Internship Report: Towards Multi-Chip Training of Spiking Networks with BrainScaleS-2

Jan Valentin Straub under supervision of Elias Arnold

March 2023

Contents

1 Introduction

Spiking neural networks (SNN) on analog neuromorphic hardware are a promising approach to solve machine learning (ML) tasks with high energy efficiency. While training of SNNs on the BrainScaleS-2 (BSS-2) system has already been shown in [12], this only holds for SNNs that fit on a single chip. Most ML tasks however require network sizes that exceed the BSS-2 hardware resources of a single BSS-2 chip by far. With this internship we set the foundation to show that the BSS-2 system can be used in a multi-chip fashion for larger networks by means of partitioning to demonstrate the system's scalability. We approach this problem with the MNIST dataset [2] as a toy example with a topology that requires partitioning for hardware execution.

As a motivation for SNNs, this report covers an introduction to biological neurons and their mathematical description, the LIF model. Their implementation on the BSS-2 system together with an overview of the BSS-2's architecture will be given as well. We will discuss

learning in SNNs followed by a hands-on example with the MNIST dataset. After building a baseline model, an implementation with hardware constraints is shown and compared. It will be discussed how the model can be transferred to hardware and after revisiting and reflecting the results, possible further steps are mentioned in the outlook.

2 Methods

2.1 Biological Neurons

A neuron consists of a soma, the cell body, containing the nucleus, several dendrites, an axon, that is for some neurons covered in a myelin sheath, and axon terminals. The most important parts are illustrated in figure 1. It typically receives input in form of electrical potentials via

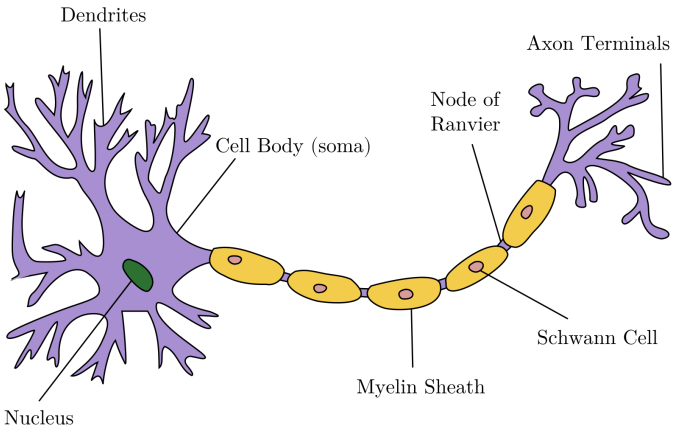

Figure 1: Sketch of a biological neuron. [1]

its dendrites that are then superimposed at the soma. If the potential at the soma exceeds a certain threshold ϑ, a spike called action potential is emitted, traveling along the axon and axon terminals to other neurons. After that, a short refractory period is entered where the neuron is unlikely to spike again. The neuron's membrane is leaky, meaning ions can be exchanged through it causing the membrane potential V_m to adjust back to a resting potential V_l. Neurons communicate their spike events through synapses: The axon terminal of a *presynaptic* neuron releases neurotransmitters upon an action potential, traversing to the *postsynaptic* neuron and therewith changes its membrane potential. The change in the membrane potential due to a presynapitc spike event is called postsynaptic potential (PSP) the height of which depends on the synaptic strength, meaning, how many neurotransmitters are released during the transmission. The ability to regulate synaptic strength over time is called plasticity and allows learning in neural networks.

2.2 Neuron Model

Interconnected biological neurons (neural networks) as in mammal's brains have proven themselves to be extremely capable of solving tasks like object recognition, scene classification and natural language processing, while only consuming very little energy due to their event based nature. Consequently, they build a set of desirable properties for novel computing devices that can solve problems in these categories (and beyond). The field of analog neuromorphic computing tries to adapt these attributes in electronical chips. The foundation is a mathematical description of a neuron, which for the leaky integrate-and-fire (LIF) model

is the following differential equation, modeling membrane dynamics $V_m(t)$ (the membrane voltage) according to:

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

$$z(t) = \sum_k \delta(t - t_k) \tag{2}$$

with the membrane time constant $\tau_{mem} = C_m/g_m$, the membrane capacitance C_m, leakage conductance g_m, the synaptic input current $I(t)$, membrane threshold ϑ and spike times t_k. The membrane of the neuron is modeled by a capacitor with capacitance C_m. We get the differential equation by differentiating the common capacitor equation $C_m V_m = Q$ (Q being the charge) with respect to time, where on the right side we can split the total current $\dot{Q}(t)$ into $-g_m(V_m - V_l)$, which describes the leaking through the membrane, and $I(t)$, which resembles the sum of incoming currents triggered by neurotransmitters at the connection to other presynaptic neurons. $z(t)$ is called a *spike train* and implements the firing of the neuron: If $V_m = \vartheta$, V_m is set to a reset potential V_r and a spike is emitted at time t_k.

The synaptic currents to neuron i due to presynaptic events at neurons j can be described by

$$\frac{\mathrm{d}}{\mathrm{d}t}I_i(t) = -\frac{1}{\tau_{syn}}I_i(t) + \sum_j \sum_k w_{ij}\delta(t - t_{jk}), \tag{3}$$

with τ_{syn} being the synaptic time constant, describing the time scale on which the current exponentially falls. The synaptic strengths between the presynaptic neurons j and neuron i are modeled by a multiplicative factor w_{ij} called synaptic weight. In total with spike events incoming at times t_{jk}, we can model $I_i(t)$ as follows:

$$I_i(t) = \sum_j \sum_k w_{ij} e^{-\frac{1}{\tau_{syn}}(t - t_{jk})}\theta(t - t_{jk}), \tag{4}$$

with the heaviside function $\theta(t)$. The non-spiking version of the LIF model is called leaky integrator (LI).

2.3 The BrainScaleS-2 System

The BSS-2 system [12] is a neuromorphic computing platform that emulates networks of spiking neurons in analog circuits. The circuits implement 512 AdEx neurons, which can be configured to behave like the models discussed above (LIF, LI). They are placed in two hemispheres with 256 neuron circuits each (see figure 2). Each hemisphere has two synapse matrices that consist of 128 synapse arrays (columns) and can receive inputs from 256 synapses, each synapse array connecting to one neuron. Events arriving in synapses trigger exponentially decaying currents into the respective neuron, matching the mathematical description in section 2.2. Equation 1 is implemented without the resetting term, which is replaced by a jump condition, that resets the membrane potential as soon as $V_m(t) \geq \vartheta$ which can be implemented using a voltage comparator. These membrane potentials can be sampled in parallel via the Columnar ADC (CADC), of which one exists on each hemisphere, and read out by the host computer. The spike events can also be recorded. The synaptic weights can be configured with 6 bit values. Also, up to 64 neurons can be connected to form larger neurons, allowing for more fan-in per neuron. As synaptic weights can either be row-wise inhibitory or excitatory, signed weights can be realized by using two hardware synapses, one inhibitory and one excitatory, but effectively halving the total fan-in. The neuron parameters

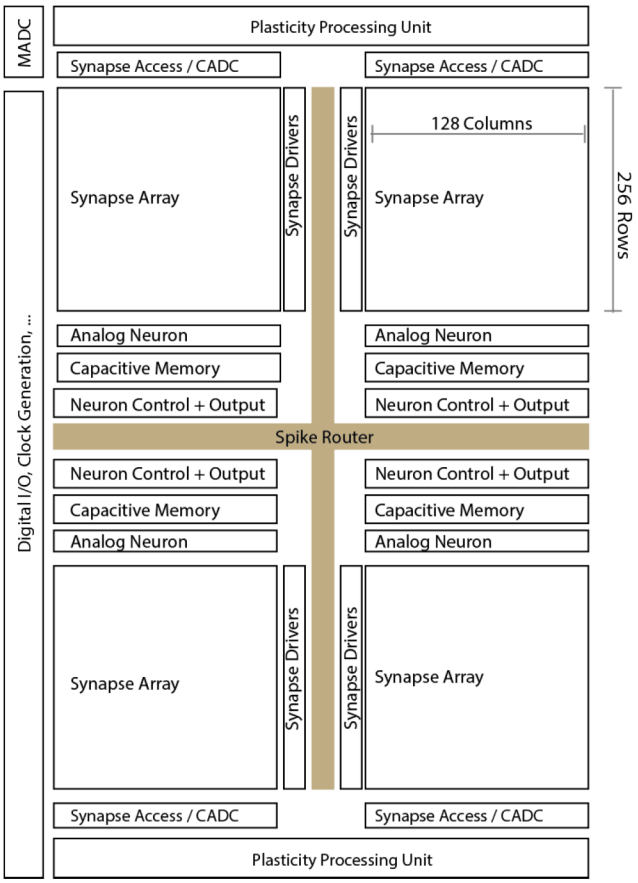

Figure 2: Floorplan of the BSS-2 chip. Image taken from [12].

are subject to calibration, including membrane and synaptic time constants. Inputs to the chip can be injected externally and routed internally to realize different network topologies. In comparison to the biological time domain, the BSS-2 system is accelerated by a factor of 10^3.

2.4 Spiking Neural Networks

In general, SNNs are neural networks that mimic natural neural networks. Typically, the artificial spiking neurons, based on some neuron model, are arranged into layers that are then interconnected. An example of a simple feed forward architecture is shown in figure 3. At neuron i in the input layer, input spike trains $z_i(t)$ of the form

$$z_i(t) = \sum_k \delta(t - t_{ik}) \tag{5}$$

are inserted into the network and propagate through it with each neuron evolving in time according to it's own dynamic. Typical tasks that need to be considered to set up an SNN to successfully solve a given task are: input encoding, output decoding and learning on SNNs.

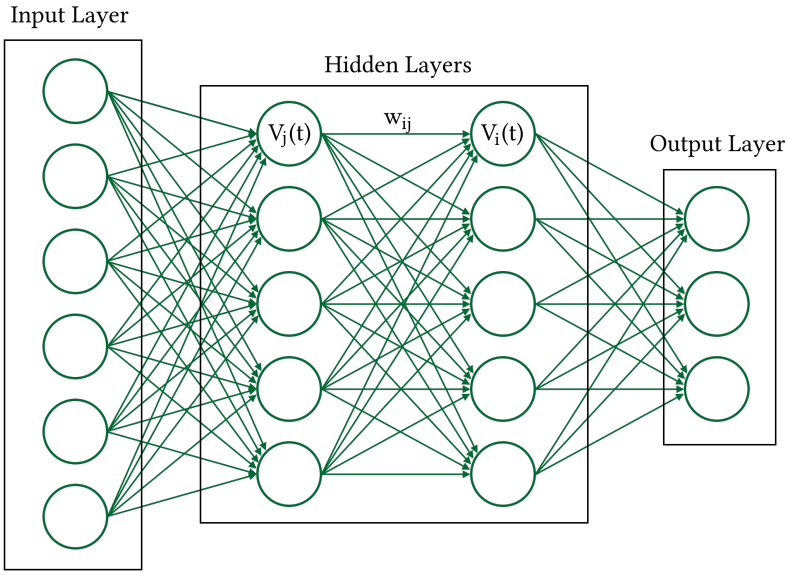

Figure 3: Sketch of a fully connected feed forward type network architecture for SNNs. The circles represent (artificial) spiking neurons, the arrows the connections and synapses between them.

2.4.1 Encoding and Decoding

For encoding, the input data has to be encoded into spike trains with a time axis such that the SNN can efficiently work with it. Once this sequence length is chosen, there are multiple ways for encoding. The method of constant current encoding injects a constant current onto the membrane of a LIF neuron, resulting in a constant firing rate. With Poisson encoding, the value is assigned an average firing rate with deviations according to Poisson statistics. Another method is the time to first spike (TTFS) method, where the spike time of the spike encodes the value. The decoding depends on the type of neurons that are used in the output layer. A reasonable choice is an LI neuron and using the maximum value of the membrane voltage trace as the output value (max-over-time decoding). Another possibility is to use a spiking output layer with a TTFS output-spike decoding.

2.4.2 Learning in SNNs

There are multiple approaches to learning and optimizing in SNNs [10, 14]. While there are methods involving biologically inspired learning rules, we will focus on gradient based learning in this report. The basis of gradient based learning is a loss function $\mathcal{L}(x, \mathbf{W})$, with x being the input data and $\mathbf{W}$ the parameters of the network, like the synaptic weights w_{ij}, describing the weight of the connection from neuron j to neuron i. $\mathcal{L}(x, \mathbf{W})$ measures how far the evaluation of the network is from the expected output. Then, the gradient of $\mathcal{L}$ with respect to a parameter $\mathbf{W}_{ij}$ is computed to adjust the parameters afterwards into the direction of steepest descent:

$$\mathbf{W}^{(n+1)} = \mathbf{W}^{(n)} - \kappa \sum_i \left. \frac{\partial \mathcal{L}(x, \mathbf{W})}{\partial \mathbf{W}_{ij}} \right|_{\mathbf{W}=\mathbf{W}^{(n)}} \mathbf{e}_{ij}, \tag{6}$$

where κ is the learning rate and $\mathbf{e}_{ij} = \delta_{ij} \cdot \mathbf{1}_{N \times N}$, $\mathbf{1}_{ij} = 1 \ \forall \ i, j$ for N^2 parameters. In repetition, this procedure is called gradient descent. Typically the gradient of a network with

respect to its parameters can be computed conveniently by applying the backpropagation algorithm. When applied to a large amount of data, the network parameters will minimize $\mathcal{L}(x, \mathbf{W})$, approaching the error of the desired output. How good this local minimum of loss is, e.g. with regards to unseen data, i.e., how good the model generalizes, depends on variety of factors: the network architecture, the degrees of freedom in comparison to the complexity of the task, the size of the training set and many more.

While this approach holds in its entirety for non spiking artificial neural networks (ANNs) when all applied functions are differentiable everywhere, i.e. linear functions or activation functions such as the sigmoid $\text{sig}(x) = (1 + e^{-x})^{-1}$, this differentiability is not given for non-differentiable activation functions and spiking neuron models such as the LIF (which, in the context of ANNs, can be understood as an activation function). Another issue that has to be addressed is the time dependency of the neurons.

The latter is resolved when viewing SNNs as recurrent neural networks (RNNs). RNNs are neural networks whose neuron states evolve to a set of recurrent dynamical equations, as equations (1) and (3) are. This can be shown clearer by formulating a numerically suitable approximation of these equations. We therefore introduce a time step Δt, in which the numerical integration of the neuron dynamics can be performed. With n representing the n-th time step, a suitable approximation is then given by (following [10]):

$$I_i[n+1] = \alpha I_i[n] + \sum_j w_{ij} Z_j[n], \quad \text{and} \tag{7}$$

$$V_i[n+1] = \beta V_i[n] + \frac{1}{g_m} I_i[n] - Z_i[n](\vartheta - V_r) \tag{8}$$

for current I_i and voltage V_i at neuron i with $\alpha \equiv \exp\left(-\frac{\Delta t_n}{\tau_{syn}}\right)$, $\beta \equiv \exp\left(-\frac{\Delta t_n}{\tau_{syn}}\right)$ and with $Z_i \equiv \theta(V_i[n] - \vartheta)$ - the last expression coming from integration of the δ-spikes in $z_i(t)$. Now, as the mapping onto an RNN is clear, we can solve the time dependencies issue by unrolling the network in time. Therefore we generate copies of the network for each time step, which share feedforward weights and can be additionally interconnected by recurrent weights. To this network, standard backpropagation applies, while this procedure is referred to as back propagation through time (BPTT).

The approach to resolving the non-differentiability issue is given with surrogate gradients. When differentiating $\mathcal{L}(x, \mathbf{W})$ for a weight w_{ij}, at some point, one will have to differentiate $Z_i[n]$ with respect to $V_i[n]$, involving the heaviside step function θ, the derivative of which is zero for all points except zero, where it is ill defined. This causes the training to effectively vanish. However, when replaced with a surrogate derivative that has a non-vanishing interval around zero (where the step occurs), backpropagation and BPTT can be applied without intrinsic problems. This method is proven to be very effective in training SNNs and also remarkably robust, even in the choice of the surrogate gradient itself [10]. Possible choices for surrogate gradients are the SuperSpike [6] (which is the derivative of a fast sigmoid function), the derivative of a standard sigmoid function and a piecewise linear function [3, 4]. They are shown in figure 4 and are discussed in detail in [10].

2.4.3 Software Framework for Training SNNs

Norse [9] is a deep learning Python library, extending PyTorch [8] with primitives for bio-inspired neural components such as an LIF and LI model as well as the encoding options discussed above (2.4.1). The input spike train is projected onto a discrete time grid of evenly spaced binary events on which the dynamics of the SNN are integrated numerically. When implementing neural networks in Python, PyTorch's autograd functionality allows for an

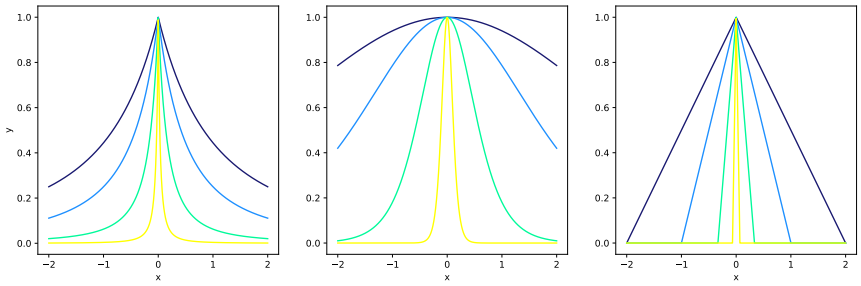

Figure 4: Plots of different surrogate gradients for the heaviside step function and different values of α. Left: Superspike (fast sigmoid derivative) $(1 + \alpha|x|)^{-2}$. Middle: Rescaled sigmoid derivative $4 \cdot \frac{\exp(-\alpha x)}{(1+\exp(-\alpha x))^2}$. Right: Piecewise linear function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

easy computation of the loss-gradient by keeping track of all operations in a computational graph. Norse also implements surrogate gradients for the LIF models, making use of the computational graph while resolving non-differentiability issues.

2.5 The MNIST Data Set

MNIST (Modified National Institute of Standards and Technology database) [2] is a widely known dataset and former benchmark for machine learning models. The data are 28×28 gray-scale images showing handwritten numbers (0 to 9) and their corresponding labels. MNIST consists of 60000 training examples and 10000 test examples.

3 Learning MNIST

On the software basis of Norse, we will implement an SNN and train it with the MNIST dataset. In this application we will use the constant current encoding for the inputs. Figure 5 shows how this encoding can look like for a sequence length of $T = 30$ time steps. As a baseline for further comparisons with hardware realizations on BSS-2, we will choose a topology of the network in accordance to the full capacity of the chip.

3.1 Network Topologies

Firstly, we will discuss, how the baseline model is chosen, making use of the full chip capacity if transferred to hardware realizations on BSS-2. As we want excitatory inputs as well as inhibitory inputs to the neurons, two rows of synapse arrays would have to be used, one for each input type in order to realize signed hardware weights. To keep the same fan-in size, we will have to create a larger neuron compartment by connecting two atomic neurons, sharing the same membrane potential, while only one of them emits spikes. With this constraint, one BSS-2 chip provides 256 input arrays to each of the 256 compartment neurons. Then, the maximum input size that fits the chip with no other issues is a $\lfloor\sqrt{256}\rfloor \times \lfloor\sqrt{256}\rfloor = 16 \times 16$ sized image. For one hidden layer and a readout layer of size 10 (for the 10 classes of the MNIST set), this leaves 246 neurons for the hidden layer. The baseline topology that exploits the maximum capacity of the chip is therefore: $16 \times 16 \to 246\text{LIF} \to 10\text{LI}$. The 10 readout neurons are chosen to be LI-types for decoding purposes (max-over-time). This has already been implemented on BSS-2 [11].

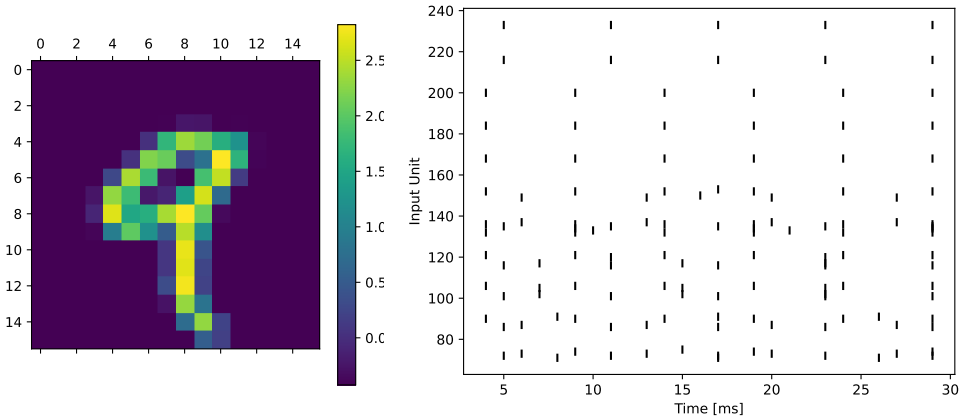

Figure 5: Left: Input image showing a 9 in 16 by 16 format. Right: Encoding of image to the left using the constant-current-LIF-encoder method by Norse with a sequence length of 30 time steps.

When adding additional hidden layers using neuron compartments of two neurons, chip-suitable sizes are $512/2 = 256$ hidden units as this does not require partitioning of layers and at the same time allows to efficiently exploit all fan in when the output is passed on to the next layer. The resulting topologies of multiple hidden layers of size 256 require a multi-chip setup. A partitioning of the network is then to evaluate each layer in consecutive hardware runs, storing the results of the previous hardware execution and feeding it as an input into the next.

The other expanding possibility is to increase the number of hidden units in a hidden layer. As the neurons are not interconnected within a layer, partitioning can be done by splitting the layer into parts containing 256 neurons or less. When connecting this oversized layer to the next layer, again with 256 neurons, we will have to partition again due to a fan-in larger than 256. Creating neuron compartments with more than two neurons for the next layer solves this issue as the fan-in can be adjusted to $n \cdot 128, n \in \{1,2,...,64\}$ by connecting n neurons per compartment, resulting in $\lceil 256/\lceil 512/n \rceil \rceil$ necessary layer partitions.

Consider, for example, the topology $22 \times 22 \rightarrow 256 \rightarrow 10$. Firstly, the fan in to the hidden layer is $22 \times 22 = 484 > 256$. While maintaining signed hardware weights, this means, that we will have to use compartments of $\lceil 484/128 \rceil = 4$ neurons in the hidden layer, resulting in $\lceil 256/\lceil 512/4 \rceil \rceil = 2$ partitions for the hidden layer. These two partitions have to be executed in two consecutive hardware runs. Their results however can be injected into the readout neurons all at once, resulting in a total of three partitions for this topology.

In hxtorch.snn [13], a machine learning based modeling framework for BSS-2, these partitions can be manually implemented for the specific use case via experiment instances that are defined before the execution on hardware. The discussed partitioning possibilities can therefore be directly implemented in hardware compatible software.

3.2 Implementing Weight Quantization and Saturation

As synaptic weights on the BSS-2 system can only be chosen in unsigned 6 bit (or signed 7 bit (counting 0 twice) with signed hardware weights (see above), this is an important hardware

constraint when considering the standard float value weights. When implementing into the model, a suitable range for the 7 bit quantization has to be found. When reviewing weight histograms of trained networks without weight quantization (see figure 6), a reasonable choice for this range is given by the interval $[-2.5, 2.5]$, minimizing the number of weights effected by saturation.

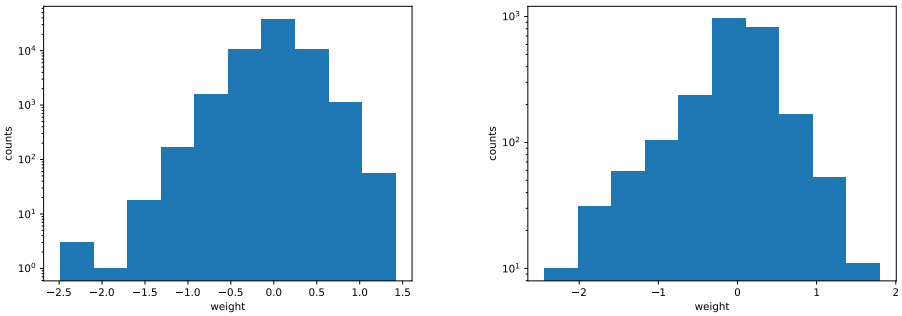

Figure 6: Histograms of weights for the two weight matrices of a baseline model ($16 \times 16 \to$ 246LIF$\to$ 10LI). Left: histogram for the weights between input and LIF-layer, right: histogram for the weights between LIF-layer and LI-layer.

For the implementation with Norse and PyTorch, it is useful to define a new function that extends from a torch-autograd-function [8]. While in the forward method, we just apply the rounded weights, the backward method has to implement the gradient for this function. As the rounded weights can be described by a step function, we will use a surrogate gradient for this application. When considering a linear surrogate gradient, the results match the case for non-quantized weights and therefore prove its effectiveness.

3.3 Results

Figure 7 shows how accuracies and losses evolve during the training of a network and thus show that one gets good results even though weight quantization and saturation is implemented. In total, the networks listed in table 1 were trained and give an overview on

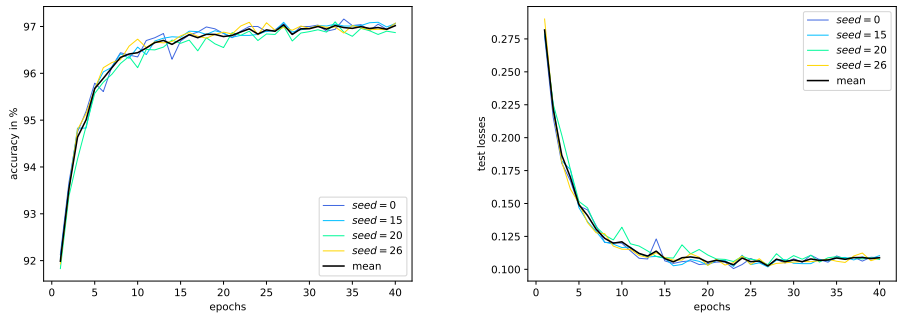

Figure 7: Accuracies and losses (negative log-likelihood of the log-softmax of the maximum voltages) of the test data set evolving with the number of epochs in training a $16 \times 16 \to$ 246LIF$\to$ 10LI network using weight quantization. Different seeds (0, 15, 20, 26) show different curves, the mean of all final accuracies is listed in table 1.

performance. It should be noted that the implementations are far from optimized and can be further improved in many ways, a few of which are pointed out in chapter 4.

model parameters				performance (in % acc.)	
Inp. Lay.	Hidd. Lay. 1	Hidd. Lay. 2	weight quant.	Acc. Tr.	Acc. Test
16×16	246 LIF	-	×	99.911 ± 0.007	96.94 ± 0.10
16×16	256 LIF	256 LIF	×	99.91 ± 0.08	97.12 ± 0.21
16×16	246 LIF	-	✓	99.866 ± 0.009	97.02 ± 0.09
16×16	256 LIF	256 LIF	✓	99.938 ± 0.008	97.16 ± 0.16
22×22	246 LIF	-	×	99.995	97.53 ± 0.10
22×22	256 LIF	-	×	99.995 ± 0.001	97.51 ± 0.08
22×22	256 LIF	256 LIF	×	99.95 ± 0.07	97.66 ± 0.16
22×22	246 LIF	-	✓	99.993 ± 0.003	97.47 ± 0.05
22×22	256 LIF	256 LIF	✓	99.985 ± 0.008	97.77 ± 0.07

Table 1: Overview of different topologies that were used to train an SNN in Norse with the MNIST data set. The accuracies are averaged over trainings with seeds 0, 15, 20 and 26, each after 40 epochs. Hyperparameters: Batch-Size: 100; Sequence Length: 30; Encoder: ConstantCurrentLIFEncoder; $\tau_{syn} = 6 \times 10^{-3}$; $\tau_{mem} = 5.7 \times 10^{-3}$; $\alpha = 100$; $dt = 0.001$; $\vartheta = 1$; learning rate: 1.5×10^{-3}; l.r.-decay: 0.03 each epoch.

4 Discussion and Outlook

In this report, we showed how training on BSS-2 can be approached by discussing network topologies that exploit the hardware resources of BSS-2 in a single-chip use case as well as a multi-single-chip fashion, the goal being comparisons to future hardware executions on BSS-2. Surrogate gradients were presented as an effective way to overcome differentiability issues and how unrolling RNNs shows that backpropagation (through time) can be used as a training method. With topologies that are suitable for partitioning and considering hardware resources, we implemented SNNs in Norse/PyTorch and trained them on the MNIST data set as a toy example for demonstration purposes, building a baseline for further comparisons with hardware executions on BSS-2. However, this baseline still leaves room for optimization, mainly with regards to hyperparameters. As table 1 shows, the instances with implemented weight quantization even tend to be better than the corresponding model without. That might be due to randomness or not optimized hyperparameters (e.g. learning rate and it's decay, number of epochs) as the fluctuations in performance with each epoch can be minimized with a better choice. Another optimization possibility is to modify the input data, generating more training data. One such modification would be to tilt the inut image by a small angle. Additionally, a spiking regularization can be implemented, suppressing unnatural and (for hardware executions) energy-inefficient high firing rates.

The partitioning of networks is of great significance for the scalability of the BSS-2 system in order to approach increasingly complex tasks. One such task is landscape classification of satellite images for which a data set for training already exists with the EuroSAT data set [5, 7]. Together with showing how partitioning can be approached and optimized for different tasks in a more general fashion, it would also be desirable to develop auto-partitioning algorithms for BSS-2.

References and Sources

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

[2] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[3] 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: *Proc Natl Acad Sci U S A.* 113(41) (2016), pp. 11441–11446. DOI: 10.1073/pnas.1604850113.

[4] Guillaume Bellec et al. "Long short-term memory and learning-to-learn in networks of spiking neurons". In: *Advances in neural information processing systems* 31 (2018).

[5] 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.

[6] Friedemann Zenke and Surya Ganguli. "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks". In: *Neural Computation* 30.6 (June 2018), pp. 1514–1541. ISSN: 0899-7667. DOI: 10.1162/neco_a_01086. eprint: https://direct.mit.edu/neco/article-pdf/30/6/1514/1039264/neco_a_01086.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01086.

[7] 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).

[8] 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.

[9] Christian Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.7. Oct. 2021. URL: https://github.com/norse/norse.

[10] 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.

[11] Benjamin Cramer et al. "Surrogate gradients for analog neuromorphic computing". In: *Proceedings of the National Academy of Sciences* 119.4 (2022), e2109194119.

[12] C. Pehle; S. Billaudelle; B. Cramer; J. Kaiser; K. Schreiber; Y. Stradmann; J. Weis; A. Leibfried; E.Müller; J. Schemmel. "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Front Neurosci.* 16:795876 (2022). DOI: 10.3389/fnins.2022.795876.

[13] Philipp Spilger et al. "hxtorch. snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2". In: *arXiv preprint arXiv:2212.12210* (2022).

[14] Paweł Pietrzak et al. "Overview of Spiking Neural Network Learning Approaches and Their Computational Complexities". In: *Sensors* 23.6 (2023). ISSN: 1424-8220. DOI: 10.3390/s23063037. URL: https://www.mdpi.com/1424-8220/23/6/3037.