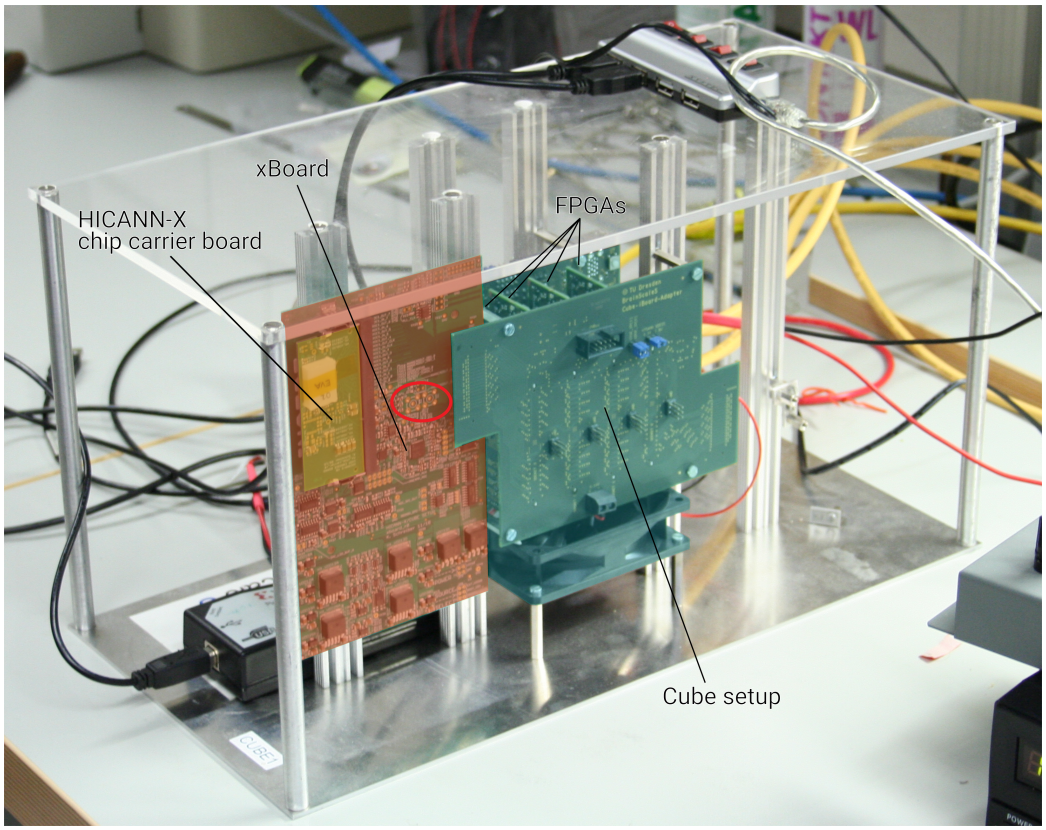

Figure 4: The current test setup of the BrainScaleS-2 system which will be used for the advanced lab course (modified from [20]). The HICANN-X chip with its carrier board is supplied with voltage by the xBoard. The communication with the host works via the four FPGAs which are contained in the cube setup. An oscilloscope can be connected to the two analog readout channels (circled in red) to observe the membrane or parameter voltage.

3 Results

The general steps for migrating a task onto the new BrainScaleS-2 hardware are firstly to make sure that the chip is in theory able to perform the required operation. Then the code needs to be adapted to run on the new hardware which for instance means that parameters are translated to digital values and adjusted to a wider parameter range. Since PyNN is work in progress, it might also be necessary to program a certain functionality if it can be realized within the scope of this thesis. This is the reason why task four is not yet migrated as it requires a new type of synapse that is able to learn from previous input. The according implementation into PyNN for the new hardware requires the handling of routing that would be too extensive for the thesis.
After the initial results are checked for their usefulness, usually certain parameters need to be swept to find a suitable working point. Certain tasks require the use of a calibration to compensate for the fixed-pattern noise of the hardware. The values for it are obtained by automated calibrations that run overnight on the setups. The set of parameters for the respective chip is then load and applied using PyNN. Subsequently, plots have to be adapted or newly created and comments for students are added to understand how the software operates. The result of all these steps will be presented in the following and will show to which extend it was possible to transport the functionality onto the new hardware. Thereby, the structure is according to the tasks of the advanced lab course and the respective code can be found in the review tool Gerrit (change ID given in table 1). For the fifth task the PyNN script is also exemplary printed in the appendix.

3.1 Task 3: A Single Neuron with Synaptic Input

This experiment is conducted with a single neuron which is connected to synaptic input and should show the decay of the postsynaptic potential (PSP) without firing as shown in the right part of figure 1 (upper plot). In order to prevent spiking, its threshold is set to a high value. In figure 5 the observed shape for one excitatory (left) and one inhibitory (right) input spike coming from one other neuron can be seen. The average over eight spikes (red graph) shows a less noisy behavior. This shape can be modified by parameters set in PyNN. For the version on Spikey these are the two hardware parameters `drvifall` and `drviout` for whom most students

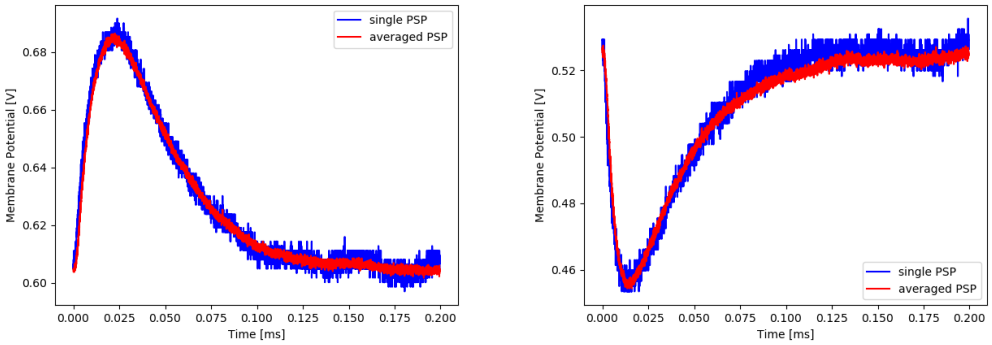

Figure 5: The PSP responding to excitatory *(left)*/inhibitory *(right)* presynaptic input recorded for one neuron. The blue curve shows a single event while the red one shows an average over eight spikes.

have no intuition since they have no direct biological representation and are set in digital values. To account for this circumstance, the synaptic input time constant τ_{syn} and the synaptic input conductance g_{syn} (or more precisely the bias currents that control those quantities) of a calibrated neuron are changed for the experiment on BrainScaleS-2. The left plot of figure 6 shows that a larger bias current for the synaptic conductance increases the amplitude, but doesn't modify the slope. Knowing that the bias current is directly proportional to g_{syn}, this can be explained by the fact that a higher synaptic conductance means a higher postsynaptic current which is directly proportional to the measured voltage. The right plot shows that a larger

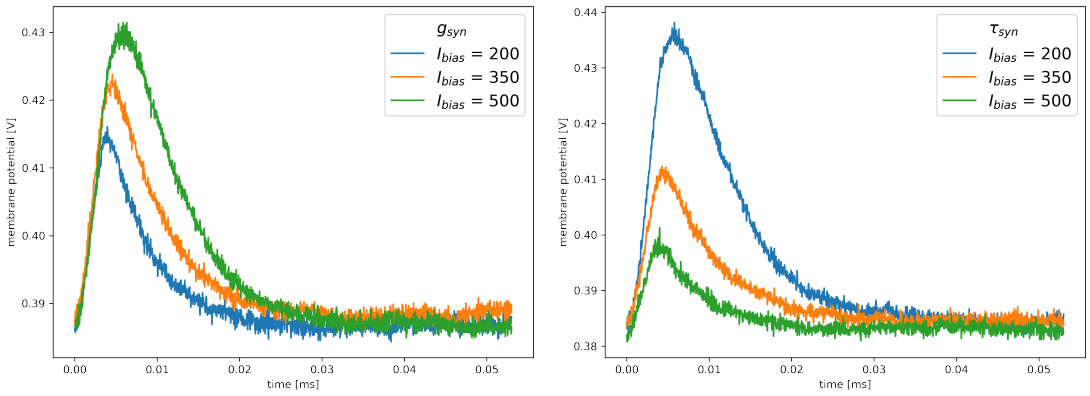

Figure 6: The response of the membrane potential onto a presynaptic spike. The different colors show the dependency of the PSP on the synaptic conductance g_{syn} *(left)* and the synaptic time constant τ_{syn} *(right)*. The bias currents given in digital values are proportional to g_{syn} and inversely proportional to τ_{syn}.

bias current for τ_{syn} yields a smaller peak and a decreased slope. For the explanation it is important to know that the time constant is inversely proportional to its controlling bias current. To start with, the synaptic time constant determines how fast the synaptic current is decaying after an excitatory presynaptic spike [13]. With a larger τ_{syn} the current decays slower and more current flows over the synapse, i.e. the result is a higher rise of the PSP.
In order to examine the fixed-pattern noise across the different synapse drivers, we modify the row of the stimulating synapse and measure the amplitude of the (excitatory) PSPs for one neuron. Thereby, the used calibration accounts for the noise of the neuron, but leaves the synapse drivers unmodified. For a sweep over all 256 synapses the histogram shows a symmetric distribution around a value of 59.3 mV with a standard deviation of 1.4 mV which equals 2.4 % of the mean (fig. 7).

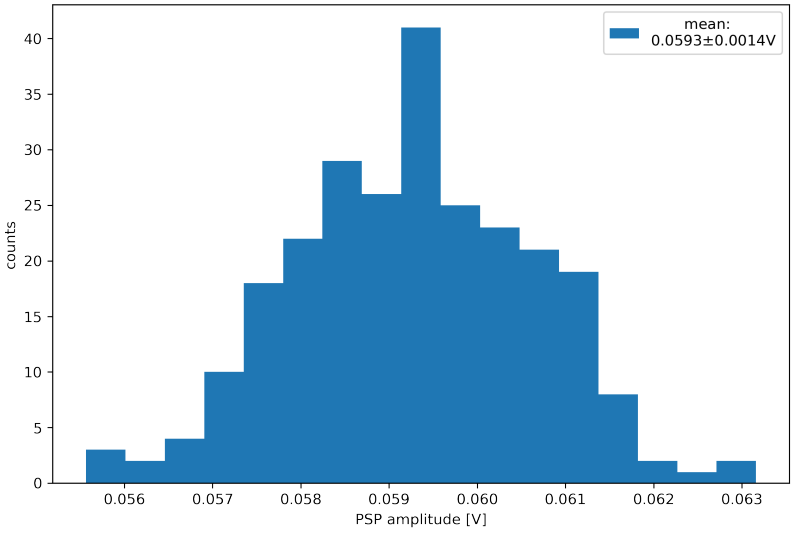

Figure 7: An excitatory PSP has been measured for 256 different synapse drivers. The histogram with the resulting relative heights therefore shows the fixed-pattern noise across synapses which has a value of 2.4 %.

The next subtask emulates the firing of a neuron after it receives multiple excitatory inputs which are stacked together. Therefore, a train of input spikes is sent over the same synapse. To prevent that the membrane is completely drawn back to the leak potential after each stimulation, either the neuron parameters or the temporal distance of adjacent input spikes can be changed. In the experiment the latter should be modified to produce an output spike. A clear visualization of how the decay is

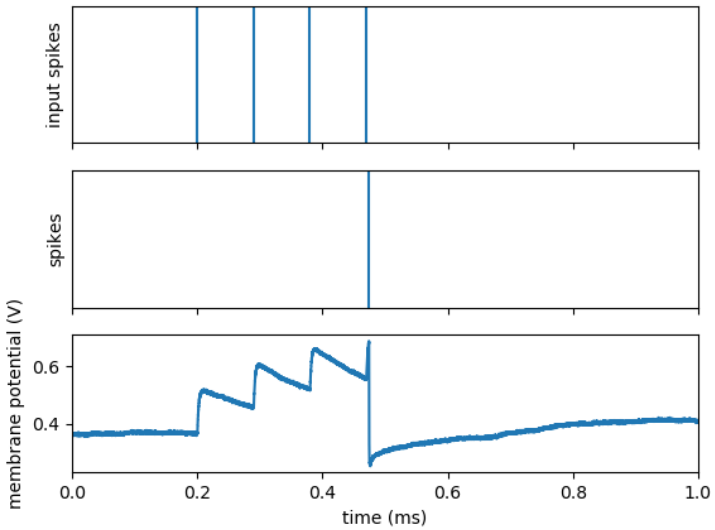

Figure 8: Visualization of the integrative behavior of the membrane. Four successive input spikes *(top)* are send onto the postsynaptic membrane *(bottom)*. The decay of each PSP is interrupted by the next input which eventually rises the potential above the threshold and therefore produces an output spike *(middle)*.

interrupted and the eventual firing as a result of the aggregation has been realized in figure 8. The optional task in the advanced lab course handles the comparison of the fixed-pattern noise observed previously (cf. fig. 7) and the temporal noise occurring in a single run. It doesn't have an illustration so far and therefore figure 9 could be used as a template in the future. The plot shows the different synapses on the x-axis with their respective PSP amplitude (for better visibility only every fifth value is plotted). These values have been obtained by calculating the difference between the minimum and maximum value of a PSP for each of the eight input spikes in a single run and then averaging over them. The mean temporal noise over all 256 synapses (i.e. the average over the standard deviations) is 2.2 mV which equals a deviation of 3.7 %. This fits to the value of 2.3 mV which is obtained for a neuron in 256 single runs with a single PSP using only one synapse driver. In contrary, the fixed-pattern noise is the standard deviation resulting by calculating the mean of the red crosses and represented by the orange tube. With 1.4 mV it makes up 2.7 % of the mean value and therefore it is just two thirds of the temporal noise.

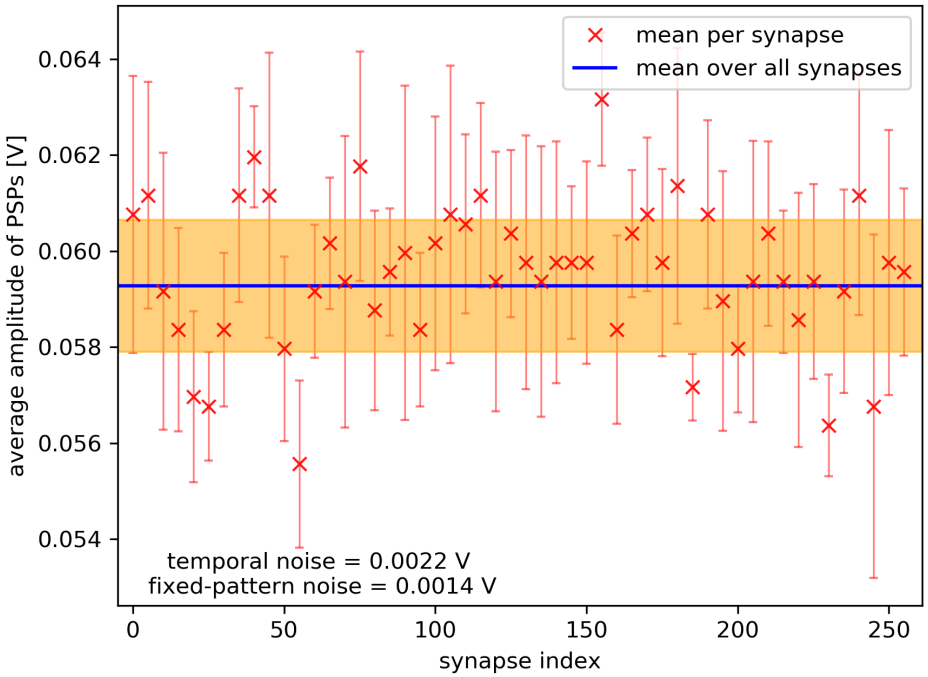

Figure 9: The average PSP amplitude for every synapse is measured for a single run with eight input spikes. The mean temporal noise is the average over all red errrorbars. The fixed-pattern noise is the standard deviation obtained when calculating the mean of all red crosses and represented by the orange tube.

3.2 Task 5: Feed-Forward Networks (Synfire Chain)

This task is the first to use internal connections and thereby setting up a network with neurons receiving their input from other neurons (in contrast to the external spike sources from previous tasks). The very network is a synfire chain with feed-forward inhibition. The setup (fig. 10) is a chain of excitatory populations supplemented by inhibitory neurons where the last neuron population is connected to the first one (cf. the corresponding PyNN script in the appendix). After the first excitatory and the first inhibitory population are stimulated once, the populations are intended to spike one after another. In order to prevent immediate successive spikes from a population, each one is connected to its own inhibitory partner. To start with, relatively large

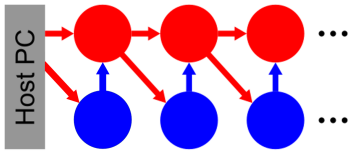

Figure 10: Setup of a synfire chain where excitatory populations are red and inhibitory populations are blue (modified from [21]).

populations with seven excitatory and seven inhibitory neurons are chosen to produce multiple output spikes for every population even if some neurons are not firing. Otherwise, the neurons would need to be very sensitive to be triggered by only one or two spikes. This results in accidental spikes and would destroy the controlled propagation because the network wouldn't be insensitive enough against noise on the membrane. The chosen population size yields a maximum chain length of eight (i.e. 16 populations in total) because of the algorithm for the allocation of the synapse drivers in PyNN when external and internal connections are used [14]. With the unmodified default calibration only one cycle through the chain is observed. The relative short time for the pulse propagating from the first to the last population (less than 0.02 ms) causes that the neurons' potentials are still considerably below the leak potential when a new input spike arrives and therefore no output is produced. To change this circumstance, at first the refractory time constant has been reduced and a continuous firing pattern could be achieved (fig. 11). Likewise, other neuron pa-

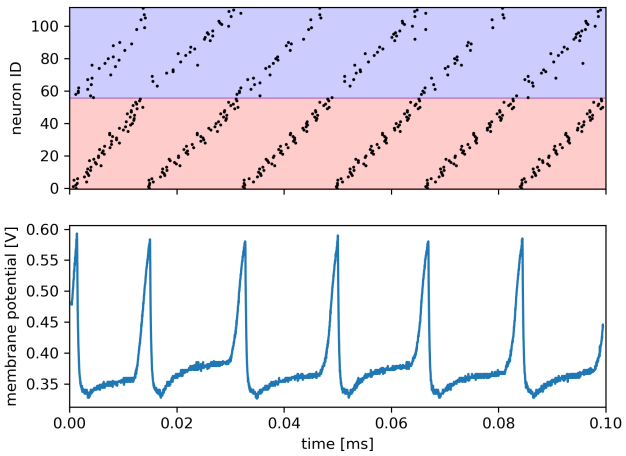

Figure 11: *Top:* The network activity of a synfire chain with excitatory neurons in red and inhibitory neurons in blue. *Bottom:* The membrane voltage over time for an excitatory neuron in the first population.

rameters that have an influence on how fast the membrane is drawn back to its rest potential have been changed with the same result. Exemplary, this is shown for the membrane capacitance in figure 12. There the influence of the membrane capacitance on the membrane time constant causes an increasing propagation time for the input spike. Above a critical value, the membrane is reaching V_{leak} too slow and the synfire chain eventually extinguishes after only a few cycles.

Lastly, the lab course requires to observe to which number of neurons the populations can be reduced until the synfire chain either stops propagating or is firing uncontrolled. The result for the default calibration with the described modifications is found to be a network with 11 inhibitory and 11 excitatory populations which have five neurons each.

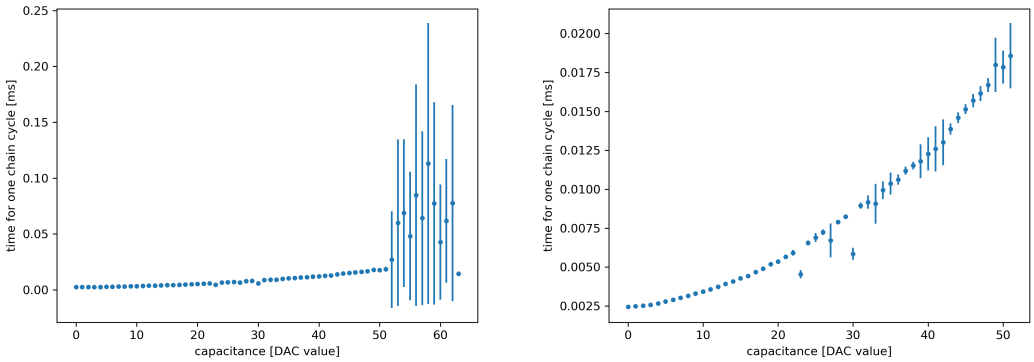

Figure 12: The time for the input pulse to cycle through the synfire chain once, is plotted in dependency of the membrane capacitance. Below a certain value, the two quantities have a positive correlation as it can be seen in the zoomed in plot *(right)*. At the upper end of the x-axis *(left)* this relation vanishes because the neurons' membranes take too much time to be drawn back to V_{leak}.

3.3 Task 6: Recurrent Networks

In the following, the aim is to build a recurrent network which can be used as a noise source for other experiments. To start with, the largest entity with a mostly homogeneous spiking behavior is chosen. For the HICANN-X chip, this is one quarter consisting of 128 calibrated leak-over-threshold neurons (fig. 13, left). The irregularity is then caused by randomly choosing K other neurons as synaptic input for each neuron and setting up those connections with an inhibitory weight w. The result (fig. 13, right) is evaluated by measuring the time between two consecutive spikes for every neuron. More precisely, the coefficient of variation $CV = \mu/\sigma$ indicates the amount of randomness per neuron using the mean and the standard deviation of the averaged ISI. For a homogeneous Poisson process, this dimensionless quantity has a value of 1 while evenly spaced events yield a value of 0 [22].

The observed neurons show CVs that exponentially decrease with their respective firing rate (fig. 14). In analogy to excitatory input [23] this relation could be expected.

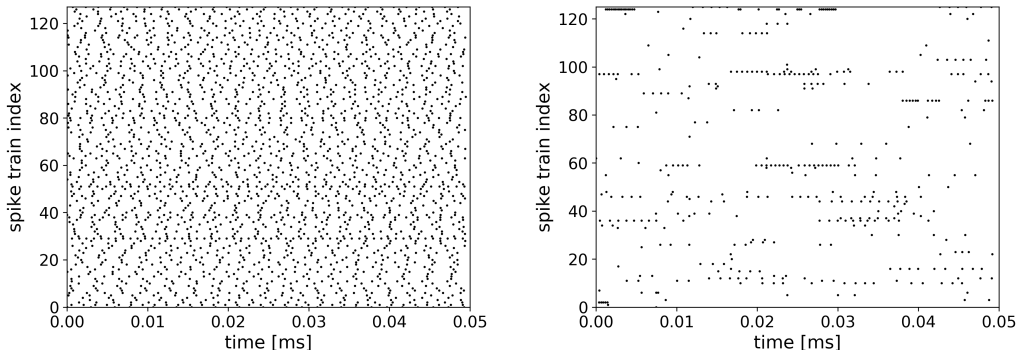

Figure 13: The spiking behavior of all neurons of one block without any connections *(left)* and with randomly drawn inhibitory connections *(right)*.

It is due to the fact that a high firing rate means an ISI close to the refractory time constant which can only occur for a weakly inhibited neuron. Consequently, such a neuron is constantly firing directly after the refractory period is passed and therefore shows no variance in the time between two spikes. In contrast, strongly inhibited neurons show randomness as the refractory time constant has negligible influence on their ISI and the random timing of the input spikes dominate. In order to measure the influence of the number of presynaptic partners K and the connection weight w, those two parameters are swept over their whole range and plotted in a 2D histogram with the firing rate (fig. 15, left) and the CV (fig. 15, right) encoded in the color. There, the firing rate strongly depends on the weight and number of inputs, i.e. more and stronger inhibitory connections suppress the firing activity of almost all neurons to the point that less than 3 % of the initial output spikes are recorded (firing rate of ca. 10 kHz compared to 380 kHz with no inhibitory connections). More precisely, a small number of neurons is firing with their maximum frequency and suppressing the activity of all their postsynaptic partners. On the other hand, the CVs don't depend in the same way on the swept parameters. If the neurons are not inhibited, they are firing with their maximum frequency and therefore show a constant activity which is represented by a low coefficient of variance. With more and stronger inhibitory connections, the randomness increases as the inter-spike intervals are not dominated by the refractory time constant and therefore yield a CV close to 1. With maximum inhibition, the dominating neurons are constantly firing with their maximum frequency while all other neurons are completely suppressed and therefore they have no contribution to the CV.

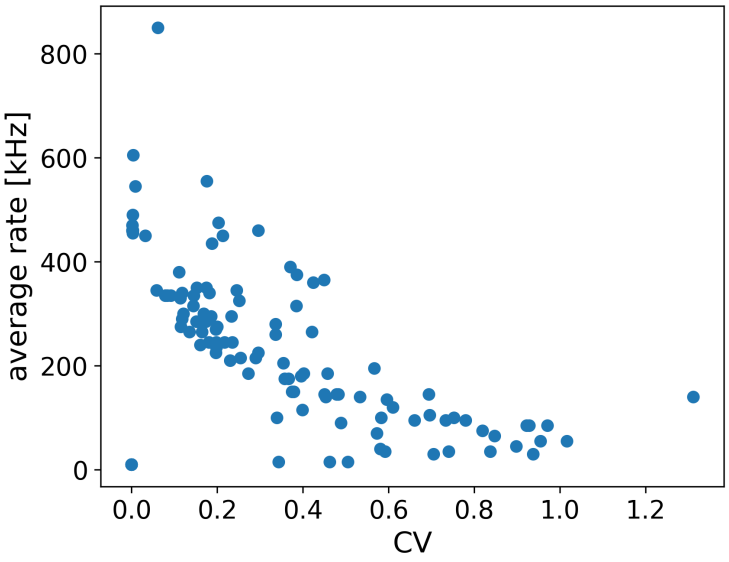

Figure 14: The correlation of the firing rate and the coefficient of variation for the created recurrent network. One data point is obtained by calculating the average ISI of a single neuron and determining the CV and the firing rate with it. A high firing rate leads to a low CV because weakly inhibited neurons constantly fire with their highest possible frequency.

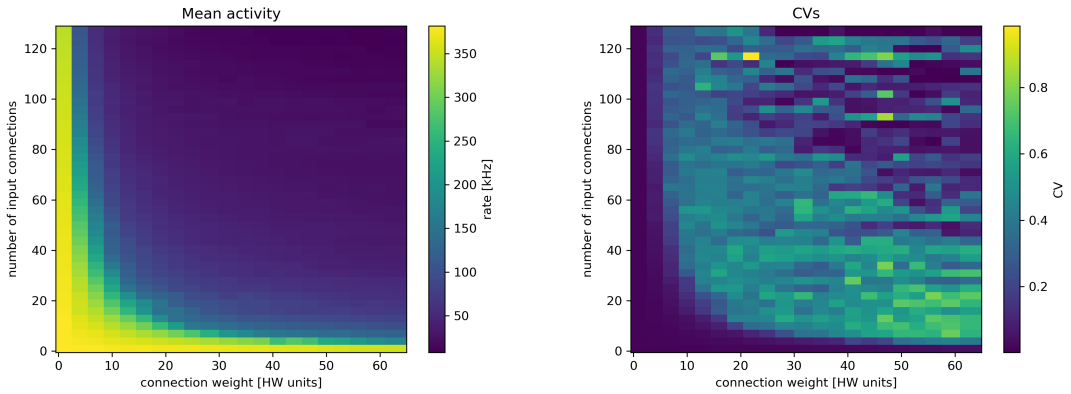

Figure 15: The 2D histograms show the dependency of the firing activities *(left)* and CVs *(right)* on the connection weight (x-axis) and the number of inhibitory presynaptic connections (y-axis). The average value of the measured quantities is encoded in the color bar on the right side of the plots.

3.4 Task 7: A Simple Computation - XOR

The seventh tasks uses a spiking neural network (fig. 16) for the emulation of an XOR gate. The input is given by regular spiketrains labelled as `i1` and `i2` which are echoed by the neuron populations `y1` and `y2`. The spikes are forwarded to the respective population `i1` and `i2` and simultaneously an inhibitory signal is send to the opposite population `i2` and `i1`. The latter is done to ensure that the propagation is suppressed if both inputs receive an input (following the logic of an XOR gate). As pictured in figure 17, the output population correctly fires if only one input population is active.

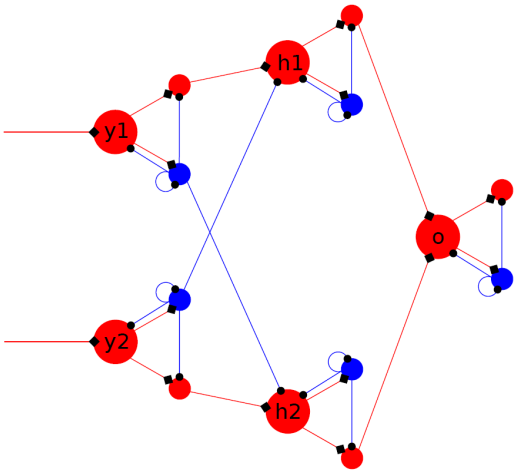

Figure 16: Setup of a spiking XOR gate where excitatory populations are red and inhibitory neurons are blue (modified from [21]).

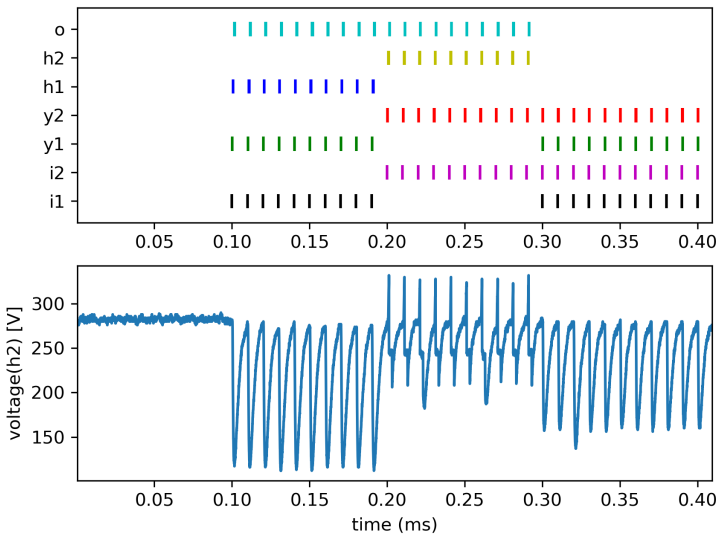

Figure 17: The network activity of neurons mimicking the functionality of an XOR gate. *Top:* The populations are listed on the y-axis and their spikes over time are drawn along the x-axis. With the connections between the populations, the output `o` is suppressed if both inputs `i1` and `i2` are firing. *Bottom:* The membrane of the population `h2` is plotted and shows how it is inhibited during certain time intervals.

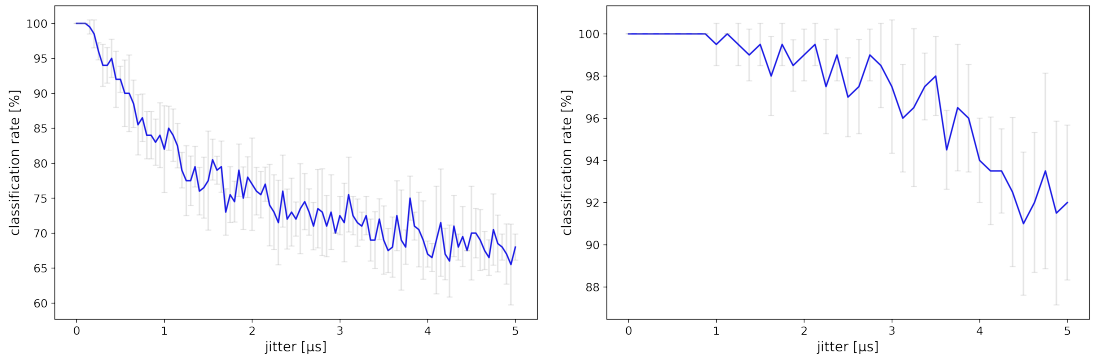

Figure 18: The influence of jitter in the input signals onto the output of the XOR network. The classification rate falls with an increasing amount of jitter *(left)*. A maximal classification rate of 100 % is reliably observed for a jitter up to 0.7 µs *(right)*.

The next subtask observes the effect of jitter on the input signals. It is modelled by a normal distribution around the initial spike times of the input. For a systematic evaluation, the standard distribution is increased stepwise. The limit is set where two adjacent distribution are about to cross their 1σ interval. Since the input is spaced with a distance of 10 µs, this means an upper limit of 5 µs. The result (fig. 18) shows that the classification rate reaches 65 % whereas a random output would be equal to a classification rate of 50 %. This is expected since the refractory time constant and membrane time constant add up to around 5 µs. In this magnitude the membrane potential is no longer significantly inhibited which means that an input spike triggers an output spike and potentially causes a false output.

3.5 New Task: Sudoku Solver

The aim of the new task is to show a practical application of the neuromorphic hardware beyond the binary computation example from the previous task. In conventional computing, a simple sudoku solver would use a brute-force algorithm to iteratively go over all empty cells and check for an allowed number [24]. This is different to the neural network that is build for this task. It works on different cells in parallel and therefore uses four neurons for every cell which represent the numbers from one to four respectively (fig. 19). The reason that a 4×4 sudoku is used instead of the classic 9×9 version is that BSS-2 has not enough neurons to represent the latter. Because every cell would need nine neurons to represent each number between one and nine, the network would require $9 \times 9 \times 9 = 729$ neurons.

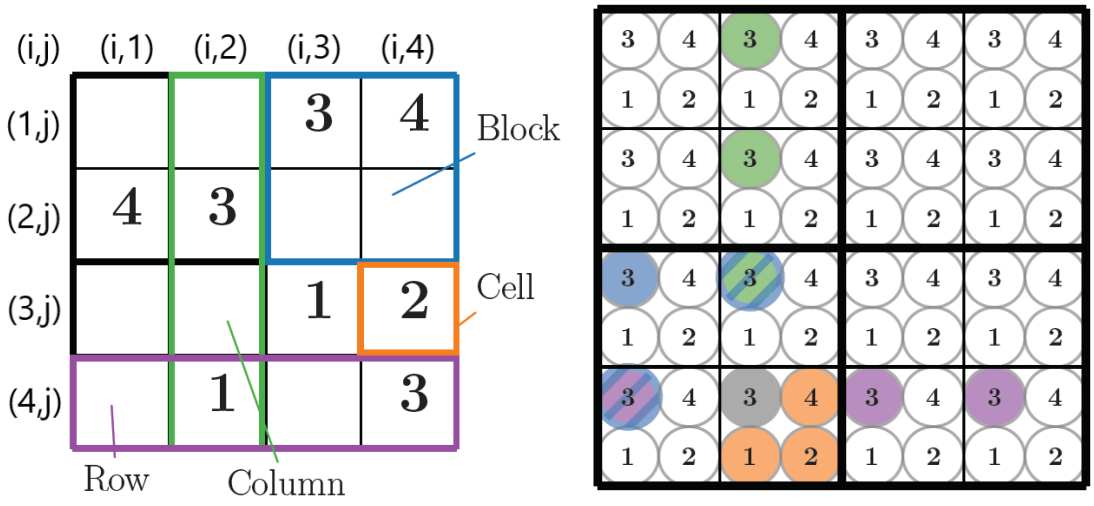

Figure 19: *Left:* The terminology of a 4×4 Sudoku and how the cells are indexed. *Right:* Each cell contains four neurons in order to represent every possible number. The connections between the neurons of the 4×4 Sudoku are exemplary drawn for the gray neuron. The inhibitory synapses are between neurons in the same cell (orange), the same block (blue), the same row (purple) and the same column (green). This is applied to all neurons and amended by a self-excitation for every neuron (figures modified from [25]).

The neurons are configured in a way that an incoming spike directly triggers an output if the potential is not inhibited. With these settings, a stimulated neuron can directly send an inhibitory signal to the remaining numbers of the cell to suppress that they are set as well. Following the same logic, the neurons are also connected to the same numbers of the same row, the same column and the same block. In this way, the neural network can simultaneously check the sudoku rules for all cells. To preserve the firing behavior for neurons that have been stimulated, it is necessary to add excitatory self-connections and a continuous external stimulation. Together they cause a new spike as long as no suppressing input from other neurons is arriving.
To start the experiment, the given numbers (clues) are send as regular input spikes to the respective neuron with a high frequency while all neurons receive a background stimulation with half the frequency after a lag of 0.1 ms. In this way, the membrane potential of the neurons representing forbidden numbers is already pulled down before the external spike source would evoke a spike and theoretically only the allowed numbers are firing. This explains what can be see in figure 20 where only the neurons representing the clues are firing before the vertical blue line. For comparing networks with background noise from a Poisson process to those with regular input, it is helpful

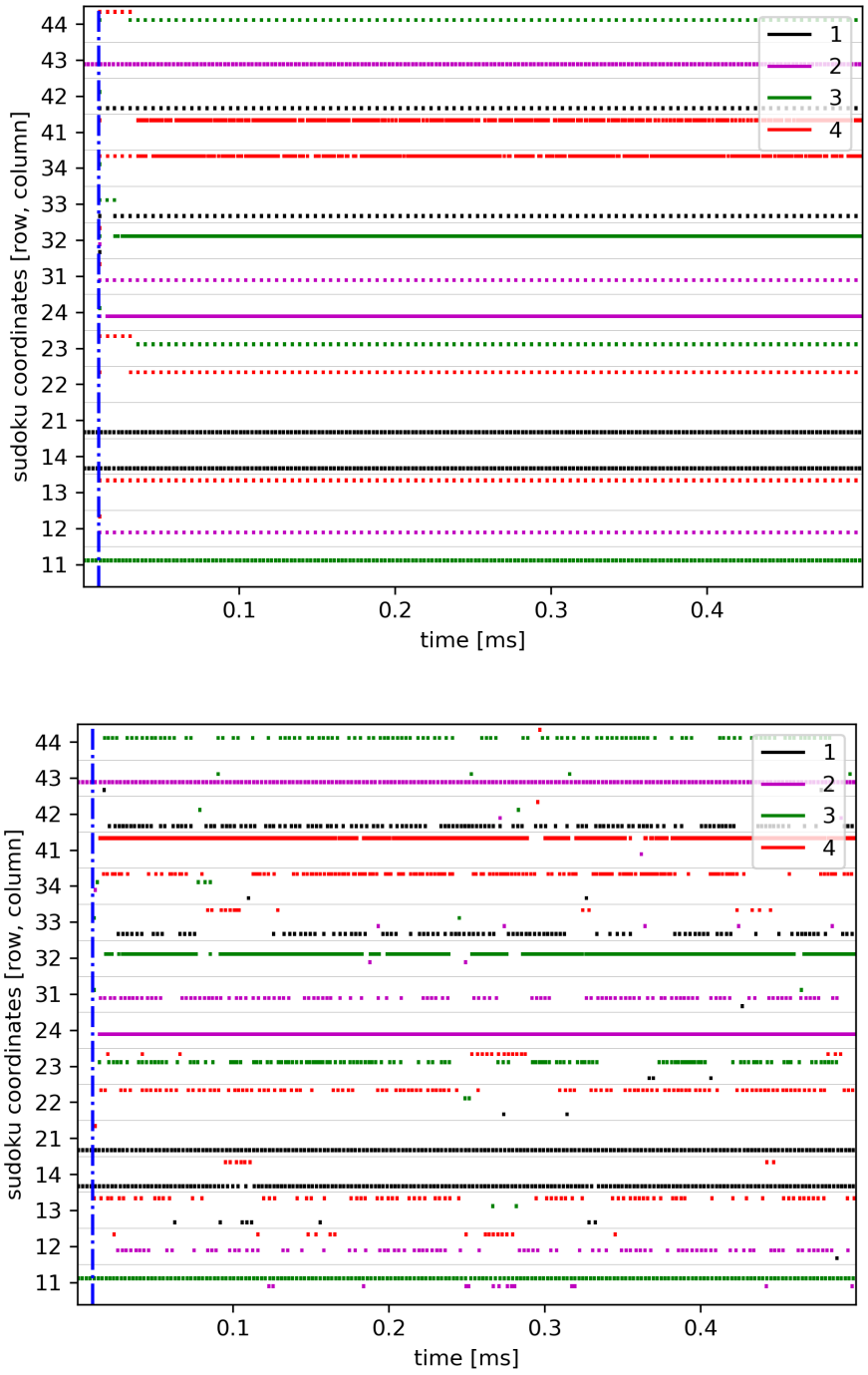

Figure 20: Recorded spike activity for all neurons of a sudoku solver network for a regular *(top)* and a Poisson *(bottom)* background stimulation. The different cells of a sudoku in matrix notation are on the y-axis while the time is on the x-axis. The four neurons of every cell are distinguished by the different colors. The clues are firing before the blue line and the empty cells are correctly solved over time. The background stimulation for all neurons is either constantly spaced in time or produced by a Poisson process.

to take a closer look at the way the spiking pattern develops over time. As soon as the neurons receive a regular input, they are mostly either firing or inhibited for the rest of the runtime. In contrast, the Poisson background allows the cells to switch their associated number over time. To conclude, the Poisson background seems preferable as it allows to correct for imprecise timing in the beginning which makes it more robust and illustrates the idea of a neural network that ideally develops towards the right solution over time.

In order to quantitatively measure the outcome, each cell is associated with the number represented by the neuron having the most spikes. If no neuron of a certain cell fires more than 10 times, the cell would be considered empty. The number for every cell obtained by the neural network is then compared to the solution and the amount of falsely labelled cells is stored. In this way the performance of the two networks is measured for 60 different sudokus [26] and five times each per number of clues. In the beginning, only four clues are given and after every round another clue is added by revealing a random cell that was previously empty. Thereby should be mentioned that less than four clues wouldn't make sense since then the sudoku could always be solved in various ways [27], i.e. the solution would be unambiguous. It has also been tested that the given clues lead to an unique solution.

Although the results (fig. 22) show that the regular background stimulation on average has 0.36 less neurons that are falsely labelled, this difference is not significant since the values are always within the standard deviation of the other network.

Increasing the number of clues until a maximum of 12, yields a clear improvement of the performance for both cases (fig. 21). For four clues in the Poisson case, the average cells that are wrongly solved, lies at 4.2 and shrinks to 0.1 ± 0.4 cells for 12 clues given. While an average of 4.2 false numbers for 12 empty cells means an error rate of 35 %, this number decreases to 3.3 % for the case with four empty cells. For the other network the decline is from 30 % to 1.5 %.

A further analysis can be conducted when looking at the detailed results for the 60 different sudokus for an exemplary number of eight clues (fig. 22). It shows that for both cases the majority of sudokus can be solved correctly for all five runs. Although the average rate is better for a regular background, it has up to six falsely labelled cells compared to a maximum of four for the more random background. Interestingly, there is no shared pattern that can be observed, i.e. it seems like there aren't any sudokus that are particularly hard to solve for both networks.

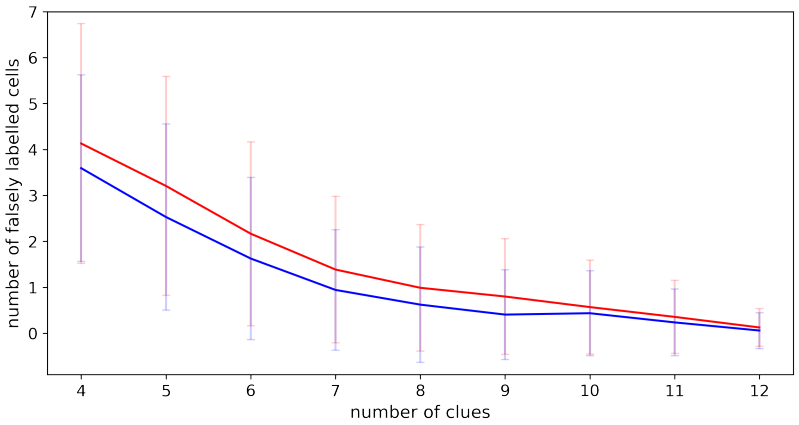

Figure 21: The average number of falsely labelled cells for 60 different sudokus plotted over the number of clues. The neural network with constant background (red) performed slightly better compared to the one with Poisson stimulation (blue). For both versions, the rate of falsely solved cells declines with more clues given to the network.

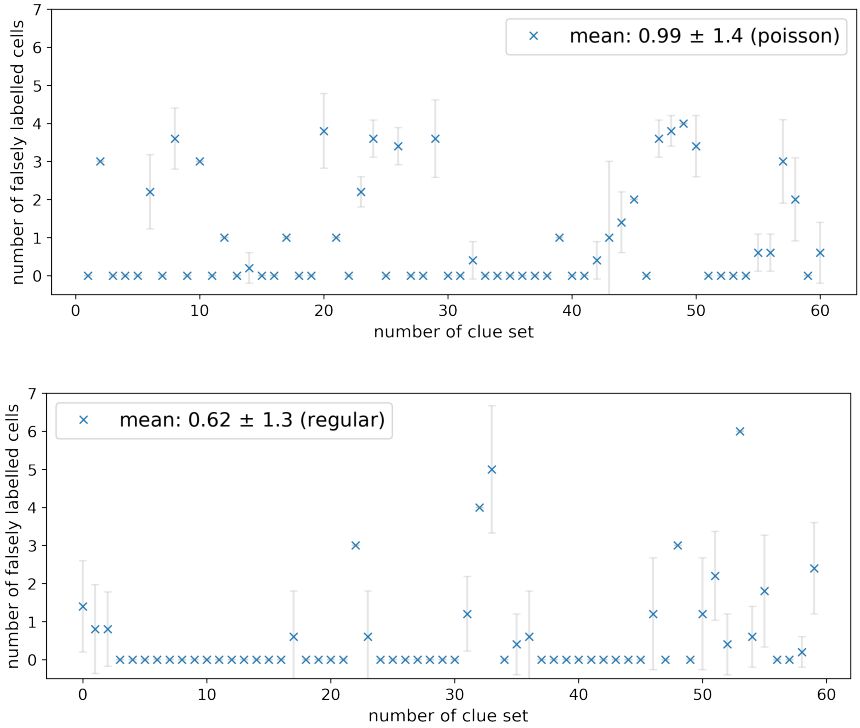

Figure 22: The number of falsely solved cells by a network containing Poisson spaced *(top)* and constantly spaced *(bottom)* background spikes for 60 different sudokus with eight provided clues. For both versions the majority of the sudokus can be correctly solved for five times in a row.

To form a task out of the observations, it is important to consider the comparable small amount of time for the advanced lab course. Therefore, my suggestion contains the following three subtasks if the students have the choice between task 6, task 7 and this task:

1. Draw the inhibitory connections in such a way that neurons which represent the same number suppress each other in the same block, row and column (start with the weight of each synapse set to their maximum). The connections between the neurons representing different numbers of the same cell are given as an example. Test your network for the first set of 12 clues. Verify the error rate on the plot. If the error rate is more than 0%, adapt the neuron parameters and the connections weights.

2. Observe how the error rate changes if you reduce the number of clues. Less than four clues lead to more than one possible solution and therefore don't make sense. Try to improve the error rate when only four clues are given. Redo the sweep over all numbers of clues afterwards.

3. Test how well your network works if you change the sudoku. Therefore, the data for 60 different set of clues is given in the script. Choose a number of clues you want to give for each sudoku, create a loop which measures the error rate for all 60 sudokus and then run the experiment overnight. The results should than be plotted in a histogram with the given script.

Alternatively, the last subtask could be made optional and instead the sixth task could stay mandatory. It could also be a possibility to let the students compare the differences of the regular and the Poisson background stimulation although tuning both networks can be very time consuming.

4 Discussion

The thesis successively ported the the tasks of the advanced lab course to BrainScaleS-2. For each one it was necessary to adapt the code to the PyNN functionality given on the new hardware and implement new parameters or scale them to a wider range. Executing the experiments and finding a suitable working point partly involved the implementation of a calibration and the selection of an adequate number of neurons. Given these steps, it was necessary to evaluate in which way the (sub)tasks could sustain their aim of showing the respective aspect of neural networks.

To start with, the synaptic input given onto a single neuron could be observed through its membrane potential and also be visibly modified. Switching from abstract hardware parameters to more descriptive synaptic properties could be helpful for the intuition of future students doing the lab course. The subtasks dealing with fixed-pattern noise could be equally conducted on the new hardware and the optional comparison with temporal noise put the results into relation. Thereby, a visualization was produced that could be integrated into the scripts given to the students. Stacking input spikes together could be executed with PyNN as intended by the task.

The forth task required a different type of synapse that is able to learn from previous input. Its implementation into PyNN for the new hardware required the handling of routing that was out of scope for my thesis and couldn't yet be taken care of by another group member. Before finally porting the lab course, it is recommended to have it in place since synaptic plasticity is an essential part of learning in the biological sense.

For the realization of the synfire chain, it was necessary to increase the maximum firing rate of the neurons under the given input strength. Two possible ways how this can be done with tuning neuron parameters have been outlined. To obtain the maximum length of the synfire chain, the initial size of the populations has been reduced as far as it was possible with sweeping selected parameters.

The sixth task used the coefficient of variation of the inter-spike intervals for building a network simulating a Poisson process. The expected relation of the firing rate and the CV could be verified on the new hardware. However, it would make sense to plot it in dependency of the ISI and not the firing rate since it is calculated using the former. This could avoid confusion in the future and thereby following the convention in the literature [22]. The already computationally expensive 2D histograms have a more precise resolution on the BrainScaleS-2 hardware and shouldn't be computed in

full resolution by the students to avoid time constrains for the following experiments. Similiar to the previous tasks of the current version of the lab course, the XOR network could be transferred by scaling up the former 4-bit synaptic weight to the now 6-bit resolution. Thereby, making the network better tunable and facilitating a working system, but it remains an interesting model case for the experimenters. As intended, an increasing amount of jitter strongly influenced the classification rate of the output.

Lastly, a new task could be constructed which introduces the students to a network which is able to solve 4×4 sudokus. When working on the existing tasks, several students expressed the wish to not only modify, but also create a network structure. This is taken care of by assigning the students to set up the connections between the neurons of the sudoku solver. It will also provide an alternative to the XOR network for those who prefer a more functional task for a neural network. While designing the task, the different responses of the network to constantly spiking and Poisson distributed input have been observed. Although the error rate was slightly in favor of the former, it has been argued that it is more important that randomness in the input allows the neural network to develop over time. To demonstrate what students ideally obtain for the second subtask, the influence of more clues has been measured not only for one sudoku, but for all 60 sudokus. It could be shown that more clues mean a constant improvement in the performance of both networks. As the parameters have been manually tuned, a further (automated) optimization would lead to a more quantitative evaluation of the two networks. This has been done by Ostrau et al. [28] for the BrainScaleS-1 architecture and showed a strong dependency of the performance on single parameters like the self-excitation. Although the parameters differ for the two hardware versions as for instance the synapses there are conductance-based and not current-based as for BSS-2, a similar influence of the parameters has to be assumed. To conclude, even if a further tuning of the parameters will most likely lead to an improved error rate, the obtained one is sufficient for all observations that are supposed to be made in the frame of the advanced lab course.

5 Outlook

Besides the previously explained aspects, the following ones have to be considered before finally porting the advanced lab course onto BSS-2. First of all, the new hardware necessitates a description and explanation in the lab manual [21]. This could also be accompanied by an introduction of potential applications of neural networks which is currently missing and has been desired by students. Moreover, the new task has to be documented and with it other (sub)tasks have to become optional to remain within the time frame of the lab course.
Pertaining to software, it has been started to automate the loading of a calibration in PyNN and give the option to deactivate it for the examination of fixed-pattern noise or similar tasks. Moreover, it is planned to give users the possibility to not only operate in units of the Digital-to-Analog Converter or a mixture thereof with hardware values. Following the example of PyNN on Spikey, the aim is to have the possibility to also obtain results for the advanced lab course in biological values. This improves the intuition for which result to expect and overall makes it easier to calculate with the parameters during the experiments, but can only be realized when having an elaborate implementation of the underlying calibrations with according transformation models.
Lastly, all software scripts used for the advanced lab course will become part of the continous integration (CI) in order to ensure a running setup for the students. This means automated testing on a daily basis to verify the compability with new code that is developed by the group. Therefore, software problems occurring in the tasks can be located promptly and fixed accordingly.

6 Acknowledgment

The work carried out in this bachelor thesis used systems, which received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 720270, 785907 and 945539 (Human Brain Project, HBP)

7 Appendix

Change ID	Topic
12143/5	Task 1: Investigating a Single Neuron
12424/5	Task 2: Calibrating Neuron Parameters
13116/2	Task 3: A Single Neuron with Synaptic Input
12964/2	Task 5: Feed-Forward Networks (Synfire Chain)
13178/3	Task 6: Recurrent Networks
13649/1	Task 7: A Simple Computation - XOR
13216/2	Extra Task: Sudoku Solver

Table 1: Overview over the Gerrit changes submitted for the thesis.

Code Example (Synfire Chain)

```
# import PyNN interface
import pynn_brainscales.brainscales2 as pynn
from pynn_brainscales.brainscales2.standardmodels.cells import \
    SpikeSourceArray, HXNeuron
from pynn_brainscales.brainscales2.standardmodels.synapses import \
    StaticSynapse

def main(no_pops, pop_sizes, weights, duration, neuron_params,
        closed, calib_path):

    """
    Create a synfire chain.
    :param no_pops: chain length
    :param pop_sizes: number of neurons in the excitatory ('exc')
                and inhibitory ('inh') populations
    :param weights: connections strengths between 'stim_exc',
                'exc_exc', 'exc_inh' and 'inh_exc'
    :param duration: emulation time in ms
    :param neuron_params: parameters for all HXNeurons
    :param closed: indicates if the chain is closed
    :param calib_path: needed for calibration
    :return: excitatory and inhibitory spiketrains
    """
    # load calibration
    [...]
    pynn.setup()

    # create neuron populations
    pop_collector = {'exc': [], 'inh': []}
    for syn_type in ['exc', 'inh']:
        for _ in range(no_pops):
            if calib_path:
                pop = pynn.Population(pop_sizes[syn_type],
                        HXNeuron(hx_coco,**neuron_params))
            else:
                pop = pynn.Population(pop_sizes[syn_type],
                        HXNeuron(**neuron_params))
            pop.record(["spikes"])
            pop_collector[syn_type].append(pop)


```

```
42      # record first neuron of first excitatory population of chain
43      pop1exc = pop_collector['exc'][0]
44      pop1exc[[0]].record('v')
45
46      # kick starter input pulse
47      stim_pop = pynn.Population(pop_sizes['exc'],
48                                 SpikeSourceArray(spike_times=[0]))
49
50      # connect stimulus
51      pynn.Projection(stim_pop, pop_collector['exc'][0],
52              pynn.AllToAllConnector(), synapse_type=
53              StaticSynapse(weight=weights["stim_exc"]),
54              receptor_type='excitatory')
55      pynn.Projection(stim_pop, pop_collector['inh'][0],
56              pynn.AllToAllConnector(), synapse_type=
57              StaticSynapse(weight=weights["stim_exc"]),
58              receptor_type='excitatory')
59
60      # for closing the loop you need to change the for-loop range
61      # i.e. if pop_index < no_pops - 1: open chain
62      if closed:
63          lastiter = no_pops
64      else:
65          lastiter = no_pops - 1
66      for pop_index in range(lastiter):
67          pynn.Projection(pop_collector['exc'][pop_index],
68                  pop_collector['exc'][(pop_index + 1) % no_pops],
69                  pynn.AllToAllConnector(), synapse_type=
70                  StaticSynapse(weight=weights["exc_exc"]),
71                  receptor_type='excitatory')
72          pynn.Projection(pop_collector['exc'][pop_index],
73                  pop_collector['inh'][(pop_index + 1) % no_pops],
74                  pynn.AllToAllConnector(), synapse_type=
75                  StaticSynapse(weight=weights["exc_inh"]),
76                  receptor_type='excitatory')
77          pynn.Projection(pop_collector['inh'][pop_index],
78                  pop_collector['exc'][pop_index],
79                  pynn.AllToAllConnector(), synapse_type=
80                  StaticSynapse(weight=weights["inh_exc"]),
81                  receptor_type='inhibitory')
82
83
```

```
84      # emulate the network
85      pynn.run(duration)
86      # read back all recorded spikes
87      spike_collector = {'exc': np.zeros(no_pops, dtype=object),
88                         'inh': np.zeros(no_pops, dtype=object)}
89      for syn_type in ['exc', 'inh']:
90          for pop_index in range(no_pops):
91              spike_collector[syn_type][pop_index] =\
92                  pop_collector[syn_type][pop_index].\
93                  get_data("spikes").segments[0].spiketrains
94
95      # read back the membrane potential
96      mem_v = pop1exc[[0]].get_data("v").segments[-1].\
97              analogsignals[0].base
98      pynn.end()
99
100     return spike_collector, mem_v
101
102 if __name__ == "__main__":
103
104     # configure the network
105     no_pops = 8  # chain length
106     pop_size = {'exc': 7, 'inh': 7}  # size of each chain link
107     runtime = 0.2  # ms
108     close_chain = True
109     calib_path = pynn.helper.find_calib()
110
111     # define weights in digital hardware values
112     synapse_weights = dict(
113         stim_exc=63,
114         exc_exc=60,
115         exc_inh=20,
116         inh_exc=15)
117
118     # default doesn't yield a small refractory period
119     neuron_params = {"refractory_period_refractory_time": 5}
120
121     # execute the experiment and retrieve all result spikes
122     results, mem_v = main(no_pops, pop_size, synapse_weights,
123                 runtime, neuron_params, close_chain, calib_path)
```

8 References

[1] N. Karayiannis & A. N. Venetsanopoulos. *Artificial neural networks: learning algorithms, performance evaluation, and applications*. Vol. 209. Springer Science & Business Media, 2013. Chap. 9.

[2] D. Ciresan; U. Meier & J. Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3642–3649. DOI: 10.1109/CVPR.2012.6248110.

[3] Y. Xu; J. Du; L.-R. Dai & C.-H. Lee. "An experimental study on speech enhancement based on deep neural networks". In: *IEEE Signal processing letters* 21.1 (2013), pp. 65–68. DOI: 10.1109/LSP.2013.2291240.

[4] S. Schmitt; J. Klähn; G. Bellec; A. Grübl; M. Güttler; A. Hartel; S. Hartmann; D. Husmann; K. Husmann; S. Jeltsch; V. Karasenko; M. Kleider; C. Koke; A. Kononov; C. Mauch; E. Müller; P. Müller; J. Partzsch; M. A. Petrovici; S. Schiefer; S. Scholze; V. Thanasoulis; B. Vogginger; R. Legenstein; W. Maass; C. Mayr; R. Schüffny; J. Schemmel & K. Meier. "Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2227–2234. DOI: 10.1109/IJCNN.2017.7966125.

[5] S. K. Esser; P. A. Merolla; J. V. Arthur; A. S. Cassidy; R. Appuswamy; A. Andreopoulos; D. J. Berg; J. L. McKinstry; T. Melano; D. R. Barch; C. di Nolfo; P. Datta; A. Amir; B. Taba; M. D. Flickner D. S. Modha. "Convolutional networks for fast, energy-efficient neuromorphic computing". In: *PNAS* 113.41 (2016), pp. 11441–11446. DOI: 10.1073/pnas.1604850113.

[6] A. Nock. *Internship Report*. University of Heidelberg, 2020.

[7] W. Gerstner; W. M. Kistler; R. Naud & Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. Chap. 1. DOI: 10.1017/CBO9781107447615.

[8] W. Gerstner & W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. Chap. 4. DOI: 10.1017/CBO9780511815706.

[9] S. L. Bressler & V. Menon. "Large-scale brain networks in cognition: emerging methods and principles". In: *Trends in Cognitive Sciences* 14.6 (2010), pp. 277–290. DOI: https://doi.org/10.1016/j.tics.2010.04.004.

[10] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain ". In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[11] A. L. Hodgkin & A. F. Huxley. *A quantitative description of membrane current and its application to conduction and excitation in nerve*. 117. 1952, pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764.

[12] R. Brette & W. Gerstner. *Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity*. 94. 2005, pp. 3637–3642. DOI: 10.1152/jn.00686.2005.

[13] M. A. Petrovici. *Form Versus Function: Theory and Models for Neuronal Substrates*. Springer Nature, 2016. Chap. 2. DOI: 10.1007/978-3-319-39552-4.

[14] Milena Czierlinski. "PyNN for BrainScaleS-2". Bachelorarbeit. Universität Heidelberg, 2020. Chap. 3. URL: https://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=4114.

[15] J. Schemmel; S. Billaudelle; P. Dauer & Johannes Weis. *Accelerated Analog Neuromorphic Computing*. 2020. arXiv: 2003.11996.

[16] E. Müller; C. Mauch; P. Spilger; O. Julien Breitwieser; J. Klähn; D. Stöckel; T. Wunderlich & J. Schemmel. *Extending BrainScaleS OS for BrainScaleS-2*. 2020. URL: https://arxiv.org/abs/2003.13750.

[17] A. Grübl; S. Billaudelle; B. Cramer; V. Karasenko & J. Schemmel. *Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity*. 92. 2020, pp. 1277–1292. DOI: 10.1007/s11265-020-01558-7.

[18] A. Davison; D. Brüderle; J. Eppler; J. Kremkow; E. Muller; D. Pecevski; L. Perrinet; P. Yger. "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics* 2 (2009), p. 11. DOI: 10.3389/neuro.11.011.2008. URL: https://www.frontiersin.org/article/10.3389/neuro.11.011.2008.

[19] A. P. Davison; E. Müller; S. Schmitt; B. Vogginger; D. Lester & T. Pfeil. *HBP Neuromorphic Computing Platform Guidebook*. URL: https://flagship.kip.uni-heidelberg.de/jss/FileExchange/HBPNeuromorphicComputing/PlatformGuidebook.pdf?fID=1504&s=qqdXDg6HuX3&uID=65. (Chapter 5.2: Spikey School. Accessed: 29.01.2021).

[20] *SP9 Neuromorphic Computing Platform - Results from SGA2 year 1 (D9.6.1 -SGA2)*. URL: https://sos-ch-dk-2.exo.io/public-website-production/filer_public/c8/1a/c81a22a7-5d94-416e-8003-613f37280996/d961_d621_d30_sga2_m13_accepted_190723.pdf. (accessed: 29.01.2021).

[21] A. Grübl & A. Baumbach. *F09/F10 Neuromorphic Computing*. 2006. URL: https://www.physi.uni-heidelberg.de/Einrichtungen/FP/anleitungen/F09.pdf.

[22] F. Gabbiani & S. J. Cox. *Mathematics for Neuroscientists*. Elsevier, 2010. Chap. 15. ISBN: 978-0-12-374882-9.

[23] W. R. Softky and C. Koch. "The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs". In: *Journal of Neuroscience* 13.1 (1993), pp. 334–350. DOI: 10.1523/jneurosci.13-01-00334.1993.

[24] S. Chatterjee; S. Paladhi & R. Chakraborty. "A Comparative Study On The Performance Characteristics Of Sudoku Solving Algorithms". In: *IOSR Journal of Computer Engineering* 16.5 (2014), pp. 69–77. URL: https://www.jstor.org/stable/25678701.

[25] A. Kugele. "Solving the Constraint Satisfaction Problem Sudoku on Neuromorphic Hardware". Master's thesis. Universität Heidelberg, 2018. URL: https://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3666.

[26] *Sudoku Download 4x4*. URL: http://www.sudoku-download.de/sudoku_4x4.php. (courtesy of Alexander Maack, accessed: 15.01.2021).

[27] L. Taalman. "Taking Sudoku Seriously". In: *Math Horizons* 15.1 (2007), pp. 5–9. URL: https://www.jstor.org/stable/25678701.

[28] C. Ostrau; C. Klarhorst; M. Thies & U. Rückert. "Comparing Neuromorphic Systems by Solving Sudoku Problems". In: *2019 International Conference on High Performance Computing & Simulation (HPCS)*. 2019, pp. 521–527. DOI: 10.1109/HPCS48598.2019.9188207.

Erklärung

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

Heidelberg, den 01.02.2021,

Alexander Nock