Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Jan Valentin Straub

born in Heidelberg (Germany)

July 2023

Multi-*Single-Chip* Training of Spiking Neural Networks with BrainScaleS-2

This Bachelor Thesis has been carried out by Jan Valentin Straub at the

Kirchhoff Institute for Physics in Heidelberg

under the supervision of

Dr. Johannes Schemmel

Multi-*Single-Chip* Training of Spiking Neural Networks with BrainScaleS-2

Spiking neural networks (SNNs) on analog neuromorphic hardware are often limited by the properties of the underlying system. Despite being a promising approach to machine learning (ML) tasks with high energy efficiency, this often disables the opportunity to challenge problems of higher complexity due to the necessary network sizes. The idea of partitioning networks into independently executable parts defines a possibility to overcome this problem. An algorithm on how to find suitable partitions for feedforward networks given a set of limitations is developed and implemented within this thesis. Additionally, methods are designed to deal with further limitations of the neuromorphic hardware system BrainScaleS-2 (BSS-2) when approaching the training of SNNs with back propagation through time (BPTT) and surrogate gradient methods. The scalability of BSS-2 is shown by demonstrating how a larger scale network can be trained with the MNIST data set. For a network topology of 22×22 $\rightarrow$ 256 leaky integrate-and-fire (LIF) neurons $\rightarrow$ 10 leaky integrator (LI) neurons, the simulations reached accuracies of $98.42\% \pm 0.06\%$, the hardware in the loop training reached 97.22%. This thesis builds the basis for further development and research regarding larger scale networks on limited hardware, e.g. optimization with multi chip setups or other network structures such as convolutions.

Multi-*Single-Chip* Training von spikenden neuronalen Netzwerken mit BrainScaleS-2

Spikende neuronale Netzwerke (SNNs) auf analoger neuromorpher Hardware sind oft durch die Eigenschaften des verwendeten Systems begrenzt. Obwohl sie einen vielversprechenden Ansatz darstellen, Machine-Learning-Aufgaben mit hoher Enerieeffizienz zu bearbeiten, erlaubt diese Begrenzheit oft nicht, Aufgaben höherer Komplexität anzugehen. Die Idee, Netzwerke in voneinander unabhängige Teile zu partitionieren, bietet eine Möglichkeit, dieses Problem zu umgehen. In dieser Arbeit wird ein Algorithmus entwickelt und implementiert, der für feedforward-Netzwerke passende Partitionierungen bereitstellt. Darüber hinaus werden Methoden entwickelt, um mit weiteren Einschränkungen des neuromorphen Hardwaresystems BSS-2 beim Trainieren von SNNs mit BPTT und Surrogate-Gradient-Methoden umzugehen. Die Skalierbarkeit des BSS-2-Systems wird anhand des Trainings eines Netzwerks mit größerer Topologie auf dem MNIST Datensatz gezeigt. Mit einer Netzwerk-Topologie von 22×22 $\rightarrow$ 256 LIF Neuonen $\rightarrow$ 10 LI Neuronen erreichen die Simulationen Genauigkeiten von $98.42\% \pm 0.06\%$, das Hardware-Training erreichte 97.22%. Diese Arbeit legt damit den Grundstein für Weiterentwicklungen in diesem Bereich, die sich mit der Optimierung durch multi-Chip-Systeme oder anderen Netzwerkstrukturen wie Convolutions beschäftigen könnten.

Contents

1 Introduction 1
1.1 Thesis Outline . . . 1

2 Theoretical Background 2
2.1 Biological Neurons . . . 2
2.2 The LIF model . . . 2
2.2.1 Numerical Solution . . . 4
2.3 Encoding and Decoding for Spiking Neural Networks . . . 6
2.3.1 Input Encoding . . . 6
2.3.2 Decoding . . . 7
2.4 Gradient Based Learning . . . 7
2.4.1 Loss functions . . . 8
2.4.2 Backpropagation Through Time . . . 8
2.4.3 Surrogate Gradients . . . 9
2.4.4 PyTorch . . . 9
2.5 MNIST . . . 10

3 The BrainScaleS-2 System 11
3.1 In-The-Loop Learning with `hxtorch.snn` . . . 12

4 Partitioning 13
4.1 Partitioning for Feed Forward fully connected architectures . . . 13
4.1.1 Exemplified on MNIST . . . 14

5 Training on BSS-2 16
5.1 Methods . . . 16
5.1.1 Data Set Transformations . . . 16
5.1.2 Time-to-First-Spike Encoding . . . 16
5.2 Network Topology . . . 17
5.3 Simulations and Mapping to Hardware . . . 18
5.4 Hardware behaviour . . . 20
5.4.1 A Basic Experiment . . . 20
5.4.2 Investigating postsynaptic potential (PSP)-Heights . . . 20
5.4.3 Saturation Issues and Resolving Methods . . . 23
5.5 Simulation Results . . . 25
5.6 Multi Single-Chip Execution on BSS-2 . . . 27
5.6.1 Final Results . . . 28

6 Discussion and Outlook 31

References and Sources 33

Acknowledgements 35

A Acronyms 36

B Software State **37**

C Declaration/Erklärung **38**

1 Introduction

Inspired by the capabilities of the human brain, the field of ML has developed artificial neural networks (ANNs) and SNNs in an effort to achieve proficiency in a variety of complex tasks that the human brain has proven to be very capable of. Image and pattern recognition as well as language processing are just a few topics of modern ML-research. With the applications of artificial intelligence reaching far beyond the mentioned examples and tasks getting increasingly complex, optimizations in many ways are of great interest to the research progress. SNNs, particularly in combination with neuromorphic hardware [9], have proven themselves to be an energy efficient method to solve ML-tasks due to their sparse event-driven information flow. However, modern neuromorphic computing platforms such as the BSS-2 system [15, 19], developed by Electronic Vision(s) in Heidelberg, are limited in the resources of a chip, and thus prevent from addressing complex tasks that involve larger scale networks. With the training of SNNs fitting on one BSS-2-chip already shown [19], this thesis introduces multi single-chip learning with the BSS-2 system, allowing for the training of larger scale networks by partitioning. Therefore, a partitioning algorithm for feedforward network topologies was developed and implemented with the software framework `hxtorch.snn` [20]. Furthermore, this algorithm was integrated in setting up network structures. Including additional regularization methods for training with neuromorphic hardware, this is demonstrated with the example of the MNIST data set [1] for which simulations reach accuracies of $98.42\% \pm 0.06\%$ and hardware trained models 97.22%.

1.1 Thesis Outline

Chapter 2 gives an overview on the workwise of SNNs. The LIF neuron model and numerical solutions to the dynamical equations are covered as well as which steps are necessary when training SNNs.
In Chapter 3, the neuromorphic hardware system BSS-2 is introduced with the necessary feeatures that are used in this work.
In Chapter 4, I derive how partitioning enables handling larger scale feedforward networks and present an algorithm that determines the necessary partitions for given network structures.
Chapter 5 covers the training process on BSS-2 using the MNIST data set. A set of methods that were involved in the process are presented and after two basic experiments to investigate the hardware, the results of models trained in simulation and with the hardware are shown and inspected.
Chapter 6 concludes the work by discussing the results, contemplating further improvements and giving an outlook on research topics that might follow.

2 Theoretical Background

Before addressing how partitioning of networks can be accomplished and how training on BSS-2 can be approached, this chapter covers the fundamentals of SNNs. Starting with most basic component of a neural network, the neuron, this chapter introduces to learning in SNNs and walks through the necessary steps.

2.1 Biological Neurons

The main components of a neuron cell are its body (soma), its dendrites and its axon. The left image of figure 1 shows the general structure of a typical neuron. Via its dendrites, a neuron can receive electrical signals from other neurons causing a change of the neurons membrane voltage: With the membrane being impermeable to ions and polar molecules, the transport of which are only possible with particular proteins, the membrane voltage is defined as the voltage across this membrane. Without inputs from other neurons, the membrane voltage approaches a resting state of around $-70\,\mathrm{mV}$. However, when it reaches a certain threshold ϑ due to incoming signals, an action potential ('*spike*') is triggered: A phase of depolarization is entered where the membrane voltage rises quickly followed by a repolarization or even hyperpolarization (figure 1, right). In the latter case the neuron enters a refractory period where it is unlikely to spike again. Spikes are transmitted along the axon and can be passed on

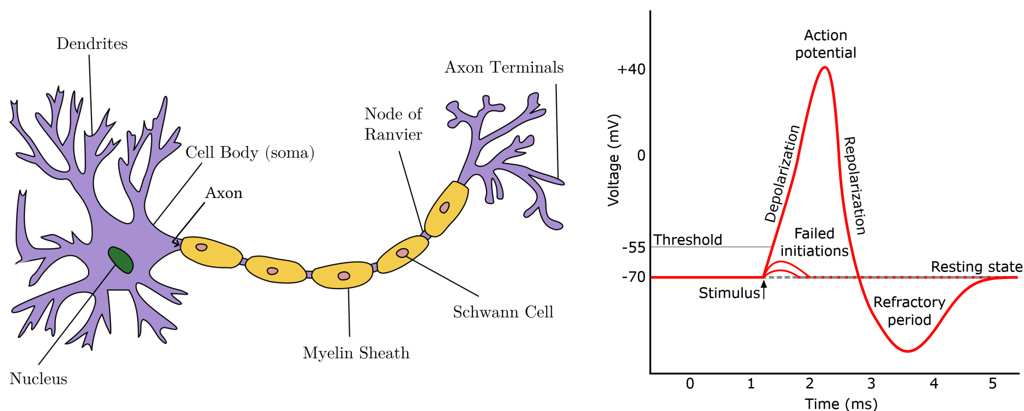

Figure 1: Left: Sketch of a biological neuron. Image taken from [5]. Right: Action potential of a neuron, image taken from [3].

to other neurons via the axon terminals. The connection between axon terminals and the dendrites of other neurons are called synapses, the *synaptic weight* determining the impact of an incoming signal on the membrane voltage of the *postsynaptic* neuron, which is referred to as PSP.

2.2 The LIF model

The foundation of modern machine learning with ANNs and SNNs are neuron models, captured mathematically. The relevant model for this work is the LIF model, a

spiking neuron model that describes the dynamics of the membrane voltage $V_m(t)$ over time in a simplified manner [2]. The idea is that a biological neuron can be modeled with a capacitor with capacitance C_m. Incoming signals can then be represented by a synaptic input current $I(t)$. When the membrane voltage reaches a threshold ϑ, the membrane voltage is reset to a value V_r and a spike is emitted. The output spikes can be described by a so called *spike train* $z(t)$:

$$z(t) = \sum_k \delta(t - t_k), \tag{1}$$

using the Dirac delta distribution δ in units of 1/time. $V_m(t)$ will also approach a leak potential V_l if there is no input signal for a period of time. With the membrane conductance g_m, these ideas are united in the following differential equation:

$$C_m \frac{\mathrm{d}V_m(t)}{\mathrm{d}t} = I(t) - g_m \left(V_m(t) - V_l\right) - C_m z(t) \left(\vartheta - V_r\right) \tag{2}$$

$$\Leftrightarrow \quad \frac{\mathrm{d}V_m(t)}{\mathrm{d}t} = \frac{1}{\tau_{mem}} \left(\frac{I(t)}{g_m} - (V_m(t) - V_l) \right) - z(t)(\vartheta - V_r), \tag{3}$$

where in the second step, we introduced the quantity $\tau_{mem} = C_m/g_m$ which is a measure for the time scale with which the voltage decays and is therefore known as the membrane time constant.

The synaptic input current $I(t)$ usually depends on presynaptic spike trains, meaning the outputs of neurons that are connected to the neuron in question via a synapse. The shape of $I(t)$ is a property of the synapse and can be modeled via a convolution of a kernel ϵ with the respective spike train [16]. On top of that, the synaptic strength is implemented by a scaling factor w, which in the context of neural networks is referred to as a weight. The contribution of the output of a neuron j to the synaptic current of a neuron i then is:

$$I_{ji}(t) = \sum_k w_{ji}(\epsilon * z_j(t)). \tag{4}$$

While for later applications, the units of most quantities are dropped, it can be mentioned that one can think of the kernel or the weights as having the unit of a charge to prevent unit inconsistencies. The most commonly used kernel is a single-exponential kernel (see [16])

$$\epsilon_{single}(t) = A\Theta(t)e^{-t/\tau_{syn}}, \tag{5}$$

with an arbitrary scaling factor $A \in \mathbb{R}_+$, the Heaviside step-function Θ and the

synaptic time constant τ_{syn}. With this kernel, we get

$$I_{ji}(t) = \sum_k w_{ji}(\epsilon_{single} * z_j(t)) = w_{ij} \int_{-\infty}^{\infty} \epsilon_{single}(t') z_j(t-t') \, \mathrm{d}t' \tag{6}$$

$$= A w_{ij} \int_{-\infty}^{\infty} \Theta(t') e^{-t'/\tau_{syn}} \sum_k \delta((t-t') - t_k) \, \mathrm{d}t' \tag{7}$$

$$= A w_{ij} \sum_k \Theta(t - t_k) e^{-(t-t_k)/\tau_{syn}}. \tag{8}$$

With the choice of $A = 1$, the total input current $I_i(t)$ at neuron i with presynaptic neurons j is therefore:

$$I_i(t) = \sum_j \sum_{k_j} w_{ji} \Theta(t - t_{k_j}) e^{-(t-t_{k_j})/\tau_{syn}}. \tag{9}$$

As will be apparent in the following section, it is also useful to write this as a differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} I_i(t) = -\frac{1}{\tau_{syn}} I_i(t) + \sum_j \sum_{k_j} w_{ji} \delta(t - t_{k_j}). \tag{10}$$

When discussing the LIF model, also the non spiking version, the *LI* should be mentioned. For this type of neuron, there is no threshold and therefore no spiking mechanism. Incoming signals are accumulated linearly and the membrane voltage converges back to the leaking potential when there are no current signals. This can be achieved by dropping the term $z(t)(\vartheta - V_r)$ in the differential equation 3.

2.2.1 Numerical Solution

Often, a numerical solution of the dynamics described by differential equations is very helpful, as it allows an investigation of the behaviour without deriving analytical solutions. When it comes to LIF neurons, numerical solutions allow an easy and computationally efficient implementation of the dynamics. The training that has been done in this work depends on a numerical and therefore discrete integration of the differential equations. With $n \in \mathbb{N}$ representing the n-th time step defined by $t = n\Delta t$, Δt being the step size with which time is being discretized, a suitable approximation is given by ([17]):

$$I_i[n+1] = \alpha I_i[n] + \sum_j w_{ij} Z_j[n], \quad \text{and} \tag{11}$$

$$V_i[n+1] = \begin{cases} V_i[n] + \frac{1}{g_m} I_i[n+1] & \text{if } V_i[n] + \frac{1}{g_m} I_i[n] \leq \vartheta \\ V_r & \text{else} \end{cases} \tag{12}$$

with $\alpha \equiv \exp\left(-\frac{\Delta t}{\tau_{syn}}\right)$, $\beta \equiv \exp\left(-\frac{\Delta t}{\tau_{mem}}\right)$, $Z_j[n] \equiv \sum_{k_j} \delta_{nk_j}$ (using the Kronecker-δ and the spike times now being $k_j \Delta t$). The use of exponential Rosenbrock integrators

[4] leaves the exponential form of α and β. Alternatively, Euler integration can be applied in which case α and β happen to become first order approximations for their previously defined values:

$$\exp\left(-\frac{\Delta t}{\tau_{syn/mem}}\right) \approx 1 - \frac{\Delta t}{\tau_{syn/mem}}. \tag{13}$$

Figure 2 shows how the different quantities discussed above create the neuron dynamics.

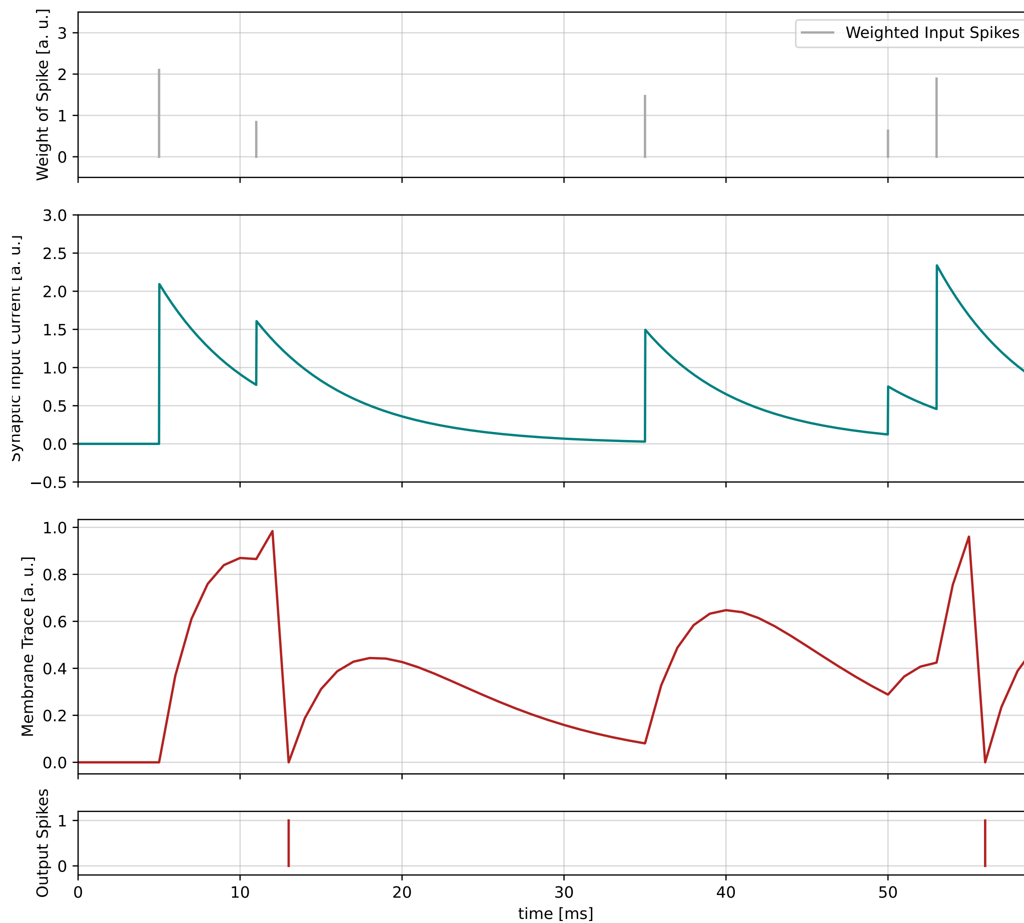

Figure 2: Simulation of the LIF-dynamics, including from top to bottom: The input spike train, the length of the dashes representing the synaptic weight, the resulting synaptic input current, the membrane trace and the spikes generated by this neuron.

2.3 Encoding and Decoding for Spiking Neural Networks

SNNs make use of spiking neuron models such as the LIF-model by connecting them in network structures via synapses. Figure 3 shows how a simple SNN can look like. For simple feed-forward dense network structures typically a distinction in nomination between input layer, output layer and hidden layers is made. When working with

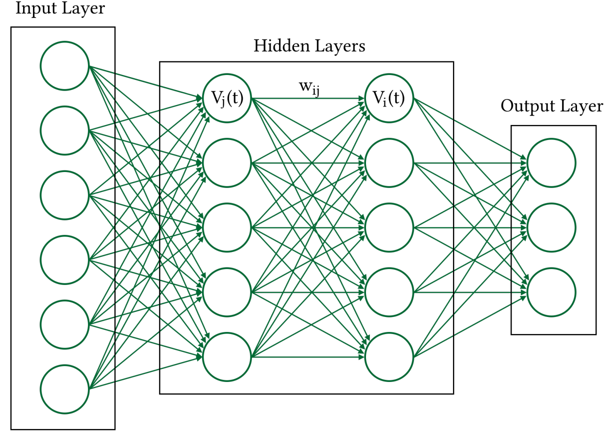

Figure 3: Sketch of a fully connected feed forward type network architecture for SNNs. The circles represent (artificial) spiking neurons, the arrows the connections and synapses with their synaptic weights between them.

SNNs a number of considerations have to be done regarding input encoding, output decoding and the learning process.

2.3.1 Input Encoding

As SNNs cannot make use of real valued inputs, some kind of encoding is needed, resulting in a spiking representation of the input data. The approach used in this work encodes each value into spikes over a sequence length $T \in \mathbb{N}$, corresponding to a time interval t_{enc}. On the neuromorphic hardware system BSS-2 we will use $t_{enc} = T \cdot \mu$s with $T = 30$. With the use of a discrete time grid with an integration time step of 1 μs, the sequence length T will directly correspond to the amount of spike times that are possible within the time span t_{enc}. A spike is then represented by the value 1, and there will be a 0 at times with no spikes.

Constant Current Encoding. With constant current encoding, the input value is interpreted as a bias current that is directed to the membrane of for example a LIF neuron, which will, depending on the bias, start firing in regular intervals, where the firing rate will depend on the current.
This encoding type depends heavily on the scaling of the input, meaning the range of its values and is therefore not invariant with respect to translations and scalings of the input values.

An encoding method that is independent on this is a time to first spike encoding [11]. The general idea is to encode a value over a given time span by the time of a spike. This method will be used throughout all of the trainings and an implementation is given in section 5.1.2.

2.3.2 Decoding

Decoding is heavily dependent on the type of task and can often be adressed individually. For classification tasks that are worked with in this work, the decoding depends on the type of neuron model that is used in the output layer. For our models, we chose LIs for each of the output classes and used a max-over-time decoding, meaning the output neuron with with the highest peak in the output trace of the membrane voltage will be interpreted as the networks guess. The link to one of the output classes enables a final prediction. Via the softmax function, the maximum values of the output traces are mapped to $(0, 1)$ which can then be interpreted as probabilities for each class. When applying the log-function on top, we get the LogSoftmax:
Let $x = \{x_i \mid i \in 1, 2, ..., N\}, x_i \in \mathbb{R}$ be the outcomes of a network regarding a classification problem with $N \in \mathbb{N}$ classes. The softmax and log-softmax are then defined as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{xji}} \quad \text{and} \tag{14}$$

$$\text{LogSoftmax}(x_i) = \log\left(\frac{e^{x_i}}{\sum_{j=1}^{N} e^{xji}}\right). \tag{15}$$

2.4 Gradient Based Learning

Gradient based learning is a standard training method for ANNs as well as SNNs. The general idea is to evaluate the networks output with a loss function $\mathcal{L}(x, \mathbf{W})$, which is dependent on the input x and all the parameters of the network, represented by the parameter vector $\mathbf{W}$. In the usual case for ANNs, only differentiable functions are applied to the input while computing the output of the network. This means that the loss $\mathcal{L}(x, \mathbf{W})|_{x=x'}$ for a given input x' can be derived with respect to any parameter making use of the chain rule. The procedure of retracing the influence of a parameter on the loss with this derivation is called backpropagation. With this method, a gradient descent can be applied to the parameters in an effort to enhance the accuracy of the networks' predictions:

$$\mathbf{W}^{(n+1)} = \mathbf{W}^{(n)} - \kappa \sum_i \left.\frac{\partial \mathcal{L}(x, \mathbf{W})}{\partial \mathbf{W}_i}\right|_{\mathbf{W}=\mathbf{W}^{(n)}, x=x'} \mathbf{e}_i, \tag{16}$$

with the learning rate κ and $\mathbf{e}_i$ being the unit vectors for each parameter in parameter space. Instead of updating the parameters after each input, it is common practice to create input batches containing several samples and update the networks parameters after each batch. One might also consider decreasing the learning rate during the training progress to take smaller steps towards a local (or global) minimum.

2.4.1 Loss functions

While there are a lot of loss functions used in machine learning for a variety of tasks, the most commonly used loss for classification tasks is a cross entropy loss. As we will focus on a specific classification task, learning the MNIST data set, this is one part of the loss functions that will be used. An equivalent implementation of the cross entropy loss is the application of the log-softmax function followed by the negative log likelihood (NLL) loss function. In the configuration that will be used here, the NLL loss is defined as (see PyTorch documentation)

$$l(x, y) = \sum_{i=1}^{N} \frac{l_i}{\sum_{j=1}^{N} w_{y_j}} \quad \text{with} \tag{17}$$

$$l_i = -w_{y_i} x_{i,y_i}, \tag{18}$$

where x as defined before is the output of the network in form of log-probabilities for the classes y. This formulation also involves the possibility to weight the classes differently with the factors w_{y_j}. This can be useful when the training data is unbalanced, meaning different classes are represented unequally often. Ignoring this could lead to the network ignoring minor classes and instead 'focusing' on the bigger classes.

Regularization. SNNs, particularly in combination with neuromorphic hardware, often have limited resources and also the minimization of energy consumption is part of the motivation behind approaching machine learning with SNNs. To ensure some of these properties, for example the mean firing rate of neurons in the network, regularizations of some kind are common. Next to the firing rate, also the weights and the membrane traces are of particular interest when working with e.g. the BSS-2 system (see section 3). With regards to energy consumption but also saturation issues, the latter of which will be discussed in further detail in 5.4.3, the mentioned properties should be contained within certain ranges. One option for regularization is to use additional terms in the loss function that increase when the respective observable approaches higher values or limits. In case of the firing rate, one might use a mean squared error (MSE) loss and add it to the loss function. Regularization can easily become subject to tuning of SNNs when expanding the loss function this way. The reason being that there might be a tread off between the accuracy of the network and its properties as the additional terms in the loss function ultimately influence the gradient.

2.4.2 Backpropagation Through Time

While gradient descent for feed forward ANNs can directly be performed with the methods discussed above, altered approaches have to be used when training SNNs. The two main differences to be addressed are the time dependency of spiking neurons and the non differentiable spiking behaviour. The latter will be discussed in the following section 2.4.3.

The first step is to view SNNs as recurrent neural networks (RNNs). RNNs are neural networks where connections between neurons are not only from one layer to the next but also to any layer before, for example allowing neurons to influence themselves. When revisiting the numerical solutions to the LIF dynamics (equation 12) it is clear the mapping from SNNs to RNNs is possible since the state of a neuron (i.e. its membrane voltage) is dependent on input from previous neurons and the current state which can be viewed as a recurrent connection. The network is then rolled out in time: For each time step of the execution, a copy of the network is created. When layed out next to each other, each time step influences a number of neurons in the following, creating a graph that only has connections in positive time direction, overcoming the recurrent view of the network. With all the copies sharing their parameters, the gradient of the loss function with respect to its parameters can be computed, following the different paths through time that lead to a copy of the parameter. This algorithm is commonly known as backpropagation through time and will be the training algorithm for the example proposed in chapter 5.

2.4.3 Surrogate Gradients

The other issue to training SNNs with gradient based methods is the not differentiable binary spiking mechanism desribed by the Heaviside Θ-Function. A solution to this are surrogate gradients. In that case, when computing the gradient, the non differentiable Θ-Function is replaced with a differentiable function smoothing the step. A collection of these surrogate gradients are shown in figure 4 The idea of surrogate gradients

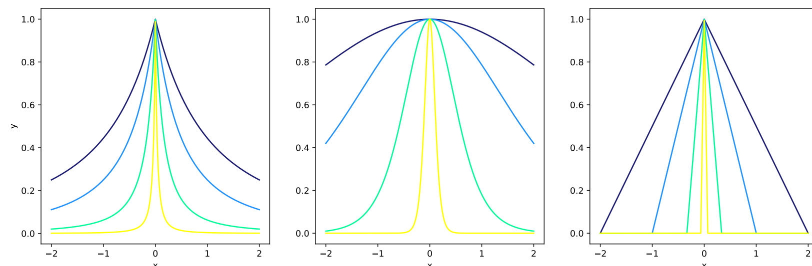

Figure 4: Plots of different surrogate gradients for the heaviside step function and different values of α. Left: Superspike (fast sigmoid derivative) $(1 + \alpha|x|)^{-2}$. Middle: Rescaled sigmoid derivative $4 \cdot \frac{\exp(-\alpha x)}{(1+\exp(-\alpha x))^2}$. Right: Piecewise linear function with slope $\pm\alpha$.

can not only be applied to the LIF dynamics but also any other situation where effectively step functions are applied.

2.4.4 PyTorch

PyTorch [12] is a deep learning research platform for python that provides a broad library of tools. Next to its tensor library torch, we will make use of torch.autograd,

an automatic differentiation library, and torch.nn, a neural network library that is integrated with torch.autograd.

2.5 MNIST

MNIST (modified National Institute of Standards and Technology database) [1] is a data set containing 70000 28×28 gray scale images of handwritten numbers (0 to 9). 60000 of these are intended for training purposes, the rest for testing. The MNIST data set stands as a former benchmark for machine learning models and will provide the data used for training SNNs in this work.

3 The BrainScaleS-2 System

The neuromorphic computing platfrom BSS-2 [15, 19] allows for the emulation of spiking neural networks through electrical circuits. It provides 512 AdEx (adaptive exponential) neuron circuits which can be modified to resemble the behaviour of LIF or LI neurons. Figure 5 shows where the neuron circuits are distributed over the two hemispheres of the chip. Each hemisphere contains two synapse arrays with 128 columns and 256 rows. Upon an incoming event, an exponentially decaying current is triggered in the respective synapse. The amplitudes of these currents are proportional to adjustable weights, that can be configured within the range of 6 bit. With a row-wise setting whether the incoming signals are inhibitory (amplitude < 0) or excitatory (amplitude ≥ 0), signed weights can be accomplished by using two rows per input unit, allowing weights in the range of $w \in \mathbb{Z}, |w| \leq 63$. The sum of all currents triggered in the same column will then be directed to the membrane of one neuron. With the standard configuration of signed weights, this allows for 128 independent input streams to each neuron. On BSS-2 several neuron circuits can be connected to form a large neuron, meaning a group of adjacent neurons that are connected so that they effectively share their membrane voltage, while the spiking mechanism is only enabled for one of them. The number number of distinct synaptic inputs can thereby be scaled up to $c \cdot 128$, $c \in \{1,...,64\}$, c being the number of neurons. The system also features a set of columnar analogue to digital converters

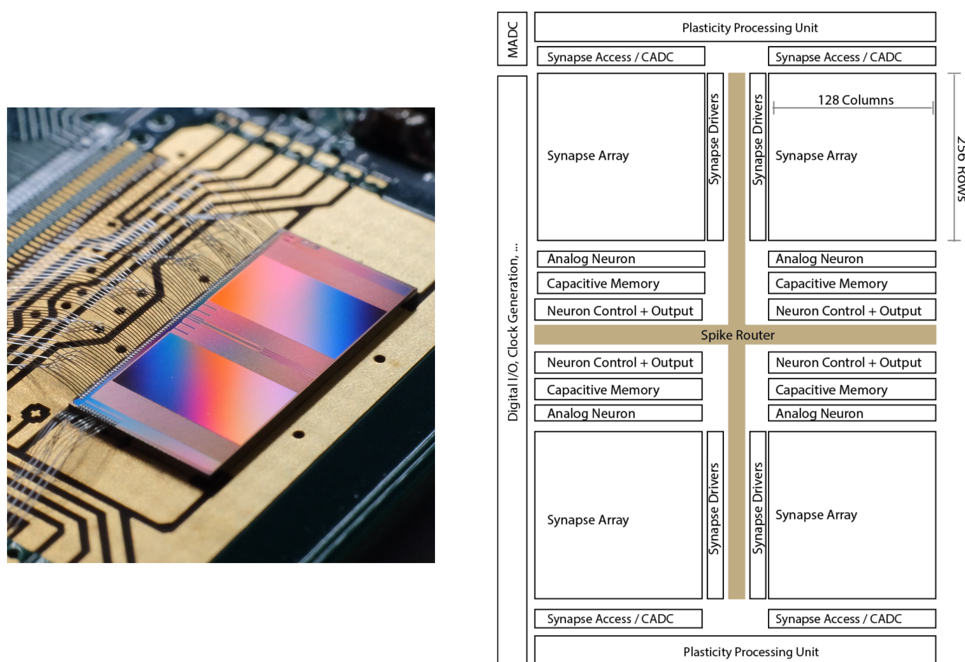

Figure 5: Left: Image of a BSS-2s chip. Right: Floorplan of the chip. Images taken from [19]

(CADCs), allowing the readout of the membrane voltages by the host computer.

The execution of specific network topologies is enabled through external configuration of input spikes and the internal routing on the chip. Neuron parameters such as time constants, leak, reset and threshold values are subject to calibration. With `calix`, a calibration framework for BSS-2, custom calibrations can be realized.

3.1 In-The-Loop Learning with `hxtorch.snn`

As a modeling framework for the BSS-2 system, `hxtorch.snn` [20] was used throughout this thesis. It provides a high level access to the system and extends PyTorch [13] to SNNs. `hxtorch.snn` implements the numerical solutions for the LIF dynamics discussed in section 2.2.1, also making use of the mentioned first order approximation for the exponential term. It also allows for an easy switch between a simulation of the network in software and a hardware execution on the BSS-2 system building a basis for easy comparisons between simulation training and hardware in-the-loop (ITL) training. For the latter, the actual membrane voltages are used in the forward path of execution and injected in the backward path for the computation of the gradient. The gradient is directed to the parameters via the simulated membrane traces and thus also building a connection between simulated and real dynamics. A feature that we will make great use of is that `hxtorch.snn` allows the handling of the hardware execution to a network built in the software. Each network topology is assigned an *experiment* that can be run at any desired time during the execution of the code. This is especially useful when working with parts of networks:

4 Partitioning

When building spiking neural networks with neuromorphic hardware such as the BSS-2 system, it is not trivial how to handle network topologies that exceed the resources of one setup. While the Electronic Vision(s) group is also working on multi-chip-setups, this work focuses on cases where only one chip is available. The following ideas, however, can be applied to multi-chip-setups as well.

In the case of the BSS-2 system, the constraints that are important for this consideration are the number of inputs an atomic neuron can receive ($\iota := 128$), and the number of atomic neurons on the chip ($N_a := 512$). This raises a problem when wanting to execute networks that exceed these numbers. As mentioned in section 3, atomic neurons can be connected to form neuron compartments, effectively sharing their membrane and expanding the number of possible inputs. These constraints can be bypassed by partitioning the respective network in the manner that the resulting parts of the network comply with the constraints for one hardware execution on the chip.

4.1 Partitioning for Feed Forward fully connected architectures

In the following, we will assume the network topology of a feed forward and fully connected network architecture, meaning a network with an input layer, a number of hidden layers and an output layer, where each neuron in each layer has a synaptic connection to all neurons of the following layer. For a total of L layers, let l_i, $i \in \{1, ..., L\}$ be the number of neurons in layer i.

The partitioning algorithm that was developed during this bachelor thesis loops over the processing layers of the network and determines, given the input size of the previous layer, how many atomic neurons c in a larger neuron are necessary, thus how many neuron compartments ($:= N_c$) of this size c can fit on one chip and how many ($=: p$) partitions are necessary when transmitting the information from layer $i-1$ to layer i. In each step of the loop, the following calculation is performed. With $2 \leq i \leq L$:

$$c = \left\lceil \frac{l_{i-1}}{\iota} \right\rceil, \; N_c = \left\lceil \frac{N_a}{c} \right\rceil \quad \Rightarrow \quad p = \left\lceil \frac{l_i}{N_c} \right\rceil. \tag{19}$$

With this result, the number p of partitions and neuron sizes c for each layer, a number of larger scale networks can be trained and executed on the BSS-2 system.

For the execution of a partitioned network, a few more things have to be considered. Since for any part of a partition, the full information of one layer is passed on to disjunct parts of the following, the executions of these parts are independent of each other. Regarding the execution of next layer however, the combined information of all executions from the previous are needed, which is why these have to be concatenated before moving on to the next layer.

Limitations. Although partitioning increases the range of executable networks on BSS-2, it is still limited by the number of neurons on each chip. That is because neuron sizes can not exceed the number of neurons on the chip. Also with increasing numbers of neurons in a larger neuron there may be other issues that remain to be investigated provided the software implementation supports these neuron sizes. With the currently implemented methods, the capability of the chip includes a fan-in of $64 \cdot \iota = 64 \cdot 128 = 8192 = 2^{13}$. For fully connected/dense network topologies this of course also limits the number of units per layer to the same number.

-> image of general concept of partitioning

Implementation. This algorithm was implemented using `hxtorch.snn`. In addition to that, also a network initialization given a partition was implemented in an automated fashion, allowing easy high level changes to the network structure.

4.1.1 Exemplified on MNIST

One of the network topologies that were used for the MNIST data set is using a hidden layer of size 256. Together with an input image size of $22 \times 22 = 484$ and the output layer containing 10 units for the ten classes, the algorithm suggests the following: Starting with the hidden layer, the number of needed neurons in a larger neuron is

$$n_c = \left\lceil \frac{484}{128} \right\rceil = \lceil 3.78125 \rceil = 4 \tag{20}$$

which means that in one hardware execution

$$\left\lceil \frac{512}{4} \right\rceil = 128 \tag{21}$$

compartments of the size 4 can be used, resulting in a total of $\lceil 256/128 \rceil = 2$ necessary hardware executions to pass the input information through the complete hidden layer. Following the same calculation for the output layer, we get:

$$n_c = \left\lceil \frac{256}{128} \right\rceil = 2, \; n_l = \left\lceil \frac{512}{2} \right\rceil = 256 \quad \Rightarrow \quad n_p = \left\lceil \frac{10}{256} \right\rceil = 1, \tag{22}$$

meaning the output can be determined in one extra hardware execution and a total of $P = 3$ are necessary for the entire network.

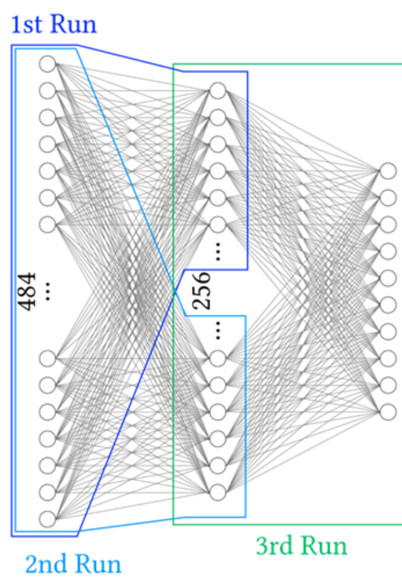

Figure 6: A visual representation on how the partition for a network topology of 22×22 → 256 → 10 works. Marked are the different partitions that can be run consecutively with the first two being interchangable.

5 Training on BSS-2

5.1 Methods

5.1.1 Data Set Transformations

In order to extract the maximum information out of the training data and to build a robust model, a few transformations are applied to the MNIST data set, i.e. the input images. It is important to notice that apart from a cropping or rescaling to match the training data, the test data set remains untouched.

By using the torchvision [6] transform RandomRotation, each image is tilted by some degree in the range of $[-25°, 25°]$. The degree by which each image is tilted will also be different every time the data set is accessed, effectively changing every epoch of the training. As the letters tend to decrease in intensity when using a bilinear interpolation method, the images are scaled by a factor of 1.5 after roation. To ensure the same range of values, each pixel value is clamped to [0,1]. This rotation method is applied with a probability of 0.5 thus augmenting the training data set.

As the original images of size 28×28 show a margin of around 2 pixels in each direction, one of the transformations is a center crop, cutting these margins and leaving the input data with a size of 22×22. This results in more input units spiking, thus encoding more detail of the image in contrast to just scaling the image down to the desired size. If after this center crop, a smaller input size is desired, this cropped version can be scaled down.

Another measure to build a more robust model and to avoid any kind of overfitting to the training data set itself is to add Gaussian noise to the images, which will be applied with a probability $p = 0.5$. As the test data does not have background noise, the noise will only be applied to the parts of the image that are close to the writing. This is done by rounding up the image values, resulting in non-zero values to be one and the rest to remain at zero. This image is blurred using a Gaussian blur and multiplied elementwise with the torch tensor holding the Gaussian noise. This is done in a last step of the transformations. It is important to notice the order in which these transformations are applied: Rotations are applied first as the quality of the resulting image is dependent on the resolution. The cropping and noise adding can be exchanged, although the given order might be slightly more efficient as no unnecessary computations are done in the cropping region of the image.

5.1.2 Time-to-First-Spike Encoding

While the idea of a time to first spike (TTFS) encoding has been mentioned before (section 2.3), I am not aware that there is any work regarding this encoding that uses the exact implementation that is explained in the following. The specific algorithm used here starts by mapping the gray scale values of each batch of images to the interval $[0, T]$. This happens via an affine linear transformation, resulting in the

minimum value v_{min} of the batch being mapped to 0, the maximum value v_{max} to T. For a batch size of 100, the value v_{min} is always equal to zero because of the background of the images. It is also very unlikely that the maximum value inside a batch is smaller than the maximum value for the total data set, making this procedure a global encoding. Nevertheless, for general values $x \in [v_{min}, v_{max}]$, this mapping is:

$$x \mapsto T \cdot \frac{(x - v_{min})}{v_{max} - v_{min}} =: x_1. \tag{23}$$

These values are then rounded and inverted in the following sense:

$$x_1 \mapsto T - \bar{x}_1 =: x_{ind}, \tag{24}$$

where we used the bar symbol to indicate the rounding to the next integer. x_{ind} now represents the index in the time dimension of the output tensor, where a spike will be encoded. As for a sequence length of T, the index T is out of range, which in this case is intentional, resulting in gray-scale values that are rounded to zero being represented by no spiking and high values to be encoded in early spikes. This choice also supports the sparsity in energy consumption of the BSS-2 system. This encoding method in general ensures maximum sparsity in input encoding and can be implemented using PyTorch methods.

For the specific use case of this work, the value $T = 30$ was chosen. It is important to notice that the sequence length is independent of the duration of the experiment for which spikes will be registered. It may be useful to chose a higher value for the latter. An example of an encoding is shown in figure 7.

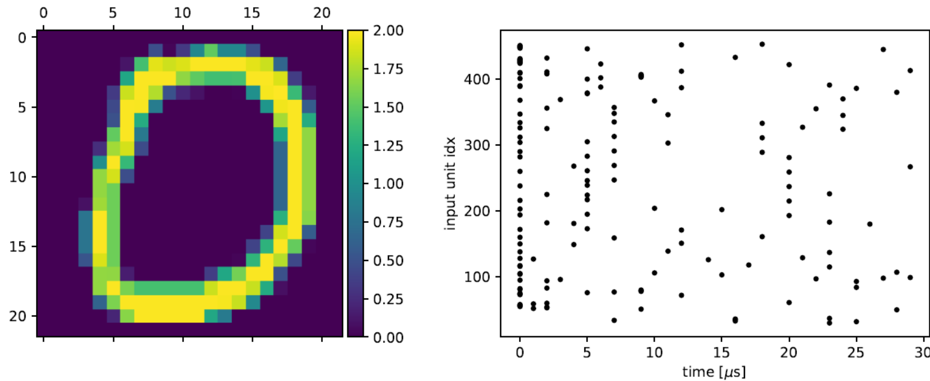

Figure 7: Encoding of a 0 using this implementation of a TTFS encoding.

5.2 Network Topology

With the MNIST data set being a former benchmark in the field of neural networks, there are many options for network topologies. In order to demonstrate network partitioning and to discuss the fundamental properties of ITL training with neuromorphic hardware, a rather simplistic model was chosen: With an input space of

$22 \times 22 = 484$, a hidden layer with 256 LIF-units and an output layer consisting of 10 LI-units, representing the ten classes of the MNIST data set, this model builds a baseline for these discussions. As derived previously in section 4.1.1, this network can be realized by using three partitions. Each of these parts will be assigned to an experiment within `hxtorch.snn` that can then be run consecutively.

5.3 Simulations and Mapping to Hardware

The simulation of membrane traces is of fundamental importance to the training of SNNs when using the hardware ITL approach of `hxtorch.snn`. With the membrane traces being deterministic in simulation, this does not hold for those on hardware. While this is for a variety of reasons, some of which will be investigated in the following sections, simulations in return give the opportunity to develop training methods for ideal behaviour while slowly approaching the difficulties of training with hardware. Another advantage of using the simulation mode regards execution times as there has to be no communication between the host and the hardware setup, meaning the transmission of information for input spikes, weight matrices and the read out membrane traces and spikes. In practice, simulation training has been faster by a factor of around 15, allowing much more progress in the exploration of methods within the same time. However, with hardware training being the ultimate goal, simulation proved models have been trained on hardware from time to time. Important changes to the model regarding the solvement of issues from hardware training can then also be applied to simulation models to obtain comparable data. Therefore, an approach of hand-in-hand development between softare simulation training and hardware ITL training with a BSS-2 setup was selected in this work.

As simulations (pure software executions) and hardware executions are entangled so fundamentally, a mapping between the properties of both are necessary. The goal of this mapping is to gain the same relative dynamics. This means that although the values for leak potential $V_{l,sw/hw}$, threshold $\vartheta_{sw/hw}$ and reset $V_{r,\ sw/hw}$ might differ for *sw/hw*, the PSP-heights should be (almost) identical relative to $\vartheta_{sw/hw} - V_{l,sw/hw}$. This relative equivalence can only be achieved when it also holds for the reset values:

$$\frac{V_{l,sw} - V_{r,sw}}{\vartheta_{sw} - V_{l,sw}} = \frac{V_{l,hw} - V_{r,hw}}{\vartheta_{hw} - V_{l,hw}}. \tag{25}$$

This property can be easily satisfied by choosing $V_{l,sw} = V_{r,sw}$ and $V_{l,hw} = V_{r,hw}$. For the PSP-heights however, a more rigourus calculation has to be done.

Therefore, we first notice that the relation between the PSP-heights and the weights is linear:

$$m_{sw} := \max(V_{m,sw}(t)) = a\,w_{sw} \quad \text{and} \tag{26}$$

$$m_{hw} := \max(V_{m,hw}(t)) = b\,w_{hw} \tag{27}$$

with constants $a, b \in \mathbb{R}^+$ and $w_{sw} \in \mathbb{Q}$, $w_{hw} \in \{-63, -62, ..., 62, 63\}$. With both of

these relations being linear, an obvious mapping between m_{sw} and m_{hw} is also linear:

$$cm_{sw} = m_{hw}, \quad c \in \mathbb{R}^+. \tag{28}$$

For spiking neurons, it is $m_{sw} = \vartheta_{sw}$ and $m_{hw} = \vartheta_{hw}$ and thus yields $c = \vartheta_{hw}/\vartheta_{sw}$. Together with equation 27 we find

$$w_{hw} = \frac{m_{hw}}{b} = \frac{cm_{sw}}{b} = \frac{caw_{sw}}{b} = \frac{a}{b}\frac{\vartheta_{hw}}{\vartheta_{sw}}w_{sw}. \tag{29}$$

Weights in software representation therefore have to be scaled with a factor of $\frac{a}{b}\frac{\vartheta_{hw}}{\vartheta_{sw}}$ when mapping to hardware and the read out membrane traces from hardware executions have to be scaled with $\vartheta_{sw}/\vartheta_{hw}$. These two procedures will be reffered to as weigh scaling and trace scaling, respectively. While this procedure in general can be performed for each neuron of the chip individually, in practice the mean values were used and showed sufficient accuracy (see section 5.6).

As weights on a BSS-2 system are also disrcete, weights in software representation will be rounded after scaling. By choosing values for ϑ_{hw} and $V_{l,hw}$, the dynamical range of weights in software representation can be adjusted. Another possibility for this is to change the scaling b with its dependency on the hardware parameters 'i_synin_gm' and 'synapse_dac_bias' (for more information see [14]).

As will be apparent from following sections, it might be useful to use a different calibration of the neurons in the output layer. As the readout layers were chosen to be LIs, reasonable changes in the calibration only influence b. To adapt for these changes and match the simulated traces, the trace scaling has to be reconsidered but with the same calculations as above.

Table 1 shows which parameter values were chosen for LIF and LI neurons.

Parameter	Software Value	Hardware Set-Value
ϑ	1	120
$V_l = V_r$	0	80
τ_{mem}	5.7 μs	5.7 μs
τ_{syn}	6 μs	6 μs
i_synin_gm (LIF)	–	1000
i_synin_gm (LI)	–	400
synapse_dac_bias (LIF)	–	1000
synapse_dac_bias (LI)	–	400

Table 1: Typical calibration parameters for LIF and LI neurons used throughout this work.

5.4 Hardware behaviour

Before training with the BSS-2-system, we will examine a few basic properties of the hardware. As an example, we will set up a basic experiment, showcasing the dynamics of the neuron circuits and afterwards inspect PSP-heights for different inputs to neuron compartments.

5.4.1 A Basic Experiment

In a first basic experiment, we will investigate the dynamics of a LIF circuit on a BSS-2 setup. Therefore, a spike train will directed to multiple synapses leading to an atomic neuron on the chip. The membrane trace and the output spikes of the neuron will be read out and can be investigated and compared to the results of a simulation of this experiment. The results of this experiment are shown in figure 8. One can clearly see the variations of the output traces when repeatedly performing the same experiment. There are even traces that do not show any spike where as the simulation does. This displays one of the basic differences between simulation training and training with hardware in-the-loop that will be discussed further after reviewing the training results.

5.4.2 Investigating PSP-Heights

The investigation of PSP-heights is a necessary step when calculating the mapping between software and hardware (parameter 'b'). We will conduct an experiment showcasing how the PSP-heights grow linearly when increasing the weights and at the same time compare them for inputs to different atomic neurons of the same compartment. As neuron compartments are a fundamental resource for training larger scale neural networks, this experiment will show if there are non-neglegiable effects when using compartments. In this case, the largest neuron size that is used for the MNIST model is $c = 4$. Figure 9 shows, how the PSP-heights relate to the weight and how this relationship changes when targetting synapses that connect to different atomic neurons than the one being read out.

	for negative weights		for positive weights	
Neuron Nr.	m_-	d_-	m_+	d_+
1	0.3993 ± 0.0016	-0.40 ± 0.06	$0.3942 \pm 0{,}0019$	0.41 ± 0.07
2	0.4050 ± 0.0015	-0.39 ± 0.06	0.4018 ± 0.0018	0.34 ± 0.07
3	0.3927 ± 0.0013	-0.46 ± 0.05	0.3894 ± 0.0016	0.33 ± 0.06
4	0.3912 ± 0.0015	-0.46 ± 0.06	0.3880 ± 0.0018	0.33 ± 0.07

Table 2: Values found when optimizing for linear fits of the form $y = mx + d$ for the PSP-heights shown in figure 9.

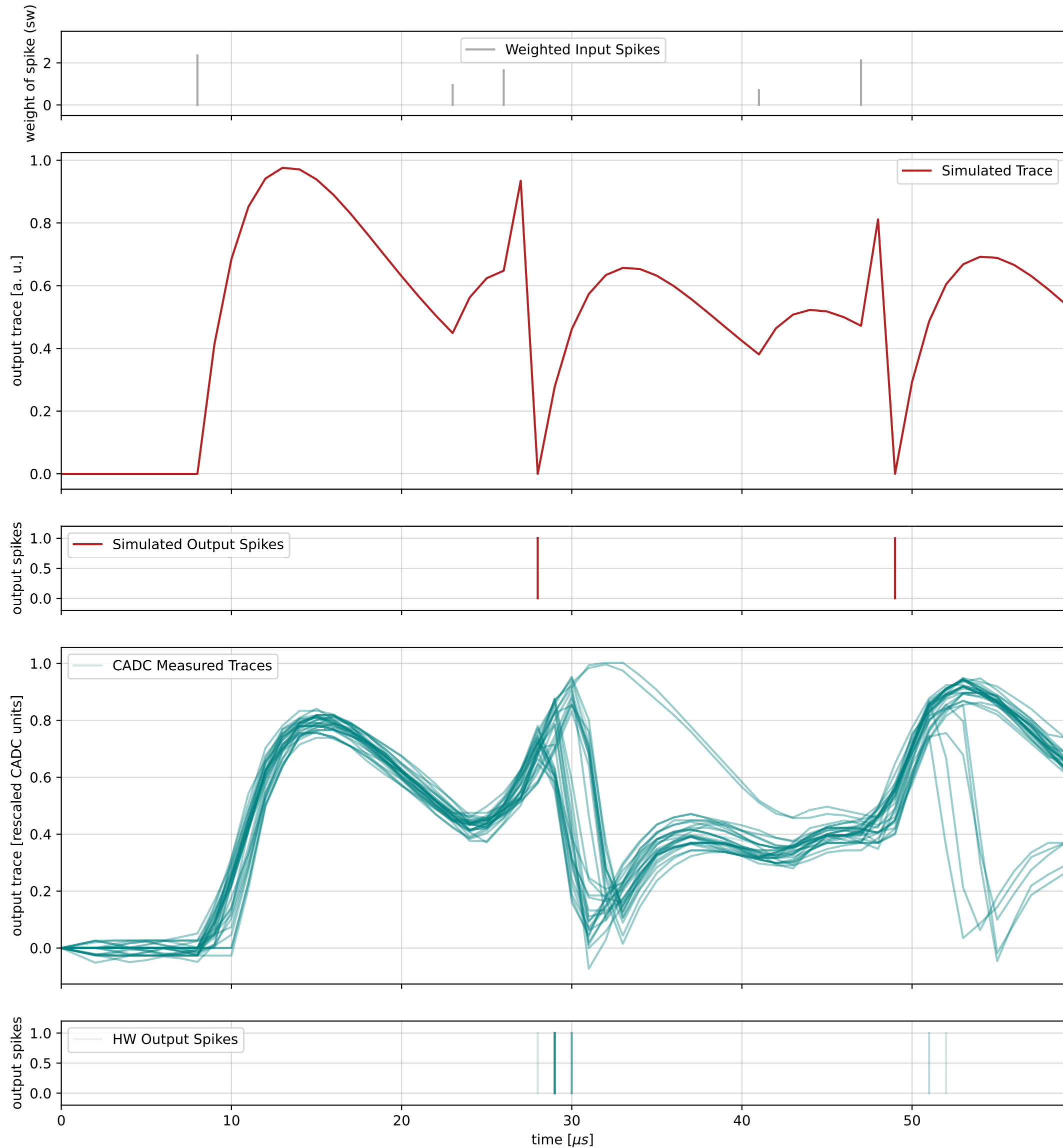

Figure 8: Simulated membrane traces and measured traces on hardware in comparison. The spike train shown in the uppermost plot was directed to the membrane of a LIF neuron 30 times, the resulting traces and output spikes of which are the lower two plots. For comparison also the outputs of the same experiment in simulation are displayed above.

While the linear fits diverge slightly for higher weights they are well contained within the standard deviation of the data from the fits. Also, for negative and positive weights,separate fits were performed and when looking closely one can see the different y-intercept. The reason for that is that the PSP-height was here defined

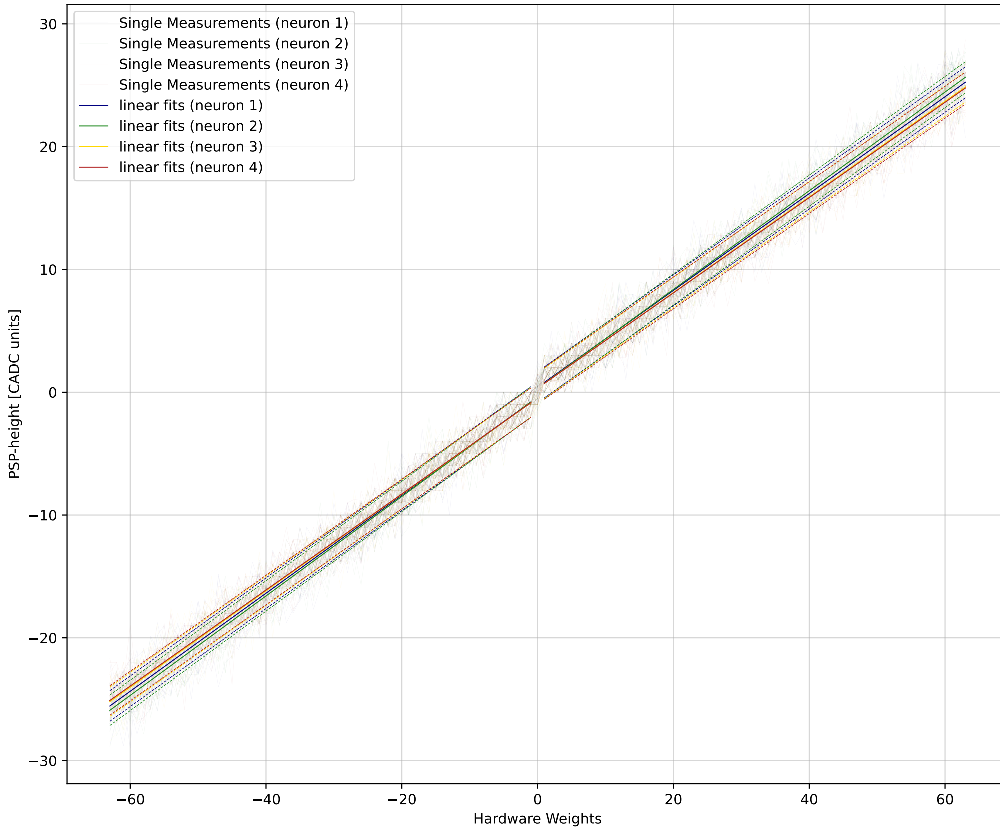

Figure 9: When a single spike is weighted and directed to the membrane of a LI on a BSS-2-setup, the peak hight of the PSPs show this behaviour. The results of inputs directed to different atomic neurons within the same larger neuron are shown in different colors together with linear fits and the standard deviation of the data to respective fit (dashed lines of same color). The optimal values for the functions are listed in table 2.

as the maximum (or minimum) value of the trace - that is subject to noise which originates from the combined effects of the electronics generating the trace but also the CADC readout. The differences in y-intercepts for the fits for negative and positive weights to some degree give information on the scale of noise of the 'actual' PSP-heights. Precisely, the y-intercept directly resembles the bias to the assigned PSP-height by choosing the maximum value of the output trace (in comparison to for example the maximum of a fit) in addition to fundamental biases in the electronics. While the distinction between these can only be made when taking a closer look at the traces themselves and performing fits, the important takeaway is that the standard deviation shown in in figure 9 is mainly a representation of noise by the readout of the membrane traces. Also, the linear correlation between the PSP-height and the weight of the input spike is shown and fulfilled with great accuracy.

5.4.3 Saturation Issues and Resolving Methods

When approaching training with neuromorphic hardware, dynamical ranges and saturation are important subjects to investigate on as certain components involved in the training are restricted in their range. One of the main components was already mentioned, the weights. A configuration within $\{-63, -62, ..., 62, 63\}$ combined with the mapping to software, it should be adressed how to deal with weights that reach these limits while training.

One option is to use a regularisation in form of a loss on all weights of a network to prevent the weights from reaching the limits of the dynamical range. An additional method is to use a function that is applied to the weights before using them, that maps to the dynamical range. This function should maintain a few properties: the weight scaling should be accurate for most parts of the dynamical range and it should contain the weights within the dynamical range to closely represent the behaviour on hardware. The function $f : \mathbb{R} \to (-\hat{w}, \hat{w}), \hat{w} \in \mathbb{R}^+$ with

$$w \mapsto f(w) = \begin{cases} w & \text{for } \frac{|w|}{\hat{w}} \leq 1 - \frac{1}{a} \\ \text{sign}(w)\hat{w} \cdot \left(1 - \frac{1}{a} \exp\left(-a\left(\frac{|w|}{\hat{w}} - \left(1 - \frac{1}{a}\right)\right)\right)\right) & \text{else} \end{cases}, \tag{30}$$

$$a = \left(1 - \frac{w_s}{63}\right)^{-1} \tag{31}$$

with $w_s \in (0, 63)$ being the starting value of the roll off. Figure 10 shows this function. These two methods adress the issue of saturating weights which can lead to uncontrollably growing weights while not measuring the impact due to the limits for hardware weights and thus interfering with the training process. A notable feature of this mapping function is also that it is differentiable everywhere with a non zero derivative, meaning that even saturated weights are affected by backpropagation. The other main component that can become subject to saturation issues is the readout of the CADC. Especially when using a very sensitive calibration for LIF neurons for a high dynamical range for weights and applying the same properties to a LI, saturation is likely. A solution to this is a combination of a regularization on the traces themselves or the scores and a different calibration, resulting in less sensitive LI-membranes.
Saturated traces can even favor weight saturation: When traces are supposed to have a high PSP relative to the ones representing other classes (which is the ideal case for a correct and certain classification) but are unable to stand out from the other traces due to saturation of the traces, backpropagation will try to increase the 'correct' trace - which is done by increasing weights. Figure 11 shows an example of traces that are saturated. While for the traces on the left the classification of a 5 works, the next example of a 9 is being falsely classified eventhough the respective trace is clearly the first to go into saturation and should return the highest trace. Next to weights and membrane traces also the mean firing rate of a network is subject to regularization. The loss function is therefore being extended to regularization terms for each of the quantities. While starting with an MSE loss, an alteration using biquadratics ('mean

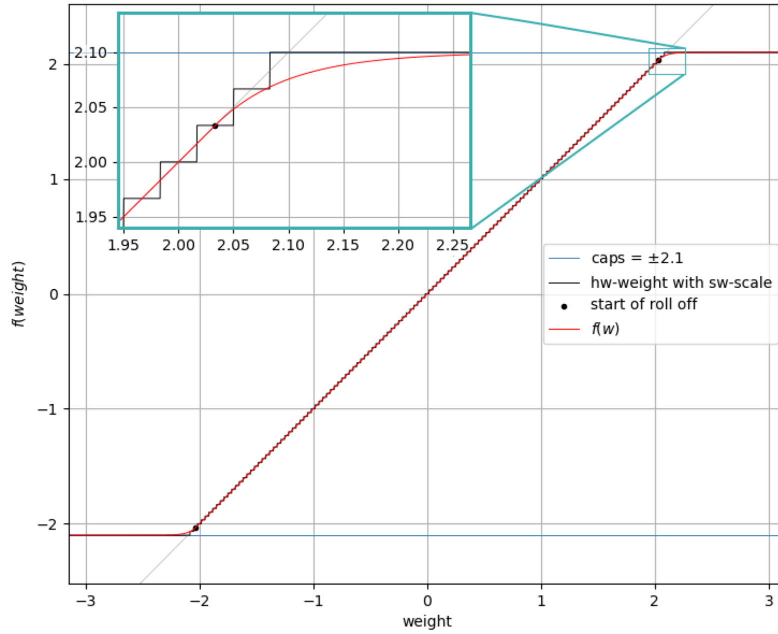

Figure 10: Graph of the clamping function f with typical parameters $\hat{w} \hat{=}$ 'cap' = 2.1 and $w_s \hat{=}$ 'start of roll off' = 61. The step function that is effectively applied when using discrete software weight is shown in black, downscaled to match the software representation of the weights. In light gray also the identity function is displayed for reference.

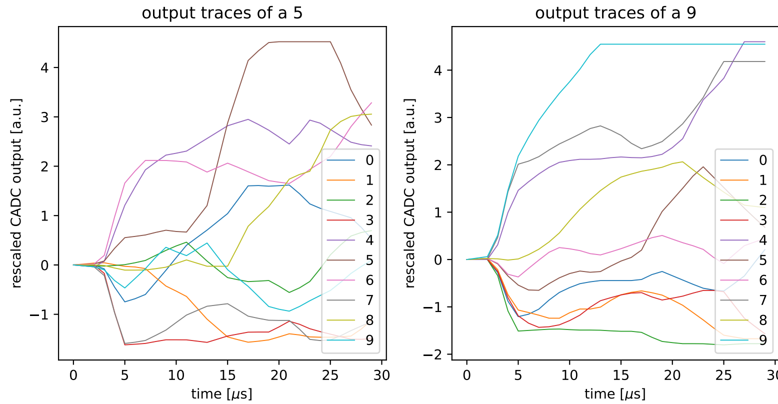

Figure 11: Output traces of ten LI neurons in the output layer of one of the networks classifying two inputs (a five and a nine) while being affected by saturation due to a poor choice of parameters leading to too sensitive membranes and potentially false classifications.

biquadratic error') proved to be more stable in the training. For the N weights w_{ij}

of the network the regularization term is

$$R_w = C_{R_w} \cdot \sum_{i,j} \frac{w_{ij}^4}{N} \tag{32}$$

with a constant $C_{R_w} \in \mathbb{Q}$.

5.5 Simulation Results

With all the methods discussed above, training on BSS-2 can be performed. Nevertheless, this section first presents the results of simulations as a baseline for comparison. Multiple networks were trained with the network topology of $22 \times 22 \rightarrow 256$ LIF $\rightarrow 10$ LI and a total of three partitions. As for the simulations, there are in principle no limitations to the weights, a maximum value of $\hat{w} = 2$ was implemented with the method discussed in section 5.4.3. This value is of good representation for the hardware with the used calibrations.

When using the not augmented training data set and after 100 epochs of training, an accuracy of $97.57\% \pm 0.08\%$ could be achieved on the test data. Figure 12 shows how the accuracy and loss evolved during training. Especially in early epochs, the

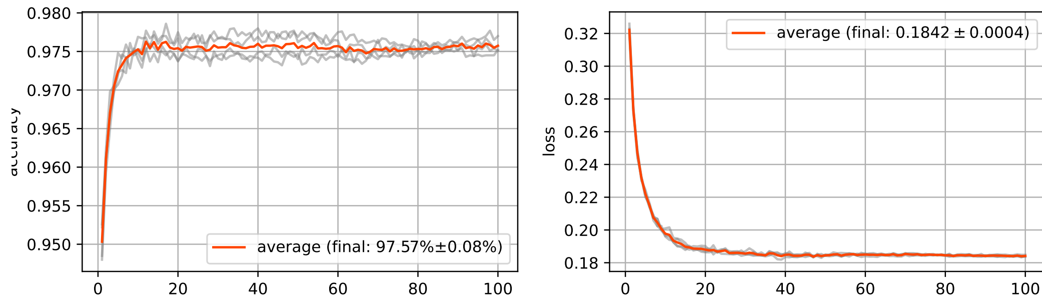

Figure 12: Test accuracies (left) and losses (right) of a simulation model not making use of training set augmentation.

accuracy of the networks differ significantly for different seeds. This effect shrinks but not fully disappears with ongoing training. The final firing rate of these networks reaches $(0.338 \pm 0.003)\frac{\text{spikes}}{\text{neuron}\cdot\text{input}}$.

In an effort to improve on the networks performances and to prevent overfitting, augmentations to the training set were implemented. With rotations of up to 25° and noise (see section 5.1.1) the models could be improved to accuracies of 98.38 ± 0.08 with a slightly increased firing rate of $(0.377 \pm 0.010)\frac{\text{spikes}}{\text{neuron}\cdot\text{input}}$. While with the previous model the accuracy and loss approach their final values after around 50 epochs, figure 13 shows that especially the loss is steadily decreasing up until the final epoch. This indicates that the model has note yet reached a final state. But To keep these simulations comparable to the hardware models where training is much more time consuming with about one and a half days per model, the number of training

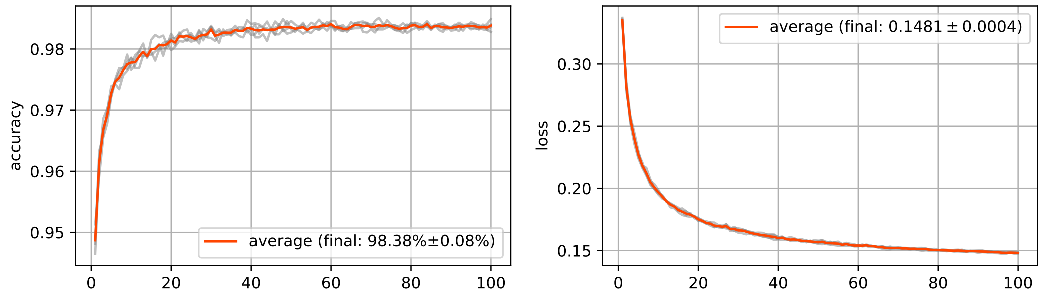

Figure 13: Test accuracies (left) and losses (right) of a simulation model with the use of augmentation methods (rotation and noise).

epochs was limited to 100.

When using a dropout of 20% in the hidden layer, the accuracy on the test data increased slightly but not significantly to $98.42\% \pm 0.06\%$ with a final loss off 0.1484 ± 0.0008 and a similar firing rate as before $(0.378 \pm 0.008 \frac{\text{spikes}}{\text{neuron}\cdot\text{input}})$. Again, at epoch 100, the loss is at a steady decrease, and also the accuracy seams to be improvable with further training. In table 3, the most important hyperparameters

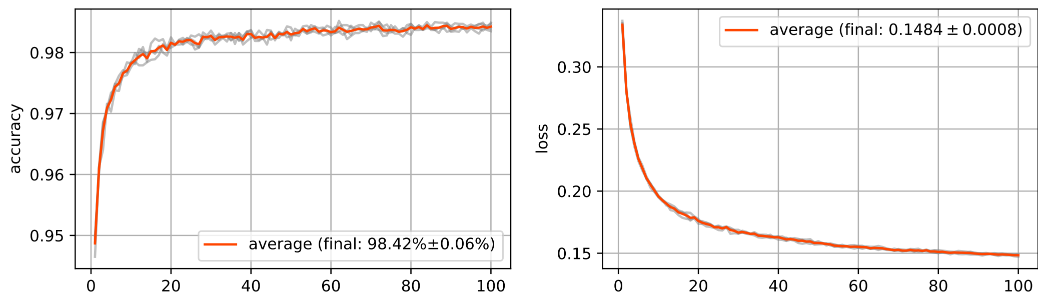

Figure 14: Test accuracies (left) and losses (right) of a simulation model with the use of augmentation methods (rotation and noise) and dropout of 20% in the hidden layer.

used for the simulation training are listed.

Figure 15 shows the observables of an execution of one of the best networks as well as how the firing rate evolves during training. Especially the latter shows an interesting behaviour: At first, the rate decreases with a steep descent but from epoch 25 on increases almost linearly up until epoch 100. All other simulation models show very similar behaviour.

hyperparameter	value
batch size	100
learning rate decay	0.002 $\times 0.97$ per epoch
C_{bursts}	$8 \cdot 10^{-4}$
$C_{h\text{-}weights}$	$6 \cdot 10^{-3}$
$C_{o\text{-}weights}$	$6 \cdot 10^{-3}$
$C_{readout}$	$4 \cdot 10^{-5}$

Table 3: Important hyperparameters used while training simulation models. The regularization coefficients are labeled with $C_{\bullet}$.

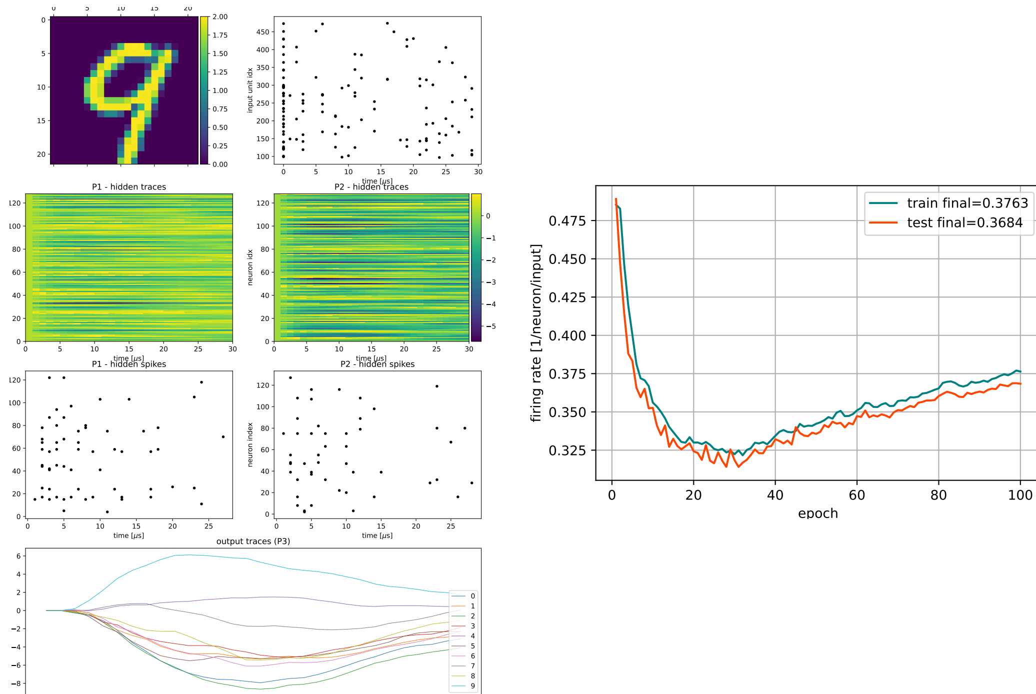

Figure 15: **Left:** Representation of an execution of a simulation model with the observables of the network. T.l.: The input image. T.r.: The encoding of this image using TTFS encoding. The four images in the middle represent the membrane traces and spikes in the hidden layer for both partitions. At the bottom, the output traces of the last LI-layer are displayed. From this, one can see that the classification of the 9 was successful. **Right:** Firing rate in the hidden layer during training.

5.6 Multi Single-Chip Execution on BSS-2

When approaching training on hardware, a few adjustments regarding the regularization terms had to be made as the training turned out to be less stable with hardware

in the loop. This mainly involved increasing the coefficients of the regularization loss terms. Also, when using MSE-losses, traces were more likely to go into saturation in some cases, interfering with the training process. With the idea to restrict the membrane voltages more while punishing smaller values less, the MSE loss was altered to a 'biquadratic' mean error (see equation 32). With the values listed in table 4 a relatively stable training process could be guaranteed. On top of that, also weights showed an increased growth. To minimize saturation effects, a relatively high efficacy was chosen for the calibration of the LIFs-neurons, allowing weights up to ± 2.1 or even ± 2.4 for some setups.

hyperparameter	value
batch size	100
learning rate decay	0.002 $\times 0.97$ per epoch
C_{bursts}	$5 \cdot 10^{-3}$
$C_{h\text{-}weights}$	$9 \cdot 10^{-3}$
$C_{o\text{-}weights}$	$9.2 \cdot 10^{-3}$
$C_{readout}$	$1 \cdot 10^{-4}$

Table 4: Important hyperparameters used while training models with hardware in the loop. The regularization coefficients are labeled with $C_{\bullet}$.

To test and quantify the improvements that can be made with hardware training, a model was trained with the raw, not augmented data set, and achieved an accuracy of 96.48% with a loss of 0.7728 (see figure 16). Also, the firing rate reached $0.395 \frac{\text{spikes}}{\text{neuron} \cdot \text{input}}$ in the final epoch. With the use of data set augmentation, the performance could be further improved to $97.095\% \pm 0.015\%$ with a loss of 0.2639 ± 0.0016. The firing rate reached $(0.75 \pm 0.03) \frac{\text{spikes}}{\text{neuron} \cdot \text{input}}$.

In the same procedure as for the simulation models also a dropout in the hidden layer with a probability of 20% was included into the model, but did not show any improvement when used with hardware ITL training. On the contrary, the performance even decreased slightly to an accuracy of 96.93% with a loss of 0.762%.

When reviewing the progression of the networks weights, a main difference to the simulations is that despite the use of higher regularization values, the distribution of weights becomes broader and distributes over the full range after about 40 epochs.

5.6.1 Final Results

As the main difference between the simulation and the execution on a BSS-2-setup is the spiking behaviour, that was investigated in section 5.4.1, an additional model

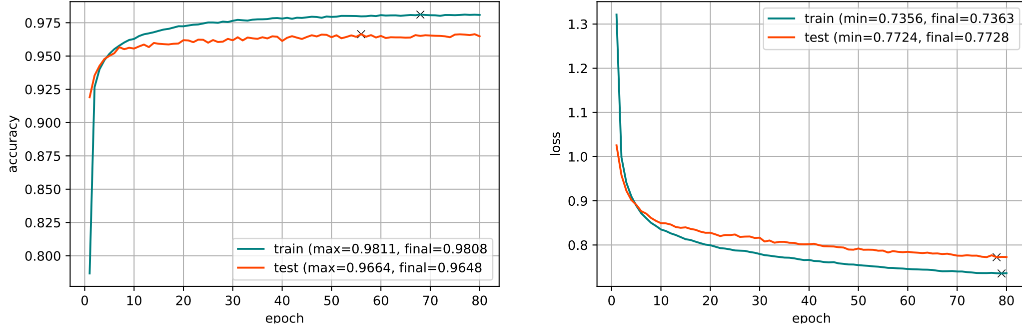

Figure 16: Test accuracy (left) and loss (right) of a model trained with hardware in the loop on BSS-2 while not making use of training set augmentation but the raw (cropped) data.

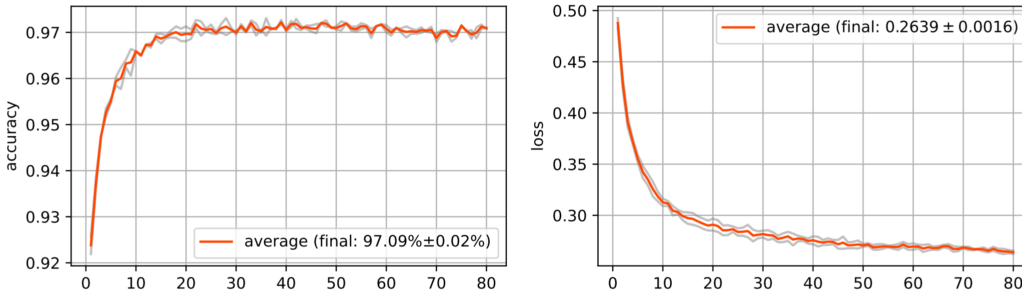

Figure 17: Test accuracies (left) and losses (right) of a two models trained with hardware in the loop on two different BSS-2-setups while making use of training set augmentation (rotation and noise).

was trained with a lower synaptic efficacy to reduce noise on the chip. Although this implies that the range of possible weights in the software representation shrinks, a slightly increased accuracy of 97.22% with a loss of 0.2312 could be achieved after only 60 epochs of training. For this best model that could be trained within the limitations of the duration of this bachelors thesis, figure 18 shows the observables during a classification as well as the firing rate. With the same values for regularization as before, the weights go into the saturation range earlier but do not overshoot when applied on hardware due to the regularization function f for the weights (see section 5.4.3).

Compared to previous results for the MNIST data set ([18]), the simulated models of this work could exceed the performance by about 0.4%, whereas the models trained with hardware in the loop, were slightly worse by about 0.3%.

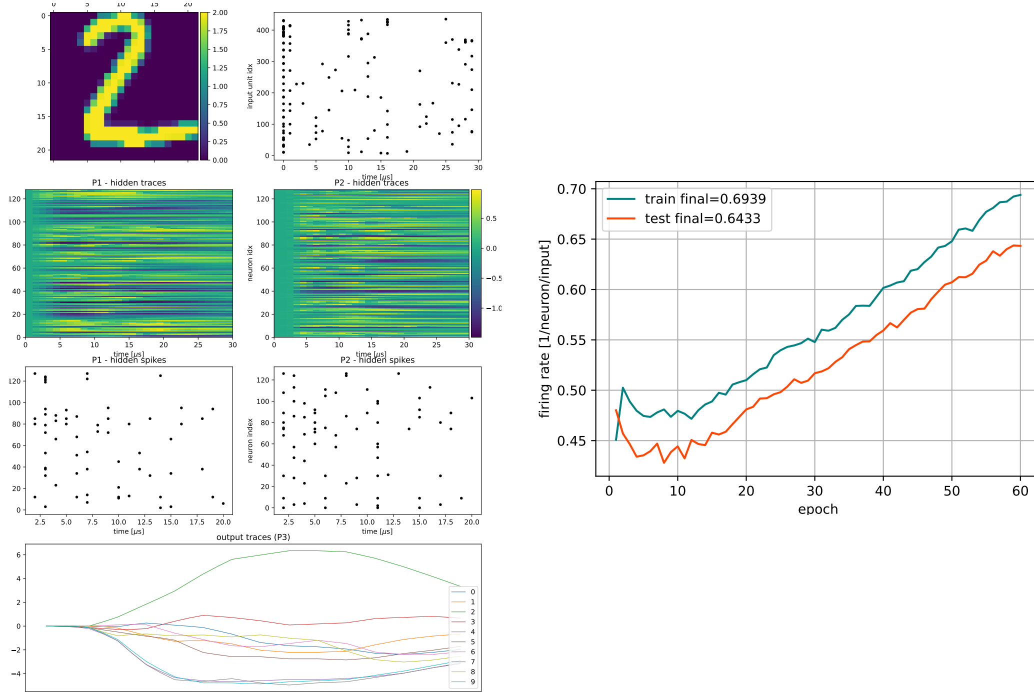

Figure 18: **Left:** Representation of an execution of a model trained on BSS-2 with the observables of the network. T.l.: The input image. T.r.: The encoding of this image using TTFS encoding. The four images in the middle represent the membrane traces and spikes in the hidden layer for both partitions. At the bottom, the output traces of the last LI-layer recorded by the CADC are displayed. From this, one can see that the classification of the 6 was successful. **Right:** Firing rate in the hidden layer during training.

6 Discussion and Outlook

The goal of this work was to approach the executions and trainings of larger scale SNNs with BSS-2. To give a foundation for this, the first chapters covered the theoretical background, starting from biological neurons and neuron models such as the LIF model and ending with an overview on how training in SNNs can be accomplished using gradient based methods. This included the discussion of encoding and decoding principles as well as BPTT and surrogate gradients as effective and robust methods.

After introducing the BSS-2 system and the important properties for this work, it was demonstrated, how the execution and training of larger scale feedforward networks on BSS-2 can be accomplished. It was discussed that the basis for this is the scalability of synaptic inputs, introduced by neuron compartments, as well as the exploitation of independent dynamics within feedforward networks, which enables partitioning. An algorithm for this task given the limitations of neuromorphic hardware in a more general case was developed and implemented within the the software framework `hxtorch.snn`. With an investigation of important functionalities of the hardware and training on the MNIST data set with a network topology that requires partitioning, an example on how the development and tuning process when training on BSS-2 can look like, was presented. This included the usage of common data set augmentation methods and a TTFS encoding, both implemented in a customized fashion. For training with limitations posed by hardware constraints, methods were developed to deal with these constraints. In particular, a function for the regularization of weights was proposed and existing regularization methods with loss functions were modified to cope with the difficulties that arose during the training.

The implemented methods were tested in simulation as well as on BSS-2. In both cases training performed well, especially in the simulation, baseline models could be improved upon significantly and performed up to $98.42\% \pm 0.06\%$ on the MNIST training data in terms of accuracies while showing low firing rates around $0.378 \pm 0.008 \frac{\text{spikes}}{\text{neuron}\cdot\text{input}}$. In the case of hardware ITL-training on BSS-2, the introduced baseline models could also be improved upon, but showed less improvement than for the simulation. The best model that could be trained within the time limitations of this thesis achieved an accuracy of 97.22% with a firing rate of $0.643 \frac{\text{spikes}}{\text{neuron}\cdot\text{input}}$. While for simulations, the best MNIST results outperform previous simulation models with smaller network sizes by about 0.4%, the results from hardware in the loop training were not able to compete with previous results, leaving with a deficit of around 0.3%. The differences in the performance between simulation and hardware executions root mainly from the qualitative differences in spiking behaviour. The basic experiment showed in section 5.4.1 demonstrates these differences for the calibrations that were used for most of the hardware models. Improvements would therefore aim to align the spiking behavior with the simulation. Approaches to this could be a minimization of noise to the signals by using different calibration parameters. As in this case a high synaptic efficacy was chosen with the idea to

increase the range of weights in software representation, a next step could be to decrease the efficacy while trying to maintain a stable training with lower ranges for the weights. For these cases, also other regularization parameters might have to be adjusted to keep saturation effects at a low level. Also, regarding the mapping between hardware and software, the weight scaling was implemented to be a median value that was applied to all weights. To optimize this mapping, a distinct weight scaling for each neuron circuit on BSS-2 could be implemented. This might even be the reason for the slightly lower output traces of the LIF-neuron measurements of the basic experiment. With individual weight scalings, individual traces scalings could also help aligning the behaviour between simulation and hardware execution. Finally, another difference are the discrete weight values on hardware whereas in software representation and also for the simulations, weights are defined as any floating points value. Another improvement could be to incorporate this discrete behaviour into the simulated traces that are used with hardware ITL training with surrogate gradients in `hxtorch.snn`. With an investigation on the size of weight updates during training, even a surrogate gradient for the step function that represents the rounding of weights could be introduces if weight updates are small enough.
An effect which both, simulations and hardware executions show, is that after a few epochs of training, the networks firing rate starts to increase. While this is not a fatal property given the amount of training epochs, this tendency could become problematic when working with more complex data sets that require more epochs of training. As the diagrams of losses during training show, at some point in training the loss is on a steady decrease but the accuracy doesn't improve any further. This means that the minimization of loss leads to a trade-off between the desired features that are implemented as regularization terms. These effects can be minimized by tuning the regularization coefficients, leading to a more balanced state after the accuracies reach plateaus.

Nevertheless, these results prove the effectiveness of the developed and implemented methods while still leaving some room for optimization, but most importantly, demonstrate that hardware ITL-training and executions of larger scale networks on a single BSS-2 chip are possible with the current state of development.

With the basis of this work, there are plenty of research and development options. Firstly, other kinds of network structures could be investigated. For example, networks involving convolutions would likely allow improved performance on image recognition tasks. With the scalability of network partitioning in mind, data sets like EuroSAT [8, 10] could be approached. While in this work, only a single BSS-2 setup was used, with the Electronic Visions group working on multi-chip setups, training and execution processes could be sped up as partitions of the same layer could be processed in parallel.

References and Sources

[1] Yann LeCun and Corinna Cortes. *The MNIST database of handwritten digits.* 1998.

[2] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002. DOI: `10.1017/CBO9780511815706`.

[3] David Iverri. *File:action potential.svg.* 2006. URL: `https://upload.wikimedia.org/wikipedia/commons/4/4a/Action_potential.svg`.

[4] Marlis Hochbruck, Alexander Ostermann, and Julia Schweitzer. "Exponential Rosenbrock-Type Methods". In: *SIAM Journal on Numerical Analysis* 47.1 (2009), pp. 786–803. DOI: `10.1137/080717717`. eprint: `https://doi.org/10.1137/080717717`. URL: `https://doi.org/10.1137/080717717`.

[5] Quasar Jarosz. *Neuron Hand-tuned.* 2009. URL: `https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg`.

[6] TorchVision maintainers and contributors. *TorchVision: PyTorch's Computer Vision library.* `https://github.com/pytorch/vision`. 2016.

[7] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific containers for mobility of compute". In: *PLOS ONE* 12.5 (May 2017), pp. 1–20. DOI: `10.1371/journal.pone.0177459`.

[8] Patrick Helber et al. "Introducing EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification". In: *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium.* IEEE. 2018, pp. 204–207.

[9] Chetan Singh Thakur et al. "Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain". In: *Frontiers in Neuroscience* 12 (2018), p. 891. ISSN: 1662-453X. DOI: `10.3389/fnins.2018.00891`. URL: `https://www.frontiersin.org/article/10.3389/fnins.2018.00891`.

[10] Patrick Helber et al. "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* (2019).

[11] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: `10.1109/MSP.2019.2931595`.

[12] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[13] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[14] Philipp Dauer. “Characterization of silicon neurons on HICANN-X v2”. Bachelorarbeit. Universität Heidelberg, 2020.

[15] Johannes Schemmel et al. “Accelerated Analog Neuromorphic Computing”. In: *arXiv preprint* (2020). arXiv: 2003.11996 [cs.NE]. URL: https://arxiv.org/abs/2003.11996.

[16] Elias Arnold. “Biologically Inspired Learning in Recurrent Spiking Neural Networks on Neuromorphic Hardware”. Masterarbeit. Heidelberg University, 2021.

[17] Friedemann Zenke and Tim P. Vogels. “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks”. In: *Neural Computation* 33.4 (Mar. 2021), pp. 899–925. ISSN: 0899-7667. DOI: 10.1162/neco_a_01367. eprint: https://direct.mit.edu/neco/article-pdf/33/4/899/1902294/neco_a_01367.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01367.

[18] Benjamin Cramer et al. “Surrogate gradients for analog neuromorphic computing”. In: *Proceedings of the National Academy of Sciences* 119.4 (2022).

[19] Christian Pehle et al. “The BrainScaleS-2 Accelerated Neuromorphic System with Hybrid Plasticity”. In: *Front. Neurosci.* 16 (2022). ISSN: 1662-453X. DOI: 10.3389/fnins.2022.795876. arXiv: 2201.11063 [cs.NE]. URL: https://www.frontiersin.org/articles/10.3389/fnins.2022.795876.

[20] Philipp Spilger et al. “hxtorch.snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2”. In: *Neuro-inspired Computational Elements Workshop (NICE 2023)*. Accepted. University of Texas, San Antonio, USA: Association for Computing Machinery, 2023. arXiv: 2212.12210 [cs.NE].

Acknowledgements

First, I want to thank Dr. Johannes Schemmel for giving me the opportunity to work with the Electronic Vision(s) group and to write my Bachelor's Thesis under his supervision.

I also want to thank Elias Arnold and Dr. Eric Müller for guiding me through the thesis and introducing me to the science of spiking neural networks. I am very grateful to have the opportunity to work on these interesting topics.

Thanks to Philipp Spilger, Jakob Kaiser for proofreading and Luca Blessing for helping me out with mapping-related questions.

Special thanks to my family and Mary-Lou for always supporting me during this time and had my back when it got stressful.

The work carried out in this Bachelors's Thesis used systems and software, which received funding from the EC Horizon 2020 Framework Programme under grant agreements 785907 (HBP SGA2) and 945539 (HBP SGA3), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2181/1-390900948 (the Heidelberg STRUCTURES Excellence Cluster), the German Federal Ministry of Education and Research under grant number 16ES1127 as part of the Pilotinnovationswettbewerb 'Energieeffizientes KI-System', and the Lautenschläger-Forschungspreis 2018 for Karlheinz Meier.

A Acronyms

Acronyms

ANN artificial neural network. 1, 2, 7, 8, 36

BPTT back propagation through time. 31, 36

BSS-2 BrainScaleS-2. 1, 2, 6, 8, 11–14, 17–20, 22, 25, 28–32, 36

CADC columnar analogue to digital converter. 11, 22, 23, 30, 36

ITL in-the-loop. 12, 17, 18, 28, 31, 32, 36

LI leaky integrator. 4, 7, 11, 18, 19, 22–25, 27, 30, 36

LIF leaky integrate-and-fire. 1, 2, 4–6, 9, 11, 12, 18–21, 23, 25, 28, 31, 32, 36

ML machine learning. 1, 36

MSE mean squared error. 8, 23, 28, 36

NLL negative log likelihood. 8, 36

PSP postsynaptic potential. 2, 18, 20–23, 36

RNN recurrent neural network. 9, 36

SNN spiking neural network. 1, 2, 6–10, 12, 18, 31, 36

TTFS time to first spike. 16, 17, 27, 30, 31, 36

B Software State

The experiments conducted in this thesis used the `singularity` [7] container `/containers/stable/2023-05-25_1.img` and `gls` app. The following table lists the software state at the end of the thesis. Additionally, the change set id for all the files used for training and plotting is 20776.

Repository	Commit-Hash
repo_db	740a6abd9e2b0733385e4ed8a5442874e99eee6a
bss-hw-params	8e5e18ebbdece63890eebd4f082084111846e965
calix	bc485dad6ad511f88be6d739f16455a033c9ee8f
code-format	24b533dd390253f5c698708fa735283c2e7282ca
extoll-driver	a0ffdc9ea5517e11bc126c0b9d54e7dca2f1dc07
fisch	976cf3d1a2d2deef69673cca940e914b7fa41fef
flange	298335251daf2d2a02764d888e02566934ad2ec9
grenade	45779707beb8228150b4788118369feb4c005576
halco	bbb5996633cfb64507b435ad53d6a9a7fee26c31
haldls	209a815d68a3fa4f643ffb6c1fc843b31e9e80c4
hate	0143bd6e177cdc2ec1f49e9276884078f3976b22
hwdb	e738de3d2ed818ae9a0e9f397c1b570df85d7ea4
hxcomm	92607bcbb3ec8cb7ba4b36de04d7d93076b43859
hxtorch	f16d7e274ef9ff9f29b9bd60dc289b826ab3e150
lib-boost-patches	ed89665b4c066629b69617ede2e8b1fbe65822d9
lib-rcf	4ac48ea216e1e9026cd0d25f52a4bf683d97a189
libnux	9b335cb56b6faa447ffda27f5ff310e648502071
librma	5159e3c602e0133c74800a33a4042f7aeebfb00f
logger	00380efdec521fb08df4a083a1d1f443fef836e4
nhtl-extoll	2d7098e2364141ebc0db2a71982570a46d68c7b8
pywrap	8eda91fcca8bfccb946a0ee5b40ca82b5b15650e
rant	0d494ce6eedfb74889cf7cee09105258819acb35
sctrltp	42a988e986906f177102813418d5fd22dd646b44
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3
ztl	773660f435e56b1ee7b962e8babfe004ff487cdd

Table 5: Software state.

C Declaration/Erklärung

I hereby declare that I have written this thesis independently and have not used any materials or aids other than those indicated.

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

Heidelberg, den 07.07.2023: ....

Jan Valentin Straub