

Department of Physics and Astronomy
University of Heidelberg

Master Thesis in Physics
submitted by

Simon Jonscher

born in Erlangen (Germany)

2025

Gradient-Based Learning of Neuron Parameters in Spiking Neural Networks on Neuromorphic Hardware

This Master Thesis has been carried out by Simon Jonscher at the
Kirchhoff Institute for Physics in Heidelberg
under the supervision of
Prof. Dr. Johannes Schemmel

Abstract

Inspired by nature, spiking neural networks (SNNs) provide a fault-tolerant and energy-efficient solution for signal processing. Since the internal state of each neuron in SNNs evolves over time, they are well-suited for processing temporal information, making them ideal for tasks like speech-recognition. Apart from only training synaptic weights between neurons, additional training parameters that influence the temporal dynamics of the neuron has been proven beneficial. The mixed-signal neuromorphic system BrainScales-2 (BSS-2) enables a fast and energy-efficient emulation of the behavior of SNNs on analog circuits, while also being highly configurable. This work implements an interface for gradient-based training of neuron parameters on the neuromorphic system. To demonstrate that neuron parameters can be trained on BSS-2, the hardware parameters are characterized to configure the synaptic and membrane time constant, respectively. Several simulation experiments, analyzing the effects of model simplification that are required for execution on BSS-2, simultaneously highlight the performance improvement achieved when these time constants are trained in addition to the weights. Applying these simplifications, a feedforward and a recurrent SNN are trained on the Spiking Heidelberg Digits (SHD) dataset on the neuromorphic substrate with time constants included in the set of learnable parameters.

Zusammenfassung

Die von der Natur inspirierten spiking neural networks (SNNs) bieten eine fehlertolerante und energieeffiziente Lösung für die Signalverarbeitung. Da sich der interne Zustand jedes Neurons in SNNs im zeitlich dynamisch entwickelt, sind sie gut für die Verarbeitung zeitlicher Informationen geeignet, was sie ideal für Aufgaben wie Spracherkennung macht. Es hat sich als vorteilhaft erwiesen, nicht nur synaptische Gewichte zwischen den Neuronen zu trainieren, sondern auch Parameter, die die zeitliche Dynamik des Neurons beeinflussen. Das neuromorphe System BrainScales-2 (BSS-2) ermöglicht eine schnelle und energieeffiziente Emulation des Verhaltens von SNNs auf analogen Schaltkreisen und ist gleichzeitig hoch konfigurierbar. In dieser Arbeit wird eine Schnittstelle für das gradientenbasierte Training von Neuronenparametern auf dem neuromorphen System implementiert. Durch die Charakterisierung der Hardware-Parameter für die Konfiguration der synaptischen und der Membran-Zeitkonstante wird gezeigt, dass diese Parameter auf BSS-2 trainiert werden können. Mehrere Simulationsexperimente, in denen die Auswirkungen von Modellvereinfachungen, die für die Ausführung auf BSS-2 erforderlich sind, unterstreichen die Leistungsverbesserungen, die erzielt werden, wenn beide Zeitkonstanten trainiert werden. Unter Anwendung dieser Vereinfachungen werden ein feedforward und ein rekurrentes SNN auf dem Spiking Heidelberg Digits (SHD) Datensatz auf dem neuromorphen Substrat unter Einbeziehung beider Zeitkonstanten trainiert.

Contents

1	Introduction	1
2	Theoretical Background	4
2.1	Biological Neuron	4
2.2	Models of Biological Neurons	7
2.3	Gradient-Based Learning in Spiking Neural Networks	10
2.4	The BrainScaleS-2 System	14
2.5	Gradient-Based Training on BrainScaleS-2	18
2.6	Spiking Heidelberg Digits Dataset	22
2.7	Related Work	23
3	Learning Neuron Parameters	25
3.1	Implementation for Simulation	25
3.1.1	Training on Neuron Trace in Software	25
3.2	From Simulation to Emulation on BrainScaleS-2	28
3.2.1	Characterization of the Membrane Time Constant	30
3.2.2	Characterization of the Synaptic Time Constant	37
4	Learning Time Constants on SHD	41
4.1	Learning Time Constants in Simulation	41
4.2	Simulation Tests for Hardware Emulation	43
4.3	Emulation on the SHD Dataset	51
4.3.1	Shifting Leak Potential	52
4.3.2	CapMem Crosstalk	53
4.3.3	Saturation of the Readout Neurons	54
4.3.4	Results on BrainScaleS-2	56
5	Discussion	63
6	Outlook	67
	Appendices	80
A	Hardware Parameters	80
B	Software State	80

1 Introduction

Nature has always been a source of inspiration for scientific and technological progress. Many breakthroughs across disciplines have emerged from observing and imitating biological systems, which are often the result of millions of years of optimization through evolution. One of the most fascinating and least understood examples of such a system is the human brain. A highly complex, yet remarkably energy-efficient machine capable of solving difficult tasks like perception, learning, and decision-making.

Motivated by the brain’s capabilities, researchers have developed a wide range of biologically-inspired learning algorithms. In recent years, artificial neural networks (ANNs) achieved remarkable success in tasks such as speech and image recognition (LeCun et al., 2015), reinforcement learning (Silver et al., 2017) and natural language processing (Radford et al., 2018). ANNs rely on perceptrons, a basic abstraction of a neuron. Inspired by the structure of the brain, ANNs are built by connecting these perceptrons to build larger networks. As ANNs process information in a dense and synchronous way, these advances have only been possible with the increase of computational resources and come with a large energy consumption.

Spiking neural networks (SNNs) not only mimic the structure of brains, but also replicate their spiking behavior. Base on the biological neuron, neurons in SNNs only transmit information when their internal state, the membrane potential, crosses a threshold. This makes SNNs inherently more biologically plausible and much more energy-efficient, just like the brain, which processes massive amounts of information using very little energy.

To fully exploit the capabilities of SNNs, neuromorphic hardware provides an ideal platform. Several research collaborations and companies build neuromorphic systems, such as Intel with Loihi (Davies et al., 2018) or SpiNNaker (Furber et al., 2014). The BrainScales-2 (BSS-2) system is a mixed-signal neuromorphic chip developed at Heidelberg University (Pehle et al., 2022). While analog circuits implement the dynamics of adaptive exponential leaky integrate and fire (AdEx) neurons (Brette and Gerstner, 2005) in continuous

time, spike events are handled digitally. This enables to emulate the dynamics of biological neuron and synapse models with high energy efficiency and real-time processing capabilities. As it is highly configurable, BSS-2 can be used for a variety of applications, such as emulation of complex neuron dynamics (Kaiser et al., 2022; Billaudelle et al., 2022) or biologically inspired learning (Atoui et al., 2024). The possibility of gradient-based learning of neural networks on BSS-2 has been demonstrated for various applications (Cramer et al., 2022; Arnold et al., 2023).

An interesting application of SNNs is the Spiking Heidelberg Digits (SHD) dataset (Cramer et al., 2020), which is specifically designed for spike-driven speech recognition. The SHD dataset transforms spoken digits into spike trains, mimicking the auditory processing of the human ear. In this analogy, the dataset represents the information processed in the ear, while the SNN acts as the brain, tasked with recognizing the spoken digits. This biologically-inspired design highlights the potential of SNNs for solving real-world temporal tasks.

A further enhancement comes from heterogeneous SNNs, in which neurons exhibit diverse dynamics, such as varying neuron parameters or delays between neurons. This heterogeneity increases the adaptability of SNNs, enabling them to better model complex temporal patterns while also improving their robustness (Hammouamri et al., 2023; Golmohammadi and Tetzlaff, 2024). Perez-Nieves et al. (2021) demonstrate the benefits of learning both the synaptic and membrane time constants across different datasets. In particular, for the SHD task, they report a substantial performance improvement and increased robustness to input speedup when training time constants in addition to synaptic weights.

First, this thesis extends the learning framework `hxtorch` (Spilger et al., 2023), which interfaces the BSS-2 to enable gradient-based training of neuron parameters on the neuromorphic system. Next the hardware parameters necessary for configuring the synaptic and the membrane time constant are characterized, enabling a demonstration of neuron parameter learning on BSS-2. After reproducing simulation results presented in Perez-Nieves et al. (2021), different

simulation experiments investigate the effects of model simplification, required for execution on BSS-2. Under these simplifications, the benefits of learning time constants in addition to synaptic weights are then demonstrated on BSS-2 by training a feedforward and a recurrent SNN deployed on the neuromorphic substrate on SHD.

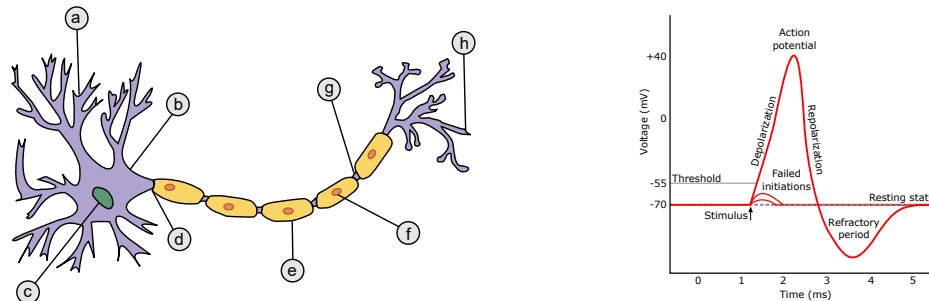
2 Theoretical Background

This section provides a brief overview of the related theoretical background as well as the neuromorphic system BSS-2 that is being used for experiments. First, the biological neuron and its fundamental properties are described. This is followed by an introduction to mathematical neuron models, an overview of (spiking) neural networks, and their gradient-based training methods. Finally, the neuromorphic system, BSS-2 used in this work is presented.

2.1 Biological Neuron

The biological neuron is the fundamental building block of the nervous system, forming complex networks that process and transmit information via synapses. Understanding its structure and function provides the foundation for mathematical models that abstract its behavior, enabling the development of spiking neural networks and neuromorphic systems. A biological neuron consists of three main parts: the soma, dendrites, and an axon. A schematic of the neuron structure can be seen in Figure 2.1a. The soma, or cell body, contains essential cellular components such as the nucleus and mitochondria, which are responsible for the neuron’s vital functions. The dendrites act as receivers, collecting and integrating signals from upstream neurons. Finally, extending from the soma, the axon serves as the neuron’s output pathway, transmitting outgoing electrical signals to other connected neurons. As for all other cell types, the neuron’s interior is separated from the surroundings by a membrane. The membrane mainly consists of a semi-permeable lipid bilayer, which means that uncharged or small polar molecules are able to pass. Large charge carriers, such as sodium or potassium ions cannot cross the layer at any location. Therefore, the membrane effectively functions as an insulator, giving rise to a potential difference across it, referred to as the membrane potential. As neurons are in comparison to other cells, excitable, the membrane has additional embedded protein structures, that are electrically active. These protein structures facilitate two processes that enable the passage of particles that would otherwise be unable to cross. First, ion pumps can transfer specific ions across the lipid bilayer. For example, the sodium-potassium pump exchanges three sodium ions from the interior with two potassium ions from the

exterior of the cell. This leads to gradients in the corresponding ion concentrations, which in turn guides the second process: diffusion across passive ion channels. In both processes charge carriers cross the membrane, generating an electrical potential across the membrane. The resting potential is determined by the dynamic equilibrium between the diffusion-guided flow through passive ion channels and the active ion transport through the membrane by ion pumps. (Gerstner et al., 2014)



(a) Schematics of a neuron. From (US-Federal, 2006)

(b) Action potential.
From (Iberri, 2007)

Figure 2.1: (a) The schematics of a neuron with “a” being the dendrites, “b” representing the representing the soma including the cell nucleus “c”. The axon begins at “d” and spreads out to axon terminals “h” at the end. The axon is covered by a myelin sheath “e” of the Schwann cell “f”. Each small part where the axon is not covered by myelin is called a node of Ranvier “g”. (b) In case a current stimulus depolarizes the membrane potential, causing it to pass a certain threshold, an action potential is generated. Typically, this is followed by a refractory period, where the cell is not excitable.

Action Potentials Apart from ion pumps and passive ion channels, neurons also feature voltage-gated ion channels, which are triggered by the membrane potential. If a high current stimulus reaches the membrane and causes the membrane potential to exceed a threshold, these channels are responsible for the membrane potential to further increase. Through this increasing potential neuron quickly depolarizes and the membrane potential shoots up, followed by a similarly fast decline often overshooting the resting potential. Afterwards the membrane potential depolarizes back to its resting potential. This voltage trajectory is termed an action potential or spike. The trajectory of an action

potential is depicted in Figure 2.1b. Action potentials usually occur in the initial segment of the axon (Coombs et al., 1957), from where the signal is then distributed to synapses, which form the connections between neurons. Passing the signal to neighboring neurons through synapses is how inter-neuronal information exchange is generated. In case the current stimulus on the neuron is not strong enough, the membrane potential decays back to its resting potential, which is referred to as failed initiation in Figure 2.1b.

Synapses The end of the axon branches out into a tree-shaped structure to connect with dendrites of neighboring neurons. Synapses represent the connection points between these pre- and postsynaptic neurons. Most synapses rely on neurotransmitters, chemical messenger molecules, that are stored in synaptic vesicles located in specialized swellings in the presynaptic axon, called synaptic boutons (Kandel et al., 2021). The process of chemical signal transmission across the synapse is shown in Figure 2.2. When a presynaptic action potential arrives at the synapse, voltage-gated calcium channels allow for an influx of calcium ions (A). The resulting high calcium concentration in the presynaptic axon, causes vesicles to fuse with the presynaptic cell membrane, releasing neurotransmitters into the synaptic cleft, a small 20-40 nm wide gap between pre- and postsynaptic neuron (B). The released neurotransmitters temporally bind to receptor molecules in the post-synaptic neuron, opening chemically gated channels. These channels allow for ions to pass the membrane and therefore a change of the post-synaptic membrane potential (C). After the neurotransmitters break loose from the receptor, they are either reabsorbed by the presynaptic neuron, where they get repackaged back into vesicles, resetting the synapse for the next signal or they are removed from the synaptic cleft by enzymatic degradation. As the amount of neurotransmitters is finite and the recycling process is comparatively slow, the ion current decreases over time. It will be shown later that this decrease is often modeled by an exponential decay.

Generally speaking, synapses forward the action potential of a presynaptic neuron to the postsynaptic neuron, by triggering a postsynaptic potential (PSP). Depending on the neurotransmitters and corresponding receptors of the synapse, the post-synaptic potentials can be excitatory or inhibitory. Excitatory PSPs depolarize the postsynaptic membrane potential, increasing the

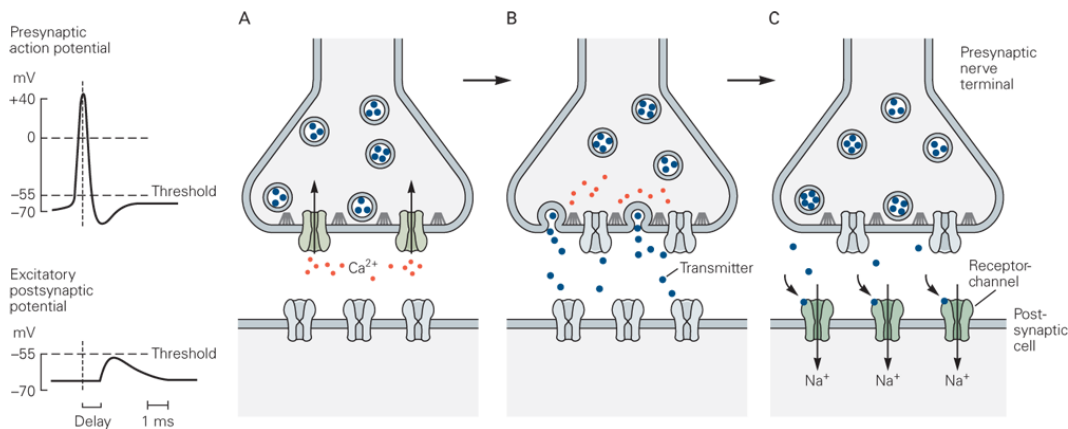


Figure 2.2: Process behind a signal transmission of a chemical synapse.
 Figure taken from Kandel et al. (2021). Copyright © McGraw-Hill Education. Reprinted with permission.

likelihood of an action potential in the postsynaptic neuron. An example of an excitatory postsynaptic output can be seen in the left bottom corner of Figure 2.2. In contrast, inhibitory PSPs hyperpolarize the post-synaptic membrane potential, thus decreasing the likelihood of an action potential. The height of the PSP depends on the synaptic strength, which is mainly determined by factors such as the probability of neurotransmitter release and the strength of the post-synaptic receptor response.

2.2 Models of Biological Neurons

The aim of a neuron models is to abstract biological features of neurons in mathematical form. There is a lot of theory on how the brain encodes information. A simple approach is to encode information in the rate of incoming spikes. For example, early work showed that the spike rate of stretch receptor neurons is related to the force with which the muscle is being stretched (Adrian and Zotterman, 1926). These observations introduced the concept of rate-based neuron models. However, to measure a rate, averaging over a time interval has to be done which means an instant reaction to changes is not possible. More recent research suggests that the information can also be encoded in the precise timing of spikes (Theunissen and Miller, 1995) leading to the concept of spiking neurons models.

Leaky Integrate and Fire Model There exists a spectrum of spiking neuron models with different complexities. A simplification that most models apply is to use point-like neurons. This means that the membrane potential is homogeneous across the whole neuron, so that for example current stimuli at a dendrite of the neuron has an instant effect on the membrane potential near the axon. A famous example, the Hodgkin-Huxley model (Hodgkin and Huxley, 1952) therefore describes the membrane as a single capacitor, where the voltage-gated sodium and potassium ion channels are represented as individual conductances, g_{Na} and g_K , while all other contributions are represented by a single leak conductance g_l . The membrane voltage v is described by

$$C_m \frac{dv}{dt} = -g_l(v - v_l) - g_{Na}(v - v_{Na}) - g_K(v - v_K), \quad (2.1)$$

where C_m is the capacitance of the membrane, v_{Na} , v_K and v_l are the reversal potential corresponding to each contribution respectively. With each conductance having its own differential equation to describe the voltage-dependent opening and closing of the ion channels, the Hodgkin-Huxley model is able to replicate the dynamics of an action potential as well as sub-threshold dynamics. This work uses the simpler leaky integrate and fire (LIF) model to describe the dynamics of a neuron, which was first introduced in Lapicque (1907) and later renamed. Additionally, the LIF model implements only a single passive leak term to pull the membrane potential to its resting potential. In the LIF model the resting potential is referred to as leak potential v_l . Assuming a generalized synaptic input I_{syn} for now, the membrane potential v then follows

$$C_m \frac{dv}{dt} = -g_l(v - v_l) + I_{syn}. \quad (2.2)$$

Instead of using electrical components, simulations often introduce the membrane time constant $\tau_m = \frac{C_m}{g_l}$ to the steepness of the exponential decay and assume the leak conductance g_l to be constant, leading to equation

$$\tau_m \frac{dv}{dt} = -(v - v_l) + \tilde{I}_{syn}, \quad (2.3)$$

where $\tilde{I}_{syn} = \frac{I_{syn}}{g_l}$ represents a voltage. Equation (2.3) only describes the leaky part and therefore a leaky integrator (LI) neuron. Although recent research suggests that the shape of mammalian spikes varies between neurons (Bean,

2007), the model considers spikes to be binary events, therefore saving compute power by not calculating the exact shape of an action potential. Whenever the membrane voltage of a LIF neuron crosses the threshold ϑ , it emits an output spike

$$z(t) = \begin{cases} 1, & \text{if } v(t) > \vartheta. \\ 0, & \text{otherwise.} \end{cases} \quad (2.4)$$

After the spike, the membrane voltage is immediately reset to the reset potential v_r . Until now, the description has been limited to the dynamics of the membrane potential only, while the input current has been assumed to be an arbitrary function of time.

Synaptic Input Each neuron is exposed to the inputs of its presynaptic neurons. As described earlier, especially chemical synapses modulate the incoming action potentials. Again, there exists a wide range of models to describe the presynaptic spike propagation by the synapse. The approach used in this thesis uses a single kernel $\kappa(t)$ to represent all synaptic dynamics and weights w_i to model the synaptic strength between the postsynaptic neuron i and presynaptic neurons j :

$$s_i(t) = \sum_j w_j z_j(t) * \kappa(t) \quad (2.5)$$

Implementing an exponential kernel $\kappa(t) \propto e^{-t/\tau_s}$, to capture the closing behavior of ion channels and the concept of current-based synapses $I_{\text{syn}} \equiv s(t)$, the input current on the neuron can be modeled similarly to the membrane:

$$\tau_s \frac{dI}{dt} = -I \sum_j w_j z_j(t), \quad (2.6)$$

where the synaptic time constant τ_s represents the time constant of the closing ion channels.

Discretization of Leaky Integrate and Fire Neuron For numerical simulations, the model described previously needs to be discretized in time. Using a small simulation time step δt , the differential equation (2.6) for the current of a postsynaptic neuron i can be well approximated by

$$I_i[n+1] = \alpha I_i[n] + \sum_j w_{ij} z_j[n], \quad (2.7)$$

with $\alpha = e^{-\frac{\delta t}{\tau_s}}$. Here, the sum runs over all spike trains of presynaptic neurons j . Note that the n -th discrete time step is denoted by the square bracket $[n]$. Having discrete input currents on the membrane, the membrane dynamics can now be expressed as

$$v_i[n+1] = \beta(v_i[n] - v_l) + v_l + (1 - \beta)I_i[n] - (\vartheta - v_r)z_i[n], \quad (2.8)$$

with $\beta = e^{-\frac{\delta t}{\tau_m}}$.

2.3 Gradient-Based Learning in Spiking Neural Networks

Up to this point, the theoretical description was limited to one neuron or at most the interconnection between two neurons. This section provides an overview of the extension to larger networks and the frameworks used to train them. Neural networks, in general, consist of a collection of interconnected units, each processing inputs received from all afferent neurons by applying a function φ . Following this line of thought, a network with L layers is effectively a mapping Φ that maps an input vector $\mathbf{x} \in \mathbb{R}^d$ to an output vector $\mathbf{y} \in \mathbb{R}^n$:

$$\mathbf{y} = \Phi(\mathbf{x}) = (\Phi_L \circ \dots \circ \Phi_1)(\mathbf{x}), \quad (2.9)$$

where Φ_i is the vector of functions φ that are applied by each input unit. All layers that do not directly interface with the input or output data are referred to as hidden layers. In standard ANNs, the function φ typically involves a weighted summation over all its inputs x_k with the addition of a bias b , followed by a non-linear activation function σ :

$$\varphi_{l,i}(x) = \sigma \left(b_{l,i} + \sum_k w_{i,k} x_k \right), \quad (2.10)$$

where the subscript l, i stands for an arbitrary neuron i in layer l of the network. The weights $w_{i,k}$ and the bias b_i are the learnable parameters corresponding to this neuron. In biological terms, each term of the summation can be seen

as one synapse, while the activation function can be understood as the neuron processing the inputs to an output. A common architecture used is a feed-forward network, where information flows in a single direction from input to output. The mapping of layer l can then be easily represented by a matrix multiplication followed by the activation:

$$\Phi_l(\mathbf{x}) = \sigma(\mathbf{W}_l \mathbf{x} + \mathbf{b}_l) \quad (2.11)$$

An example of a feedforward network with a single hidden layer can be seen in Figure 2.3.

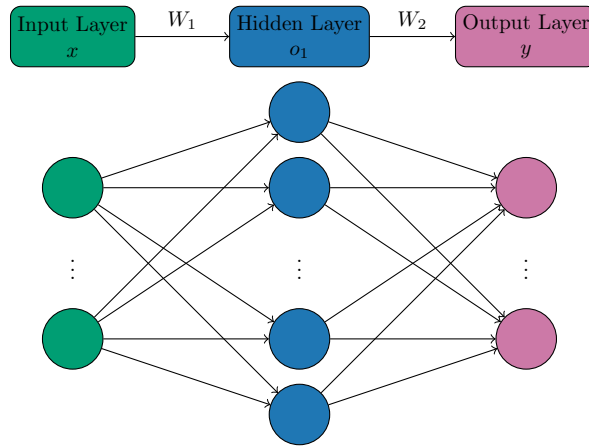


Figure 2.3: Schematic of a fully connected single layer feedforward architecture. The inputs \mathbf{x} are weighted and summed over using the weight matrix \mathbf{W}_1 . After applying the activation function, the outputs of the hidden layer are forwarded to the output layer using the weights \mathbf{W}_2 , where again an application function is applied.

Spiking Neural Networks are Recurrent Neural Networks To model sequential dependencies and dynamic behavior, recurrent neural networks (RNNs) allow information to be preserved over time. A general definition of RNNs involves that the network state depends not only on the external input but also on its previous state in time. In the common understanding of RNNs, this dependency is introduced by explicit recurrent connections, meaning that the output of a hidden layer is additionally connected to itself or a previous layer.

Spiking neural networks however use stateful neurons and synapses with internal dynamics, so that the state of the network always depends on the previous point in time, even when using a simple feedforward architecture (Neftci et al., 2019). This can be seen by looking at the computational graph of an SNN in discrete time in Figure 2.4.

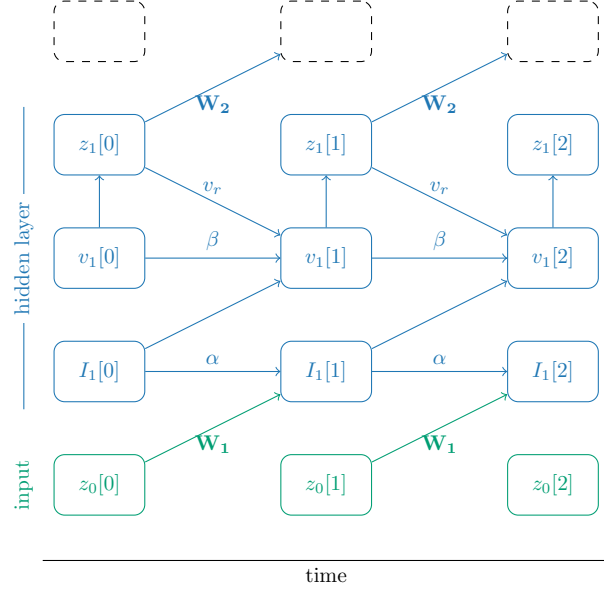


Figure 2.4: Computational graph of an SNN in discrete time. The input spikes z_0 are propagated through the network from bottom to top. The synaptic temporal updates of the synaptic current I and membrane potential v are enrolled on the horizontal axis for two time steps. The topmost layer, represented by the dashed boxes can either be an additional hidden layer or a readout layer. The latter is categorized as a single hidden layer spiking neural network.

Training Spiking Neural Networks From the previous paragraph it can be concluded that SNNs are only a special case of RNNs. For this reason, many training methods developed for classical RNNs can be used for training SNNs. In general, training networks requires two main components. The first component is an objective function, that evaluates the network’s performance. There exist many different objective functions for different sets of tasks. This work mainly focuses on a classification task, where the network needs to predict

the correct class for a certain input. For this task, the Cross-Entropy-Loss

$$\mathcal{L}_{CE} = \sum_{c=1}^C \log \frac{\exp(o_{L,c})}{\sum_{i=1}^C \exp(o_{L,i})} y_c \quad (2.12)$$

is typically used (Hastie, 2009). In this equation C represents the number of classes, $o_{L,c}$ is the output of the readout layer of the network for the class c and $y_c \in \{0; 1\}$ the one-hot encoded target vector, which is 1 for the correct class and 0 otherwise. The second component needed for training a neural network is an update mechanism, that updates the network's parameters to optimize the objective function. Since a loss function is used to evaluate the network's performance, this is a minimization problem and the simple but very powerful mechanism of gradient descent

$$\theta_{new} = \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta} \quad (2.13)$$

can be used. θ represents an arbitrary parameter of the network and η the learning rate, a hyperparameter to define the step size of the gradient descent. This thesis uses the Adam optimizer (Kingma and Ba, 2014), which is an adaptive version of gradient descent. In neural networks, many parameters do not directly influence the loss. For these parameters the chain rule is applied to calculate the gradient. An efficient calculation is possible by using the backpropagation (BP) algorithm (Rumelhart et al., 1986). The algorithm starts with the gradient calculation at the output of the neural network, since these gradients are required for updates in the previous layer. For RNNs, this can be done with the same method, once the recurrence is unrolled, as seen in Figure 2.4. The application of BP on the unrolled network is termed backpropagation through time (BPTT) (Werbos, 1990). One major challenge when training SNNs is that spikes are non-differentiable and therefore gradient descent can not be used without further modifications. There exist several ideas to overcome this. Inspired by nature, an idea is to use local learning rules for hidden units. Wunderlich and Pehle (2021) derived an algorithm, called EventProp, to compute exact gradients on spike times. For specific model configurations, where $\tau_m = \tau_s$ or $\tau_m = 2\tau_s$, there even exist analytical solutions for the direct calculation of the gradients in a time to first spike setting (Göltz et al., 2021). However, this work uses the concept of surrogate

gradients. When using surrogate gradients, each derivative of a spiking non-linearity is replaced by a derivative of a continuously differentiable function. Any monotonic function with a sharp increase that peaks at a threshold can be used since the success of the method is not heavily dependent on the surrogate derivative that is employed (Neftci et al., 2019). Zenke and Ganguli (2018) replaced the derivative of the spikes z with respect to the membrane potential v by the derivative of the negative of the fast sigmoid

$$\frac{dz}{dv} = (1 + \rho|v - \vartheta|)^{-2}, \quad (2.14)$$

where ρ describes the steepness of the surrogate gradient and ϑ the spiking threshold. The huge benefit that surrogate gradients offer is that only the derivative of the spike is replaced. Therefore, standard BP and BPTT algorithms, already implemented in standard libraries such as PyTorch (Paszke et al., 2017), can be used and the implementation is straightforward.

2.4 The BrainScaleS-2 System

The BSS-2 system is a mixed-signal neuromorphic chip designed for a variety of applications, such as faithful emulation of complex neuron dynamics (Kaiser et al., 2022; Billaudelle et al., 2022), biologically inspired learning (Atoui et al., 2024) or gradient-based learning of neural networks (Cramer et al., 2022; Arnold et al., 2023). The system is mixed-signal in the sense that while analog circuits emulate the dynamics of neurons and synapses in continuous time, spike events are handled digitally. Compared to the biological time domain, the model dynamics evolve at a 1000-fold accelerated time scale, reflecting the characteristic time constants imposed by the semiconductor substrate. This section will only provide an overview of the parts of the system that are relevant for this thesis. Further insights regarding the hardware can be found in Billaudelle (2022); Pehle et al. (2022); Schemmel et al. (2021). The essential software components required to run experiments on BSS-2 are presented in detail in Müller et al. (2022).

An overview of a mobile BSS-2 can be seen in 2.5. The heart of the system is the neuromorphic chip, which is divided in four quadrants. Each quadrant contains 128 highly configurable neuron circuits with a corresponding synapse array,

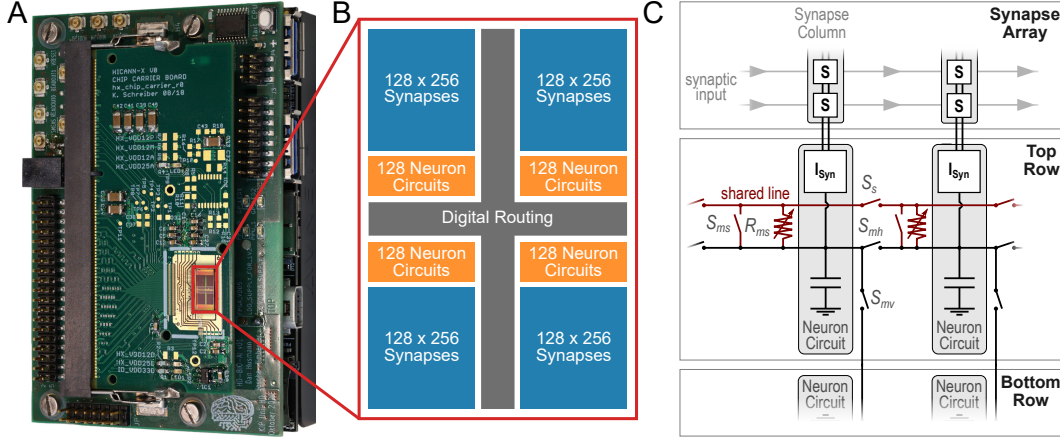


Figure 2.5: Overview of BrainScaleS-2 system. **A:** Neuromorphic chip on a carrier board. **B:** Abstract schematic of the chip design. **C:** Simple schematic of individual neuron circuits and connections. Figure kindly provided by Jakob Kaiser from (Kaiser et al., 2022).

consisting of 128 columns and 256 rows that project spikes column-wise onto the neurons. Since each row can be configured as either inhibitory or excitatory, this enables up to 128 signed inputs on a single neuron circuit. The depicted digital routing routes and injects the digital events, either external inputs or spikes of on-chip neurons, into the rows of the synaptic array. Two columnar analog to digital converters (CADCs), each responsible for one hemisphere, enable a parallel readout of data from the neurons or synapses. The CADC has an 8 bit value resolution and a measuring frequency of 1.6 MHz. The membrane analog to digital converter (MADC) can read data of two specific neuron circuit with 10 bit precision and a higher temporal resolution. Not represented in the abstract schematic of the chip, there are two single instruction, multiple data (SIMD) processors with 16 KiB of data and instruction memory each, located at the top and bottom of the chip. These processors provide a way to perform calculations and handle reconfiguration of the synapse array as well as other on-chip entities during runtime. Furthermore, they can be used to record on-chip observables, such as periodic CADC measurements, to external memory. The communication between host computer and the BSS-2 chip is handled by a field programmable gate array (FPGA) which is also responsible for the runtime control of an experiment. The FPGA allows for a cycle-accurate experiment control, ensuring precise timing for on-chip experiments. It is furthermore equipped with additional memory buffers for arbitrary data from

the host or the BSS-2 chip. Additionally, the two SIMD processors can also access dynamic random access memory (DRAM) connected to the FPGA for larger storage space. As can be seen in later sections this proves especially beneficial when conducting longer experiments that require more storage for measurements of the membrane potentials.

Neuron Circuit A single neuron circuit on BSS-2 employs AdEx neuron model (Brette and Gerstner, 2005). However, the circuit can be configured digitally so that the implemented model is reduced to the LIF model (Section 2.2), which is used in this thesis. The parts of the neuron circuit relevant for the LIF model are shown in a block schematic in Figure 2.6.

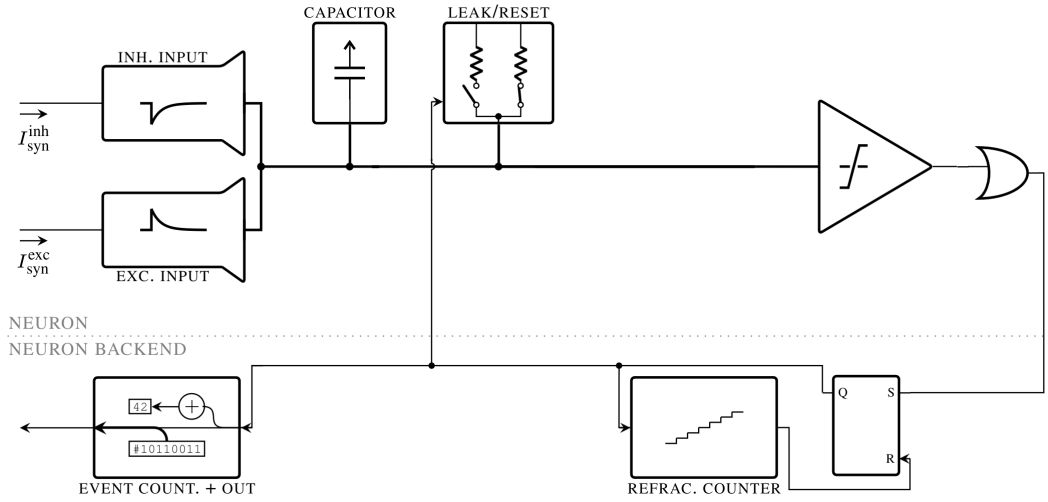


Figure 2.6: Block schematic of a neuron circuit implementing the LIF model. Figure adapted from Billaudelle et al. (2022)

The capacitor is the electric counterpart to the membrane of the neuron and gets charged by the incoming synaptic current I_{syn} . A configurable switch, that is set when emulating the LIF model, connects the capacitor via a resistor to the leak or reset potential. The leak behavior of a neuron is implemented using an operational transconductance amplifier (OTA). An OTA generally translates a voltage difference between a non-inverting v_+ and inverting input v_- into a proportional output current

$$I_o = g \cdot (v_+ - v_-), \quad (2.15)$$

where the transconductance g can be controlled by a bias current. Connecting the output potential back to the inverting input of the OTA allows emulate the leak behavior of a neuron, as in this configuration the generated output current tries to equalize the voltage difference between both inputs. In case the voltage over the capacitor exceeds the threshold set in the voltage comparator, a digital signal is sent to the flip-flop, and which sets Q. Setting Q starts the refractory counter and the spike gets registered by the event counter to be further processed by the digital event router. Additionally, the event triggers a reparameterization of the OTA, which also governs the leaky behavior, resetting the membrane potential. Once the refractory time is reached, the refractory counter sets R, un-setting Q, upon which the OTA responsible for the leak reset behavior is parameterized back to its leak operating mode. This allows for the membrane to evolve freely again. A neuron circuit on BSS-2 is highly configurable. A local 80 bit static random access memory (SRAM) allows to e.g. enable or disable the threshold comparator to switch between a LIF and LI model or to digitally set the capacitance C_m of the membrane capacitor to also change the dynamic behavior of the membrane. A further 24 analog parameters control potentials and conductances of the circuit. Each of these parameters can be set digitally by a 10 bit value. The analog values are realized as an array of capacitive cells, containing a small capacitor as well as 10 bit of local SRAM to store the digital value. To store the digital values as analog voltages, a periodically cycling voltage ramp, combined with a 10 bit counter is used. Each cell compares the counter value to its stored memory, and when a match is detected, the capacitor is momentarily connected to the voltage ramp (Hock, 2014). As a result, the capacitor acquires the potential of the voltage ramp at that specific moment in time. Since this functions as a form of capacitive memory, the stored 10 bit values will be referred to as capacitive memory (CapMem) values in upcoming sections. Figure 2.5 C shows that multiple neuron circuits can also be connected via a short circuit or a resistor. This can be done to model more complex neuron dynamics or to enable a neuron to have for more afferent connections.

Synapses Each synapse in the synapse array stores a 6 bit weight, that modulates the amplitude of the emitted current. Two integrator circuits, associated with a single neuron circuit, are located at the end of each column of the

synaptic array. Each circuit integrates the afferent stimuli, implementing an exponential kernel with an individually configurable time constant τ_s . Finally, the circuit stimulates the membrane potential depending on the filtered input stimuli. On BSS-2 this stimulation can happen conductance- or current-based. For current-based synapse, used in this thesis, one of the two circuits processes inhibitory inputs, while the other handles excitatory inputs. The integration and filtering of input stimuli is realized similarly to the membrane: A capacitor integrates the incoming currents, while a resistor connected in parallel replicates the exponential kernel in equation (2.5). Similar to the neuron circuit the conductance can be set by a CapMem value, to achieve a large range of synaptic time constants. For current-based synapses an OTA outputs a current proportional to the voltage over the capacitor.

Calibration of BrainScaleS-2 A calibration framework, calix (Müller et al., 2022; Weis, 2020), provides a simple way to configure all parameters of the analog circuits. Not only can the calibration framework be employed to set the hardware parameters to emulate a specific model behavior, but it also counteracts the inherent mismatch of circuits due to manufacturing. A simple to operate interface is provided to set target values for a specific neuron configuration. For example, in order to calibrate a LIF neuron, the target threshold, reset and leak potential, target values for the synaptic and membrane time constant and further non-model parameters can be set by the user. A model parameter is then calibrated by iteratively adjusting the relevant part of the hardware configuration. After one adjustment, a simple parameter-specific experiment is run and the model parameter is measured. The measured value is then compared to the target value to adjust the hardware configuration accordingly. This process is repeated several times until the desired model parameter is calibrated. As the parameters are calibrated one at a time, a full LIF neuron calibration of all 512 neurons on the chip takes roughly 5 min.

2.5 Gradient-Based Training on BrainScaleS-2

In order to train spiking neural networks on analog hardware in a gradient-based fashion, a learning framework based on PyTorch (Paszke et al., 2017), hxtorch (Spilger et al., 2023), is used. The idea behind hxtorch is to integrate

the modeling on BSS-2 into the PyTorch ecosystem, which provides state of the art optimizers and performs automatic backpropagation for gradient calculation.

```
class Model(torch.nn.Module):
    def __init__(self, n_hidden, n_out, ...):
        self.exp = hxsnn.Experiment()
        self.syn1 = hxsnn.Synapse(n_in, n_hidden, ...)
        self.hidden_neuron = hxsnn.LIF(n_hidden, ...)
        self.syn2 = hxsnn.Synapse(n_hidden, n_out, ...)
        self.output_neuron = hxsnn.LI(n_out, ...)

    def forward(self, input):
        x1 = self.syn1(input)
        x2 = self.hidden_neuron(x1)
        x3 = self.syn2(x2)
        output = self.output_neuron(x3)

        hxsnn.run(self.exp, time_steps)
        return output
```

Listing 1: Construction of an arbitrary spiking feedforward network using hxtorch. The construction works similar to PyTorch: The model inherits from `torch.nn.Module` and initializes all required layers in the constructor. The forward method defines, how inputs are propagated through the network. When called, the first four lines of code create a graph representation of the network in the experiment instance, required for execution on BSS-2. Calling `hxsnn.run(...)`, then emulates the network on BSS-2 and or simulates the dynamics the network.

In hxtorch, the user can easily combine different neuron types, e.g. LIF, LI or refractory LIF neurons, with synapses that connect the neurons, to form a network. In Listing 1, the construction of a spiking single hidden layer feed-forward network can be seen. The application programming interface (API) of hxtorch is designed so that the neural network construction is similar to PyTorch. In addition to its size, each neuron layer holds its parameterization, e.g. τ_s and v_{leak} . Each parameter is held internally as an instance of a `HXBaseParameter` class.

```

class HXBaseParameter:
    def __init__(self, hardware_value, model_value):
        self._hardware_value = hardware_value
        self._model_value = model_value

    @property
    def hardware_value(self):
        return self._hardware_value

    @property
    def model_value(self):
        return self._model_value

```

Listing 2: Class definition of an `HXBaseParameter`. The `HXBaseParameter` is initialized with its BSS-2 and model representation and holds these as properties.

The `HXBaseParameter` exposes its representation on BSS-2 through the property `hardware_value`. In case a hardware emulation is performed, the value is used as target value for the chip calibration (c.f. Section 2.4). For simulation and gradient computation the `model_value` is used. The definition is shown in Listing 2. This separation of model and hardware values proves useful for various reasons. At the time of writing this thesis the calibration of the leak, threshold, and reset potential requires desired values in least significant bits (LSBs) of the CADC measurement, which are integer values in the range from 0 to 255. In simulation, a dynamic range typically between 0 and 1 is chosen, which is why it is useful to have this separation. Another benefit is that the model value, required for simulation, of a time constant does not have to be a the true time constant in SI units, but can be arbitrarily transformed in simulation for better numeric stability. The forward method defines, how inputs are propagated through the network. Calling `hxsnn.run(...)` can then either only simulate the network in software or place and route the network onto BSS-2 to additionally emulate the network on hardware.

Training using hxtorch For training a network on hardware using `hxtorch`, a so called hardware-in-the-loop training methodology is applied (Schmitt et al., 2017). The general idea of this methodology is to replace estimated

values of state variables of the neurons from the simulation with actual measurements of the state variables on hardware. This is done to account for parameter mismatch of the hardware in the gradient calculation. To illustrate in more detail how the training works, the procedure of a single update step is described. The input spikes, stemming from the dataset, are fed through the analog network that was placed on BSS-2. When using surrogate gradients, the membrane trace and the spike times are required for gradient calculation. For this reason, the membrane potential, as well as the spikes of each neuron, are recorded during the emulation. After emulation, a simulation of the network model is run on the host computer to generate the computational graph which is required for the backpropagation. To account for the parameter mismatch between the analog system and the software model, the spikes and membrane potentials that were recorded during hardware emulation are normalized and injected at each time step of the simulation. In this case, normalization involves mapping the membrane potential from the measured CADC values to the corresponding range used in simulation. If the measurement timestamps do not align with the simulation time grid, the values are interpolated accordingly. For the membrane potential, an auxiliary identity function $\tilde{v}[t] = f(v[t], \tilde{v}[t]) \equiv v[t]$ is used to inject the data. $\tilde{v}[t]$ represents the simulated estimation of the hardware dynamics calculated with an update step from the value at time step $t - 1$, while $v[t]$ is the normalized recorded data at time t . The surrogate derivatives of the auxiliary function are defined as $\partial f / \partial v[t] = 0$ and $\partial f / \partial \tilde{v}[t] = 1$ for the membrane potential (Cramer et al., 2022). Similarly, an auxiliary function $\tilde{S}[t](S[t], \tilde{v}[t]) \equiv S[t]$ is implemented to replace the estimated spikes \tilde{S} from simulation by the actual recorded spikes S . The corresponding derivatives are defined using the surrogate gradient from equation (2.14):

$$\frac{\partial \tilde{S}[t]}{\partial S[t]} = 0, \quad \frac{\partial \tilde{S}[t]}{\partial \tilde{v}[t]} = (\rho|\tilde{v}[t] - \vartheta|)^{-2}, \quad (2.16)$$

where ρ describes the steepness of the surrogate gradient while ϑ is the threshold potential at which the LIF neuron spikes. Using the updated simulation and the automatic backpropagation and optimizers implemented in PyTorch, the weight updates are calculated and applied to the software model. To close the loop, the updated model parameters are scaled to the required 6 bit integer

weight resolution and written back on the BSS-2-chip. The scaling factor will be referred to as weight scale in later sections. As already described, the measurements recorded from the hardware need to be normalized, as the CADC outputs 8 bit integers while the simulation works with normalized floating-point numbers to ensure numerical precision and stability. In `hxtorch`, normalization involves two operations. First, an offset is applied to the original CADC values. This offset can either be a fixed value specified by the user or a dynamic offset based on the first measured CADC value. When setting the leak potential to 0 V, as commonly done in simulations, the dynamic offset conveniently aligns the measured leak potential with zero for each neuron individually. This works because, during the first CADC measurement, no input currents are present, and the membrane remains at its leak potential. After shifting, a trace scale is applied to adjust the CADC measurements in accordance with the simulation. Given a well-calibrated leak potential v_l and threshold ϑ , the trace scale can be calculated:

$$s_{\text{trace}} = \frac{\vartheta_{\text{SW}} - v_{l,\text{SW}}}{\vartheta_{\text{BSS-2}} - v_{l,\text{BSS-2}}}, \quad (2.17)$$

where the subscript SW is used for the values used in simulation and the subscript BSS-2 corresponds to the hardware values set in the calibration. This method therefore allows for training models on BSS-2 with arbitrary loss functions based on the membrane potential as well as spike times of the neurons.

2.6 Spiking Heidelberg Digits Dataset

When benchmarking SNNs on different visual datasets or benchmarks of ANNs (e.g. MNIST), the transformation to spike times often mitigates the comparability between performances of SNNs. To avoid this issue, Cramer et al. (2020) introduced SHD, which is a spike-based classification dataset. This dataset is based on an audio dataset, named Heidelberg Digits (HD). HD consists of 10420 high-quality aligned studio recordings with durations of around 1 s. These recordings include spoken digits from zero to nine in German and English stemming from a total of 12 different speakers. Two of these speakers are only present in the test set to ensure generalizability of a trained network.

For SHD, these high-quality recordings are transformed into spike times over 700 channels using a biologically inspired artificial model of the cochlea, called Lauscher (Cramer et al., 2020). Figure 2.7 shows the spikes over these 700 different input channels for a spoken “eins”, which is German for “one”, from the SHD dataset. The huge advantage of the SHD dataset is that these spikes directly serve as input for the SNN.

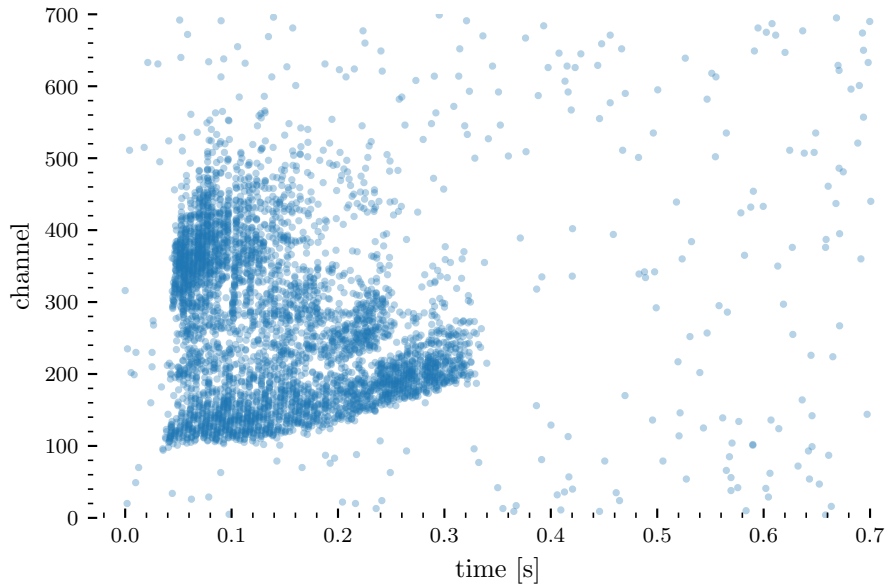


Figure 2.7: Transformed audio of a spoken “eins”, a spoken “one” in German, into spike times.

2.7 Related Work

This work is originally motivated by Perez-Nieves et al. (2021), a paper on how neural heterogeneity improves learning training, which investigates how learning the synaptic and the membrane constants in addition to the synaptic weights influences a network’s performance. Regarding the SHD classification task, they use a recurrent spiking neural network (RSNN) with a single hidden layer consisting of 128 LIF neurons. The readout layer included 20 LI neurons, of which each neuron corresponds to one class of the dataset. For the classification task, they use the a max-over-time decoder, that takes the maximal membrane potential of each readout neuron throughout the experi-

ment’s duration as input for the cross-entropy loss defined in equation (2.12). The class corresponding to the neuron with the highest membrane potential is predicted. Note that data augmentation, in form of adding spikes following a Poisson process of 1.2 Hz and randomly removing spikes with a probability of 1×10^{-3} , is added to the training data. The time constants are also clipped to be in range of $[3\delta t, 100 \text{ ms}]$, for numeric stability and biological plausibility. All parameters used in training are listed in Table 1. The weights are initialized by sampling from a uniform distribution $\mathcal{U}(-n^{-1/2}, n^{-1/2})$ with n being the number of afferent connections. The inputs from the SHD dataset are aligned to the same temporal grid used in the simulation. Learning weights and time constants of the RSNN, Perez-Nieves et al. (2021) reported a test accuracy of $(82.7 \pm 0.8)\%$ when initializing the time constants with the values from Table 1. When only learning weights, they achieved a classification accuracy of $(71.1 \pm 1.0)\%$ with the same initialization.

Table 1: Network parameters from Perez-Nieves et al. (2021).

Parameter	Value	Description
δt	0.5 ms	Simulation time step
τ_m	20 ms	Initial membrane time constant
τ_s	10 ms	Initial synaptic time constant
ϑ	1	Membrane threshold
v_l	0	Leak potential
v_r	0	Reset potential
ρ	100	Surrogate steepness
optimizer	Adam	Chosen optimizer
η	1×10^{-3}	Learning rate
n_{hidden}	128	Number of hidden neurons

3 Learning Neuron Parameters

When using analog hardware, multiple steps are required to implement the training of neuron parameters. This section describes the required steps from the implementation for simulation models only, to the full integration into the software framework, to function on BSS-2.

3.1 Implementation for Simulation

Learning the neuron parameters is first implemented in software for simulation. In this case, the implementation is straight forward, since the parameters that are trained only have to be held as `torch.nn.Parameter` with automatic gradient calculation enabled. Therefore, when running the simulation, the computational graph in PyTorch captures all calculations that occur on the corresponding parameters. Calling the automatic backpropagation of the loss determines the gradients of the neuron parameters from which the parameter updates can easily be applied using the optimizers implemented in PyTorch. Listing 3 shows the extension of the `HXBaseParameter` class. For the training to work, `HXBaseParameter` inherits from `torch.nn.Module`. This ensures that trainable parameters are recognized by PyTorch and updates can be applied. The API is chosen, so that `make_trainable()` can be called on the parameter, instantiating the model value as a `torch.nn.Parameter`.

As can be seen from Listing 4 there are no breaking changes introduced and the training of parameters is only an optional extension.

3.1.1 Training on Neuron Trace in Software

To show that the implemented learning of neuron parameters works, a small dummy experiment is performed. Spike trains which follow a Bernoulli process with $p = 0.03$ are sent on a single LI with manually chosen parameters. This is done for 100 different spike trains, generating one target trace each. Subsequently, the parameters are randomized. The goal of the training is for the originally differently parameterized neuron to reproduce the target trace, meaning the target neuron and the neuron have the same parameters describing their dynamics after training. To achieve this, the Adam optimizer in

```
class HXBaseParameter(torch.nn.Module):
    def __init__(self, hardware_value, model_value):
        self._hardware_value = hardware_value
        self._model_value = model_value

    def hardware_value(self):
        return self._hardware_value

    def model_value(self):
        return self._model_value

    def make_trainable():
        self._model_value =
            torch.nn.Parameter(self._model_value)
```

Listing 3: Extension of HXBaseParameter for training parameters in software. The HXBaseParameter inherits from `torch.nn.Module` so that trainable parameters are recognized by the optimizer. The `make_trainable()` instantiates the model parameter as `torch.nn.Parameter`.

```
model = Model(n_hidden, n_out, ...)
model.lif_h.tau_mem.make_trainable()
```

Listing 4: Example on how to achieve trainability of neuron parameters using the model from listing 1. After initializing the model, the `make_trainable()` function can be called on the parameter that should be trained. In this case the membrane time constant is trained. This call can already happen in the model initialization.

combination with a simple mean squared error (MSE) loss function

$$l = \sum_n (v_n^{target} - v_n)^2 \quad (3.1)$$

is used. Here, n iterates over the simulation time, while v_n^{target} and v_n are the membrane voltages at the corresponding time step. Using this simple task, each neuron parameter can be tested separately if trained correctly. As every parameter has a different influence on the dynamics of the membrane potential, the training of multiple parameters can also be tested simultaneously. This is done by training both time constants, τ_s and τ_m , as shown in Figure 3.1. After 50 epochs of training the trace almost perfectly aligns with the target trace and the trained neuron parameters are close to reaching the target parameters. From these results it is safe to assume that the training of parameters works in simulation and indicating that it can be tried to achieve the same results on BSS-2.

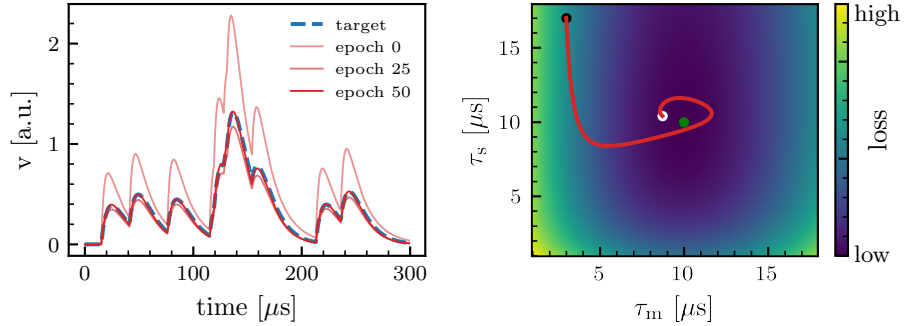


Figure 3.1: Training run of time constants on a neuron trace: Neuron trace at different training epochs (left). Loss-landscape with the time constant evolution during training (right). For the training, a batch of 100 different target membrane traces are simulated using the parameters $\tau_s = 20 \mu\text{s}$ and $\tau_m = 10 \mu\text{s}$, of which one trace is shown on the left. In the right figure, these target parameters are represented by the green dot. At epoch 0, the training starts with parameters $\tau_s = 17 \mu\text{s}$ and $\tau_m = 3 \mu\text{s}$, represented by the black dot on the right. During training the time constants are then optimized according to equation (3.1).

3.2 From Simulation to Emulation on BrainScaleS-2

The next step involves translating our working software model to the hardware. As described in Section 2.5, the aim is to train the network using the hardware-in-the-loop framework. In comparison to the original framework, not only are the weights updated on hardware, but also the other neuron parameters which are learned during training. However, the translation of neuron parameters is often not a simple scaling as for the weights. This is due to more complex dependencies that arise on hardware. Therefore, the easiest method would be to recalibrate the system with the model parameters after each training step. However, every single calibration run takes approximately 5 minutes. For a simple training with e.g. 100 epochs each containing 100 batches, this would lead to a duration of approximately 3×10^6 s, which corresponds to roughly 35 days. As this is not feasible, another approach is needed.

Following this approach, each hardware parameter corresponding to a trainable model parameter is characterized. Then an ideal parameter translation, which translates the model parameter to the corresponding hardware parameter is defined. Since the parameter translation is not used in the PyTorch model computational graph, it does not need to be differentiable.

As seen in Listing 5, the `HXBaseParameter` must be further extended to support the training of parameters on BSS-2. For greater flexibility, the ideal translation is provided by the user in form of the `set_on_chip` function. This takes the chip configuration as well as the coordinates of neurons on the chip and sets the configuration corresponding to the parameter translation on the chip. With the ideal translation, the updated parameters can easily be calculated. This method of updating the parameters is much faster than performing a recalibration after each training step. The only downside of this approach is that every circuit on the analog substrate is different. Consequently, a single ideal translation does not fit every neuron circuit, introducing further parameter mismatch between the software model and the emulation on hardware.

Based on the results presented by Perez-Nieves et al. (2021), the characterization of both time constants is performed first. In order to find the ideal

```

class HXBaseParameter(torch.nn.Module):
    def __init__(self, hardware_value, model_value):
        self._hardware_value = hardware_value
        self._model_value = model_value

    def hardware_value(self):
        return self._hardware_value

    def model_value(self):
        return self._model_value

    def make_trainable(set_on_chip_func=None):
        self.set_on_chip_func = set_on_chip_func
        self._model_value =
            torch.nn.Parameter(self._model_value)

    def set_on_chip(chip, neuron_coordinates):
        if self.set_on_chip_func is None:
            raise ValueError(
                'When executing on HW, '
                + 'set_on_chip_func needs '
                + 'to be provided'
            )
        self.set_on_chip_func(self.hardware_value,
                               chip,
                               neuron_coordinates)

```

Listing 5: Extension of `HXBaseParameter` for training parameters on BSS-2. The ideal translation is provided by the user in form of the `set_on_chip` function, that sets the configuration corresponding to the parameter translation.

translation, a desired hardware parameter is set to then measure the corresponding model parameter. Then, either a function can be fitted to the pairs of parameters or some interpolation scheme can be used to obtain the values of parameters not represented on the measured grid.

3.2.1 Characterization of the Membrane Time Constant

Following the equations for the LI dynamics on hardware,

$$\frac{C_m}{g_l} \frac{\partial u}{\partial t} = -(u - u_{\text{leak}}) + \frac{1}{g_l} I \quad (3.2)$$

and in software

$$\tau_m \frac{\partial u}{\partial t} = -(u - u_{\text{leak}}) + \underbrace{\frac{1}{g_l}}_{\text{const.}} I \quad (3.3)$$

a direct relation between the model parameter and the configurable hardware parameters $\tau_m = \frac{C_m}{g_l}$ can be derived. While the membrane capacitance C_m can be configured by a 6 bit integer, the leak conductance g_l can be varied by setting a 10 bit CapMem value (c.f. Section 2.4). Before these hardware parameters are characterized, the method with which to measure the membrane time constant is presented.

Measuring the Membrane Time Constant In general, there are multiple possibilities to measure the membrane time constant τ_m of a certain hardware configuration. In this work, the membrane time constant is measured using the method, that is also employed by the current calibration routine (Weis, 2020). The idea is to put an offset current on the membrane to shift the potential above the leak potential. At a certain point in time, the offset current is switched off, so that the membrane potential exponentially decays towards its leak potential. The membrane potential can be measured with e.g. the MADC and an exponential decay $v(t) = ae^{-\frac{t}{\tau_m}} + b$ can be fit to the measured data. An example of a single measurement with the corresponding fit can be seen in Figure 3.2.

Variation of Leak Conductance As derived from (3.2) and (3.3), the membrane time constant can be varied by setting the leak conductance g_l .

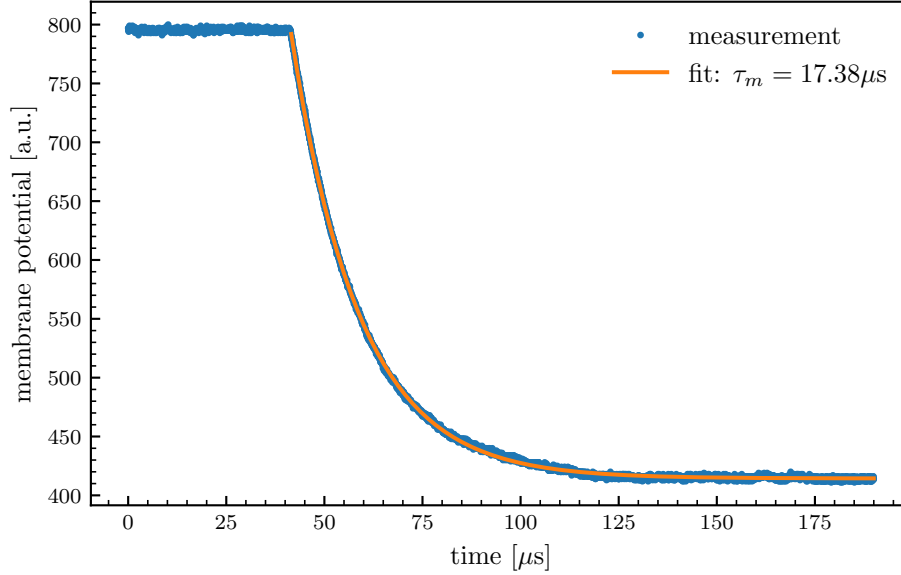


Figure 3.2: Measurement of the membrane time constant τ_m using an offset current. The membrane potential is offset by the offset current until it is switched off at $50\mu\text{s}$ leading to an exponential decay of the membrane potential.

The hardware is designed such that the circuits are able to generate conductances spanning three orders of magnitude (Billaudelle, 2022). To achieve this range, there are two separate switches that can be set apart from the CapMem values for the leak conductance. One of the switches allows to multiply the leak conductance by a factor of 9 and therefore shorten the time constants to $0.26\mu\text{s}$. This switch is mainly used to achieve a nearly instant reset of the membrane potential to a desired potential after a neuron has spiked. The other switch, which is more relevant for this work, allows to divide the leak conductance by a factor of 9, which extends the achievable range of time constants up to $950\mu\text{s}$. Typical applications of SNNs require time constants in the order of $10\mu\text{s}$. For this reason, the measurements were only made for the case where no switch is set and for the case where division of the leak conductance is enabled. A sweep of all configurations mentioned regarding the leak conductance for a single neuron can be seen in Figure 3.3. For a single measurement point, the membrane potential following the dynamics described in the previous paragraph, is recorded 10 times. To reduce membrane noise, the mean of all 10 membrane traces is calculated. In the end, the exponential

equation is fit to the mean trace over these 10 trials, to obtain the measured time constant for a single configuration of the leak conductance.

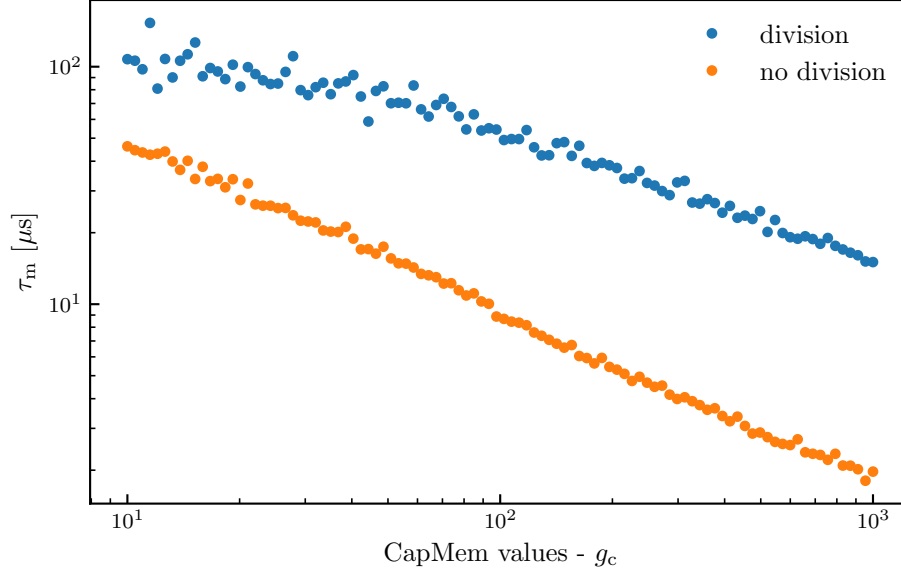


Figure 3.3: Measurement of the membrane time constant τ_m for a leak conductance sweep with and without enabling conductance division. When division is enabled longer time constants for the same CapMem value can be observed.

From Figure 3.3 a seamless transition between both operating modes can be observed, meaning that any time constant between the minimum time constant without division and the maximum time constant with division can be achieved. Additionally, the relationship between the configurable CapMem values and the membrane time constant appears to be linear in the log-log plot, revealing the dependency

$$\tau_m \propto g_c^x, \quad (3.4)$$

where g_c represents the CapMem value used to set the actual leak conductance and x is an arbitrary parameter. Since the ideal translation is the one that fits the best to all neurons on a chip, it is obtained by fitting equation (3.4) to the measurements of all neurons with and without division. Figure 3.4 shows these measurements with the corresponding fits. Especially at the lower time constant tails it can be observed that the ideal translation strongly deviates from

the measured values. However, since the membrane time constant at which division is enabled can be chosen and the measurements have an overlapping time constant range, this discrepancy does not pose a severe problem. For example, for time constants greater than $30 \mu\text{s}$ leak division can be used and then switched off for smaller time constants thereby acquiring an overall better translation. Regarding the tail, where division is not enabled, the mismatch is mainly observed at time constants lower than $1 \mu\text{s}$, which are not typical for machine learning applications or even biological networks, considering the speed-up factor of 1000.

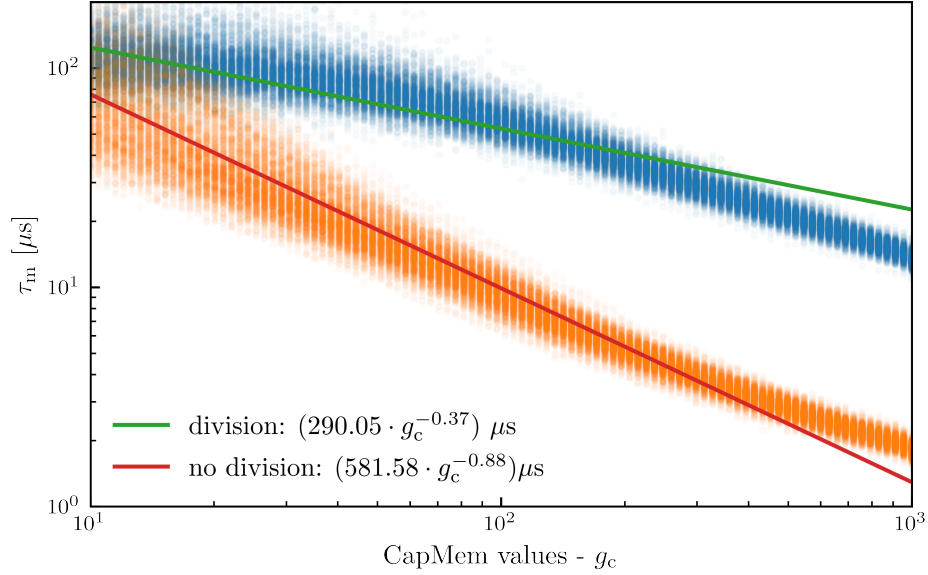


Figure 3.4: Measurement of the membrane time constant for a leak conductance sweep with and without enabling conductance division for all 512 neurons. A fit is performed on the measurements of all 512 neurons to find the ideal translation.

Training of Leak Conductance Similar to the simulation, it is shown that the training of the membrane time constant BSS-2 works by setting the leak conductance. A closer look at equation (3.2) reveals that on hardware not only does the membrane time constant depend on the leak conductance, but also the contribution of the input current to the change of the membrane potential. In simulation the leak conductance is normally set to a constant value of 1 S . Therefore, the modeling has to account for this scaling of inputs. This can be

implemented by adjusting the differential equations using a scaling factor

$$\tau_m \frac{\partial u}{\partial t} = -(u - u_{\text{leak}}) + \frac{\tau_m}{\tau_{m,0}} \frac{1}{g_{l,0}} I \quad (3.5)$$

represented in blue, where $\tau_{m,0}$ is the membrane time constant at the start of the training. The scaling can only be used when the membrane capacity C_m remains constant. With the ideal translation and the adjusted model, the same training as already done in simulation (c.f. Section 3.1.1) is implemented on hardware. For the training the target trace is created using a calibrated chip. In the training example shown in Figure 3.5 the chip is calibrated to have a membrane time constant of 10 μs and a target trace is recorded. Then, the time constant is altered to some arbitrary value already using the ideal translation.

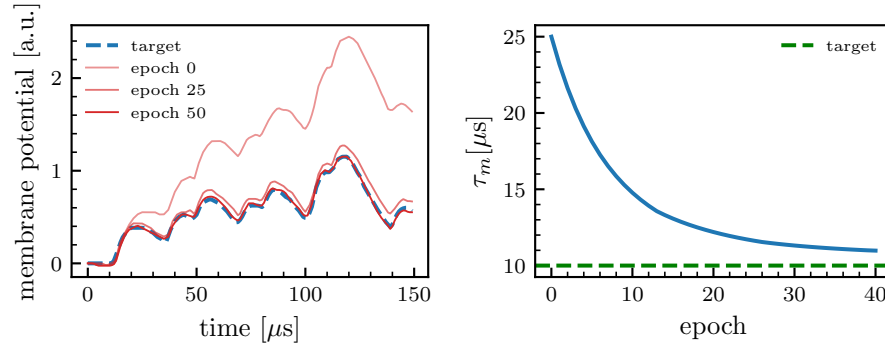


Figure 3.5: Training of membrane time constant on neuron trace using the leak conductance: neuron trace at different training epochs (left) and the evolution of the model value of the membrane time constant (right).

With this as a starting point, the model has to adjust the time constant using the translated leak conductance on hardware to replicate the recorded membrane target trace. From Figure 3.5 it can be observed that this training also works very well, even on the noisy hardware trace. The actual time constant that is found by the optimization is different from the one that was set for the calibration. The in-the-loop training is thus able to compensate for the mismatch resulting from the ideal translation.

Variation of the Membrane Capacitance The second option to alter the membrane time constant on hardware is to vary the membrane capacitance C_m . The advantage of using the capacitance to change the membrane time constant is that it does not have significant side effects like the input scaling of the leak conductance. However, the membrane capacitance can only be configured using a 6 bit integer, allowing for values from 0 to 63. This results in a much lower resolution of time constants compared to the leak conductance. Additionally, the range in which time constants can be reached heavily depends on the leak conductance of the neuron. For that reason, the ideal translation also depends on the configured leak conductance. As the calibration routine usually sets the maximum value for the capacitance and adjusts the leak conductance to achieve the desired membrane time constant, the maximum time constant for the ideal translation is fixed by the original translation. In Figure 3.6, the membrane time constant is measured for all configurable capacitor sizes using two different calibrations. The time constants set in the calibration can be read directly from the measurement at the highest capacitor size $c = 63$ and are $50\mu\text{s}$ and $10\mu\text{s}$, respectively. On the left, the measurement for a single neuron with these two calibrations is depicted, whereas on the right the measurements for all neurons, including the ideal translation fit is shown. The range differences depending on the used calibrations and therefore different leak conductances can be directly observed. The ideal translation fits the measured data considerably better than the ideal translation using the leak conductance (cf. Figure 3.4).

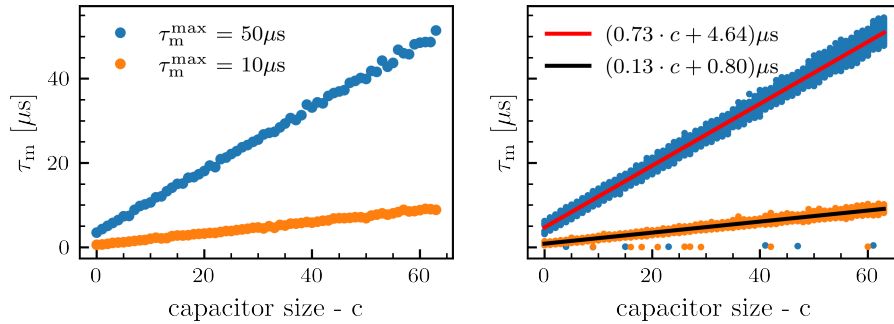


Figure 3.6: Measurement of the membrane time constant for a membrane capacitance sweep for a single neuron (left) and all 512 neurons (right). This is done for two different calibrations of the membrane time constant.

Training of the Membrane Capacitance As previously done, the training of the membrane time constant on a target trace using the capacitance is performed, to see if the framework and the ideal translation work properly. The training should be tested from both sides, e.g. starting from a time constant both smaller and greater than the target time constant of the calibration. Here, it is only shown from one side. Consequently, for this training, a calibration is used in which the capacitor size is set to the value of 32 instead of the usual 63. The target membrane time constant is set to $25\text{ }\mu\text{s}$, so that the ideal translation for the calibration to $50\text{ }\mu\text{s}$ could be used, since the range and capacitor steps are roughly similar for both of these calibrations. In Figure 3.7 the results of a training run are shown. After training the membrane trace almost perfectly fits the recorded target trace. When looking at the evolution of the membrane time constant over the different epochs, it seems that the actual minimum of the loss is not at the original target membrane time constant of $25\text{ }\mu\text{s}$, but slightly higher. This is expected, since on the one hand the calibration routine does not yield a perfectly calibrated chip and on the other hand the ideal translation also does not fit every neuron on the chip perfectly. Following this line of arguments, the training-in-the-loop framework is also able to compensate for a slight mismatch of parameters introduced by the ideal translation. Furthermore, dampened oscillations in the evolution of the membrane time constant can be observed. These oscillations stem from the optimizer overshooting the local minima slightly less every time the local minima is crossed.

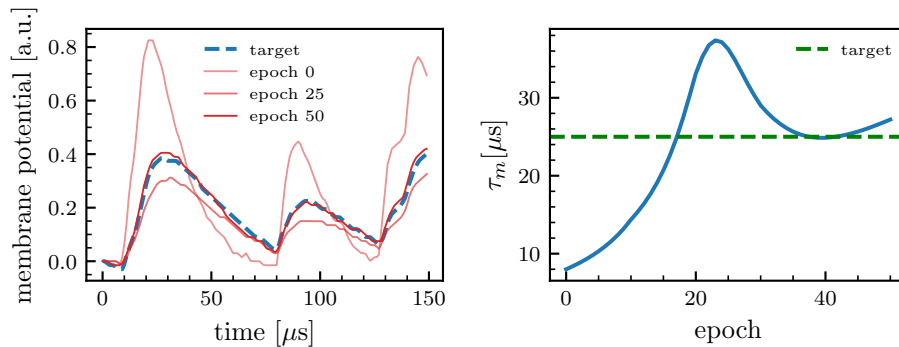


Figure 3.7: Training membrane time constant on trace with membrane capacitance: membrane trace for different epochs (left). Membrane time constant evolution over epochs (right).

3.2.2 Characterization of the Synaptic Time Constant

In this work current-based synapses are used. The model current I follows the differential equation

$$\tau_s \frac{\partial I}{\partial t} = -I + \sum_j w_j S_j \quad (3.6)$$

where the last term sums over all spikes of the presynaptic neurons S_j and weighs them with a corresponding weight w_j . As the differential equation (3.6) is similar to equation (3.3) a similar circuitry is used for the emulation on hardware. Since at the end a current is needed, the potential over a synaptic membrane serves as input for an OTA, which converts the potential to a current. Assuming linearity of the OTA, the synaptic input currents then follow

$$\frac{1}{g_{\text{inh/exc}}} \frac{\partial I}{\partial t} \propto -I + \sum_j w_j S_j \quad (3.7)$$

on hardware, where the conductance $g_{\text{inh/exc}}$ can be individually configured for inhibitory and excitatory synapses.

The capacitance for the synaptic input is not as configurable as the capacitance for the membrane of the neuronal circuits. Apart from the normal mode, there is also a small-capacitance-mode for very small synaptic time constants, which is not relevant in this work. Therefore the synaptic time constant is only set by the conductance:

$$\tau_s \propto \frac{1}{g_{\text{inh/exc}}}. \quad (3.8)$$

Measuring the Synaptic Time Constant Unfortunately, the synaptic current that charges the membrane of the neuron cannot be measured directly. Only the potential over the capacitor, that serves as input of the OTA can be measured, so that non-linearities of the OTA are not included in this characterization. Similarly to the membrane time constant, the measurement routine, employed in the current calibration framework, is used to measure the synaptic time constant. For this, a spike event is created to charge the

capacitor of the synaptic input. In order to extract τ_s , an exponential decay is fit to the measurements in Figure 3.8.

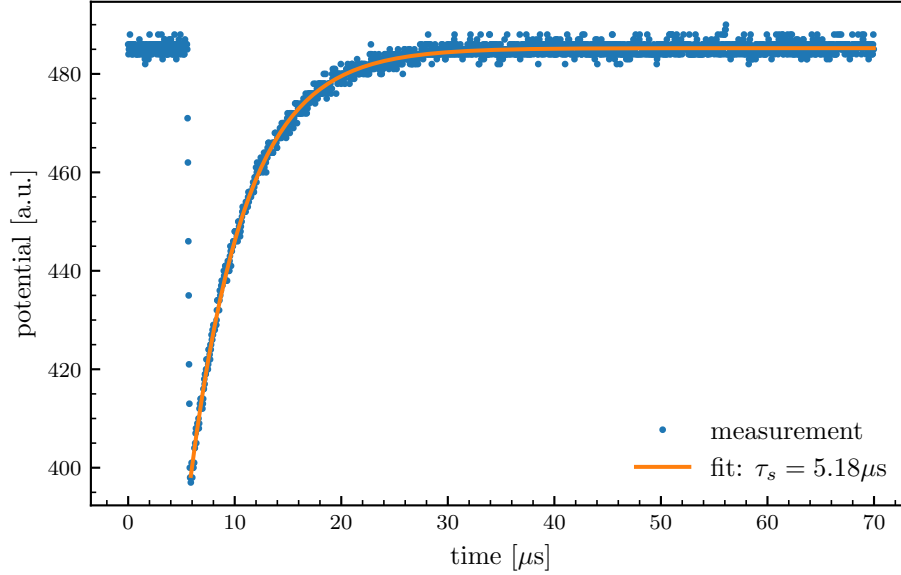


Figure 3.8: Measurement of the synaptic time constant τ_s . A spike event happens at $8\mu\text{s}$, after which the capacitor relaxes towards its resting potential, following an exponential decay with the synaptic time constant τ_s . An exponential decay is fit to the measurements.

Variation of the Synaptic Conductance In order to find the ideal translation for the synaptic time constant, the conductance controlling it is varied, measuring the synaptic time constant for each configuration. Similarly to the neuron circuits representing the membrane, the conductance can be set using a 10 bit CapMem value. On BSS-2 there are different circuits for excitatory and inhibitory synapses. Consequently, the measurements are performed on both input types individually.

In Figure 3.9 measurements of the synaptic time constants for a sweep of all possible conductances can be seen for a single excitatory and a single inhibitory synapse. It is observed that the synapses behave approximately equal, meaning the synaptic time constants are only offset by a small amount for the same set of CapMem values responsible for the conductance. However, the relationship in the log-log plot does not appear to be as linear as for the neuron membrane

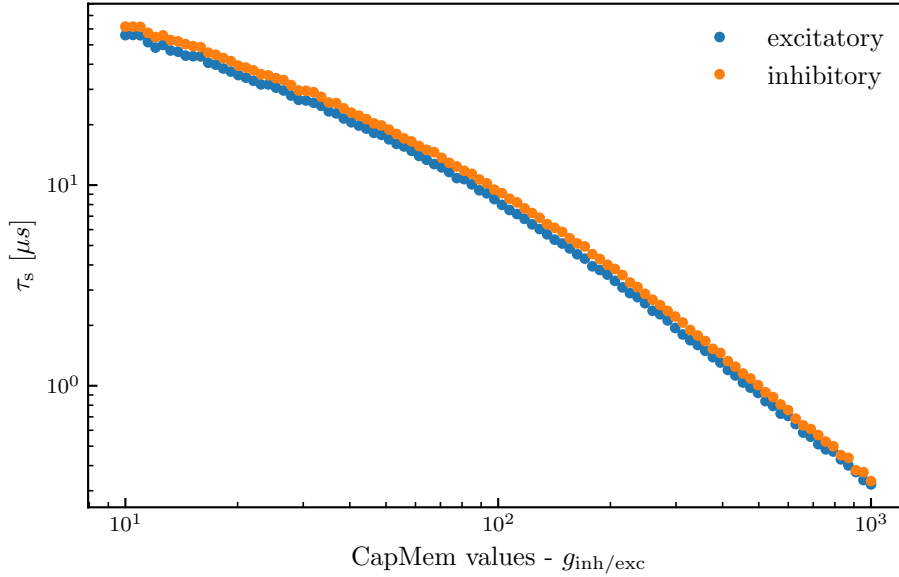


Figure 3.9: Measurement of the synaptic time constant τ_s for different conductance configurations of a single synapse. The inhibitory and excitatory circuit exhibit approximately the same synaptic time constant for the same CapMem values.

circuits. Still, a function following $\tau_s \propto g_{\text{inh/exc}}^x$ is fit to all synapses to acquire the ideal translation used to train the synaptic time constant in experiments. The ideal translation as well as the corresponding measurements can be seen in Figure 3.10. As already observed for the single synapse in Figure 3.9 the excitatory and inhibitory synapses behave quite similar overall, meaning the ideal translation has nearly the same fit parameters. However, the discrepancy between individual synapses is quite big. Especially at low conductances there is a spread of approximately $70 \mu\text{s}$ between the synaptic time constants for the same conductance configuration. Additionally, the ideal translation does not fit the measurements well at lower time constants.

Training of the Synaptic Time Constant Lastly, the same experiment as before can be used to check whether the training of the synaptic time constant works as expected. For this experiment the target trace is recorded using a LI neuron with a synaptic time constant of $10 \mu\text{s}$. The starting point of the training is a time constant of $40 \mu\text{s}$. As can be seen in Figure 3.11, after training, the membrane trace almost perfectly fits the target. However, when

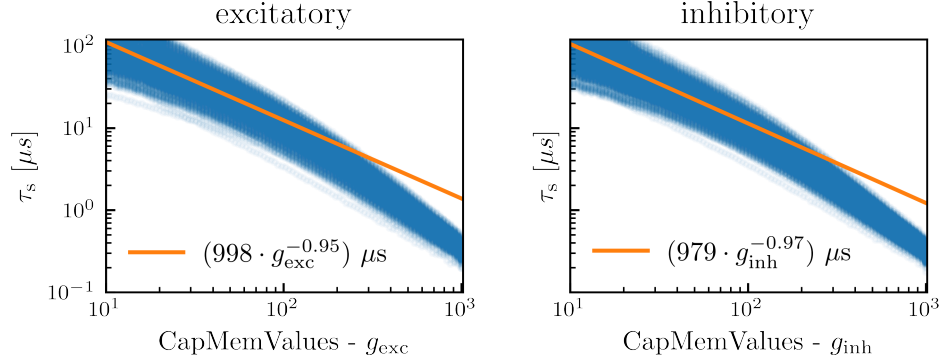


Figure 3.10: Measurement of the synaptic time constant τ_s for different conductance configurations of all synapses with corresponding ideal translation: excitatory synapses (left) and inhibitory synapses (right).

looking at the evolution of the synaptic time constant, it can be observed that the time constant converges to approximately 20 μs and not to the targeted 10 μs . This is probably due to the discrepancy between the ideal translation and the actual measured values, as seen in Figure 3.10. Therefore hardware-in-the-loop training is once again able to compensate for the parameter mismatch in this experiment.

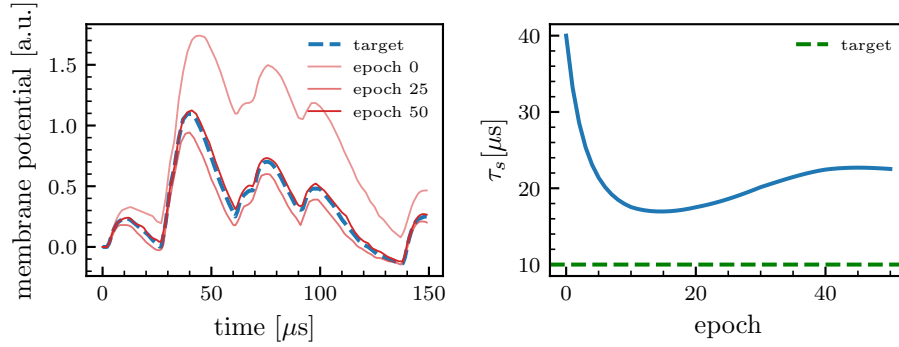


Figure 3.11: Training of the synaptic time constant on hardware: membrane trace at different epochs (left) and evolution of the synaptic time constant over the training epochs with the target model value (right).

4 Learning Time Constants on SHD

As Perez-Nieves et al. (2021) have shown, especially in tasks where the time domain is important, learning time constants is beneficial. An example of such a task is the SHD classification task, introduced in Section 2.6. In this section, experiments that reproduce the simulation results of the SHD classification task shown in Perez-Nieves et al. (2021) are presented. Next, the entire process to implement this task on BSS-2 is outlined.

4.1 Learning Time Constants in Simulation

With the confirmation that time constants can be learned in the hxtorch framework (c.f. Section 3.1.1), the simulation results related to the SHD dataset from Perez-Nieves et al. (2021) are reproduced. For this purpose training runs in simulation, using the same network parameters listed in Table 1, are performed. As these experiments are primarily performed to provide a proof of concept, the data augmentation used in the work from Perez-Nieves et al. (2021) is not implemented here. Figure 4.1 shows the mean and standard deviation of the test accuracy of during training over 50 epochs across 10 different seeds. When only training weights the network reaches a classification accuracy of $(70.5 \pm 3.0)\%$. Additionally, training the synaptic and membrane time constants of each individual neuron proves beneficial, resulting in an accuracy of $(75.9 \pm 2.5)\%$. Unfortunately, the accuracy reported in Perez-Nieves et al. (2021) is not reached when time constants are trained. This discrepancy could however stem from the data augmentation, as it proves very useful for SHD (Nowotny et al., 2025; Cramer et al., 2022).

The distribution of the time constants after training is depicted in Figure 4.2. The membrane time constant distribution after training looks similar to the distribution shown in Perez-Nieves et al. (2021). Unfortunately, they do not provide the distribution of the synaptic time constant. The small peak at 100 ms is due to the clipping of the time constants at this point.

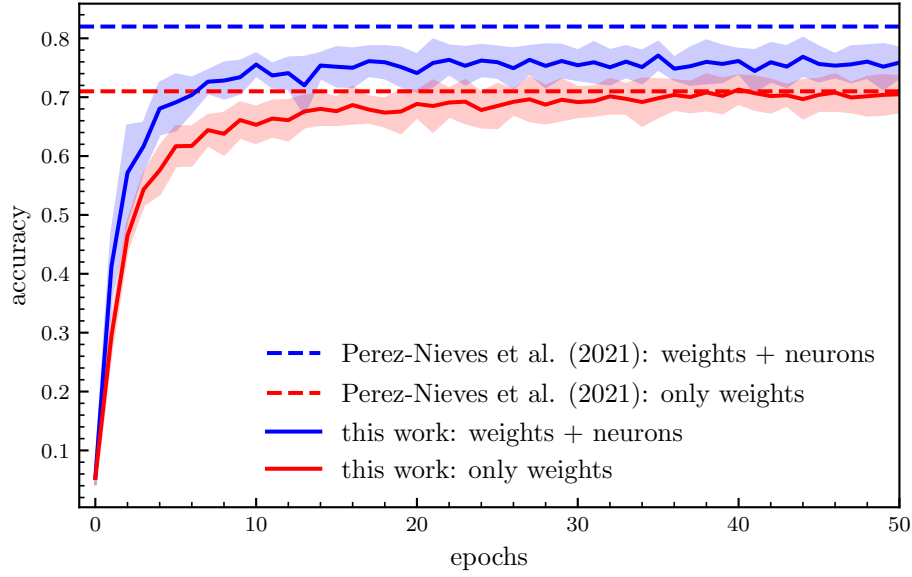


Figure 4.1: SHD classification mean test accuracies during training of a RSNN when training only the weights (red curve) and when training weights and neuron parameters (blue curve) across 10 seeds. The shaded area represents the standard deviation. The reported results achieved by Perez-Nieves et al. (2021) of the corresponding models are shown as dashed lines in the respective color.

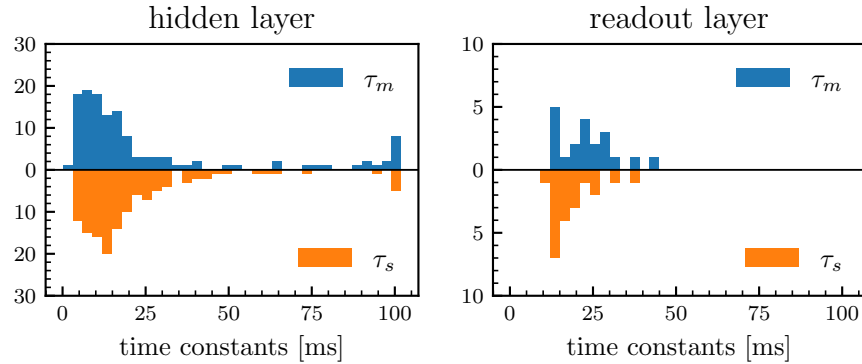


Figure 4.2: Histograms of the time constant distribution in the hidden and the readout layer after training for both the membrane time constant τ_m and the synaptic time constant τ_s for one random seed.

4.2 Simulation Tests for Hardware Emulation

With these simulation results in mind, planning the implementation on hardware can be started. Unfortunately, at the time of writing this thesis it is not yet possible to use the exact same network topology as used in Perez-Nieves et al. (2021) in an effective fashion.

The first limitation that was encountered is that a single neuron circuit BSS-2 can only have a maximum of 128 signed inputs (see Section 2.4). However, in the network used in Perez-Nieves et al. (2021), there are 700 inputs from the SHD dataset in addition to 128 recurrent connections between the hidden neurons. This leads to a total of 828 inputs per single hidden neuron, a number that is not feasible for a single neuron circuit. In order to achieve this fan-in on BSS-2 seven neuron circuits could be short-circuited to reach a maximum fan-in of 896. However, when operating on a single chip accommodating 512 neuron circuits in total, this would have the consequence of only 73 neurons being able to use.

Additionally, there arises a second challenge when using surrogate gradients in emulation on BSS-2: the hardware-in-the-loop methodology (Schmitt et al., 2017; Cramer et al., 2022) using surrogate gradients requires the membrane voltage to be measured at best at every simulation time step. The CADC is used to record the membrane trace of every neuron and then send the recorded data to memory. As mentioned in Section 2.4, the SIMD processor, responsible for the CADC measurement has two different memories that can store the data. The fast but small storage option is an SRAM. This option writes a sample approximately every $2\text{ }\mu\text{s}$, however, as the SRAM is relatively small, only 100 samples per neuron can be saved. This leads to a maximum recording time of $200\text{ }\mu\text{s}$, which is not sufficient for the 1 ms of inputs in the SHD dataset, when accounting for the speed-up factor of 1000. Therefore using the SRAM is not feasible for the problem at hand and the other option, the DRAM connected to the FPGA, must be used. As it is significantly larger, it can record and save the entire experiment duration of 1 ms . However, the DRAM, with a sampling frequency of around $6\text{ }\mu\text{s}$ significantly slower than the time step of $0.5\text{ }\mu\text{s}$ originally used in simulation.

In order to handle all of these hardware limitations, simplifications of the model are developed and simulation tests are performed to investigate the impact on

the network performance. To address the problem of the large input numbers of the neurons, the number of inputs of each hidden neuron is reduced. It is first tested if, instead of a recurrent network architecture a feed-forward network can be implemented to remove all recurrent connections and simplify the model. Then, to achieve an even further reduction of the number of inputs for each hidden neuron, the influence of subsampling the inputs on the classification accuracies is analyzed. Next, as a final simulation test to check if the DRAM can be used without any disadvantages, the network performance is investigated using different simulation and input time resolutions. For all of these experiments, all candidate models are run using 10 different random seeds. If not specified otherwise, each model uses the parameters shown in Table 1. To assess the results, the mean validation accuracy during the training of the network is used to see the impacts of the implemented simplifications. An important remark is that instead of using the test set of SHD as a validation set, as it is done in Perez-Nieves et al. (2021) and other papers (e.g. Cramer et al., 2020 and Bittar and Garner, 2022), another approach for training and validating neural networks is chosen. During training, the training set is randomly split in two subsets where 80% of the data in the training set is used for training while the remaining 20% serves as the validation set. This is common machine learning practice, as the test set should only be used once at the very end to estimate how your final model will perform on completely new data.

Feedforward and Recurrent Neural Network The first simplification that is tested is the use of a simpler feedforward network instead of a recurrent network. This is done to reduce the number of inputs per hidden neuron since all recurrent connections are removed while also simplifying the model and its numerics. For this purpose four different models are tested:

- recurrent with weight training
- recurrent with neuron parameter and weight training
- feedforward only with weight training
- feedforward with neuron parameter and weight training

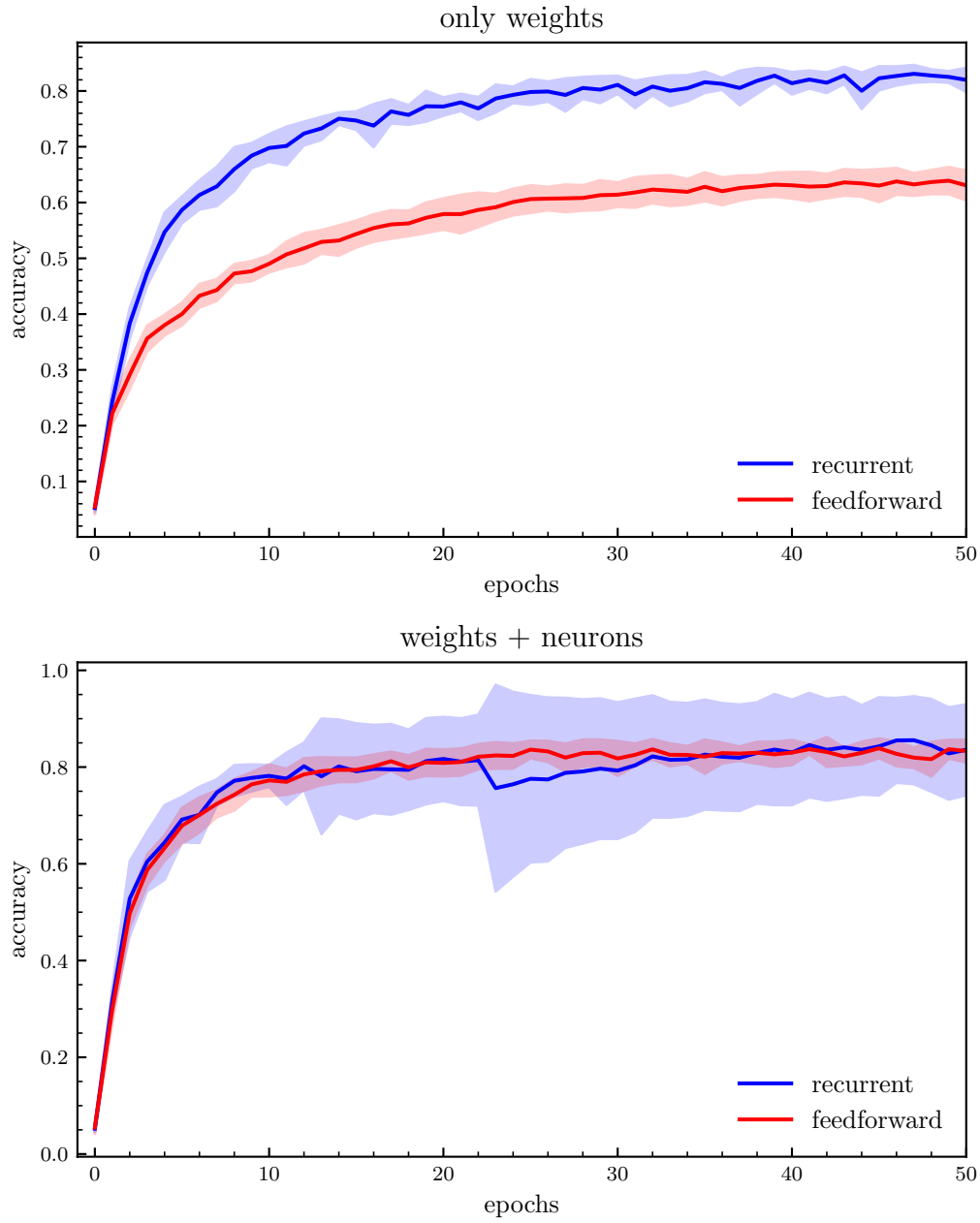


Figure 4.3: Validation accuracy during training on the SHD task for a feed-forward (red) and a recurrent (blue) network when training only weights (top) and when training weights and time constants (bottom) in simulation. The mean validation accuracy of runs across 10 different seeds is represented by the curve, while the shaded area around the curve corresponds to the standard deviation.

As can be seen in Figure 4.3, when only the weights are trained the recurrent network clearly outperforms the feedforward network. Intuitively, the recurrent network is able to hold information over a longer period of time, which is important for a language recognition task and therefore achieves a higher classification accuracy. However, when also training the time constants of the neuron, the feedforward network is able to achieve similar accuracies, since the neurons are able to capture the time dependency with the variable dynamics. In Figure 4.3, two drops in mean accuracy at epochs 12 and 22 coupled with a rise of the standard deviation over the 10 different seeds can be observed for the recurrent network training. This can be explained by looking at the individual accuracy of each seed, shown in Figure 4.4, in which two different seeds are having a huge drop in accuracy during training at these epochs, respectively. This consequently results in an observable drop in the mean accuracy. Given these results, the feedforward architecture is used for first experiments on BSS-2, since it achieves a similar performance as the recurrent network when training weights and time constants, while also clearly outperforming the network in which only weights are trained. Additionally, for initial experiments on BSS-2 it is easier to handle a network that is less complex.

A comparison of the learned time constant distributions between the feedforward and the recurrent network is shown in Figure 4.5. While the time constant distribution in the hidden layer looks similar for both models, both time constants in the neurons of the readout layer in the feedforward network shift to longer time constants. After training almost all time constants of the neurons in the readout layer are above 50 ms.

Reduction of the Number of Inputs Apart from changing the network architecture from a complex recurrent network to a simpler feedforward architecture, the influence of the number of input channels from the SHD dataset is investigated. This is done to further reduce the inputs per hidden neuron to accommodate the attainable 128 inputs per neuron circuit. Here, the subsampling method, which is already employed for experiments on BSS-2 in Cramer et al. (2022), is implemented, so that the same data are used in both experiments and a fair comparison of results can be done. Following the subsampling method, only certain input channels of the SHD dataset function as inputs of

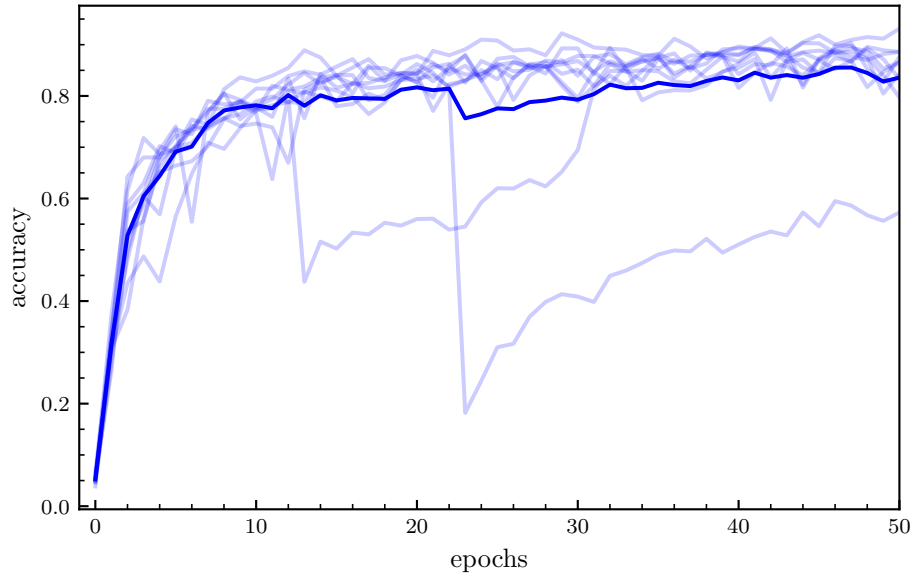


Figure 4.4: Validation accuracy during a training of a recurrent network when training weights and neuron parameters for each of the ten seeds separately.

the network. Specifically, the input channels from 0 to 70 are omitted and then every 9th channel is taken as input. This reduces the number of inputs per neuron to 70 instead of 700 in the hidden layer, and therefore enables the use of one neuron circuit per hidden neuron. To test the effect of this subsampling on the network performance, a feedforward network with 128 hidden layers is used. The network is trained in four different configurations:

- 700 inputs with neuron parameter and weight training (green)
- 700 inputs only with weight training (yellow)
- 70 inputs with neuron parameter and weight training (blue)
- 70 inputs only with weight training (red)

In Figure 4.6, the validation accuracies during training for the different numbers of inputs with their respective standard deviation over 10 different seeds can be seen. An obvious observation that can be made is that, independent of the employed training method, the network with 700 inputs outperforms

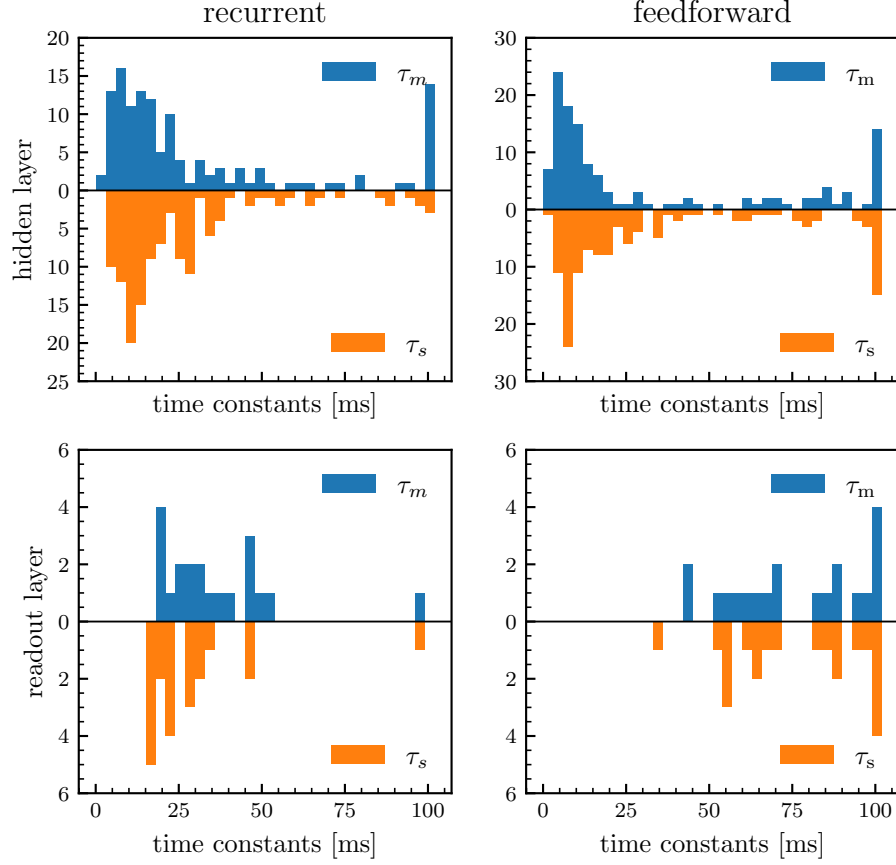


Figure 4.5: Comparison of learned time constant distributions after training between recurrent and feedforward network architecture for one random seed.

the network with 70 inputs from the SHD dataset. This makes sense, because on the one hand, the network with 700 inputs has many more parameters, while on the other hand it does not lose any information from omitting specific channels. However, there are also two remarkable observations that can be made. Firstly, despite the network with 700 inputs that only trains weights, having nearly 8 times more parameters ($128 \cdot (700 + 20) = 92160$) than the network with 70 inputs that trains weights and neuron parameters ($128 \cdot (70 + 20 + 2) + 40 = 11816$), the latter network clearly outperforms the first network. This is because the network with subsampled inputs that also trains neuron parameters is able to capture the time domain of the inherent problem which can not be compensated by the increased number of parameters resulting from the input increase. Secondly, the differences between the

networks using the same training method but different numbers of inputs are investigated. For the network that only trains weights, increasing the number of inputs results in a much larger increase in accuracy (approximately 20 %) compared to the accuracy gain (approximately 10 %) that the network which additionally trains its neuron parameters experiences upon increasing its number of inputs. This is the case even though both networks have the same information loss through the decrease of inputs.

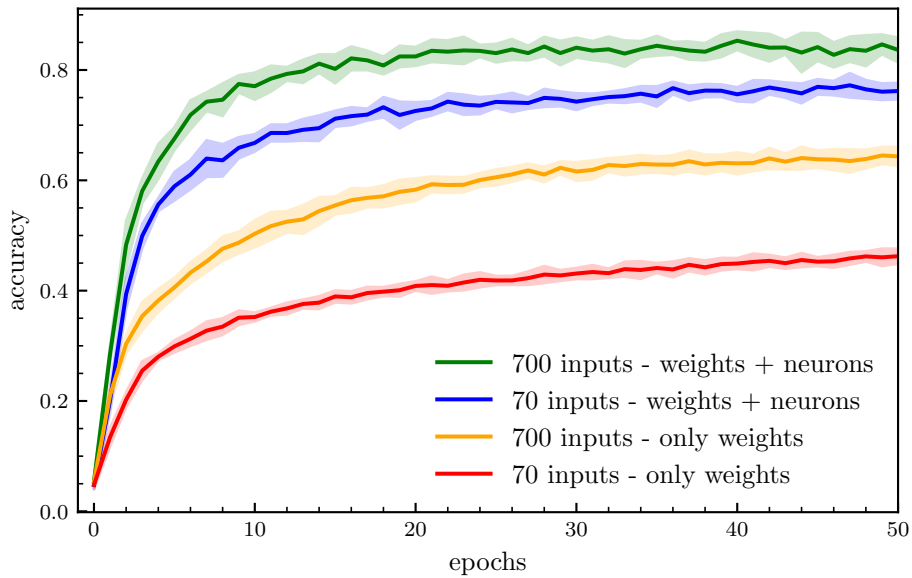


Figure 4.6: Comparison of the validation accuracy between different numbers of inputs (70 and 700) when training only weights (red, yellow) and when training weights and neuron parameters (blue, green).

Time Resolution Using all previous simplifications from, namely a feedforward network using the subsampled SHD dataset with 70 inputs, the impact of the time resolution used for the input coding as well the simulation time step is investigated. Note that for these experiments the inputs from SHD are again aligned to the same temporal grid used in simulation. All input spikes that are inside the same time step are regarded as one spike on the grid in simulation. Furthermore, the range of allowed time constants is adjusted for the time step at hand. The smallest allowed time constant is set to be equal to three times the duration of a simulation time step, while the upper limit

is held constant at 100 ms for all time resolutions. The mean and standard deviation of the accuracy from training across 10 different seeds is shown in Figure 4.7. The networks are trained again over 50 epochs. While a small decline in mean accuracy can be observed when only training weights, the chosen time resolution does not have any impact on the validation accuracy when the time constants are trained in addition to the weights.

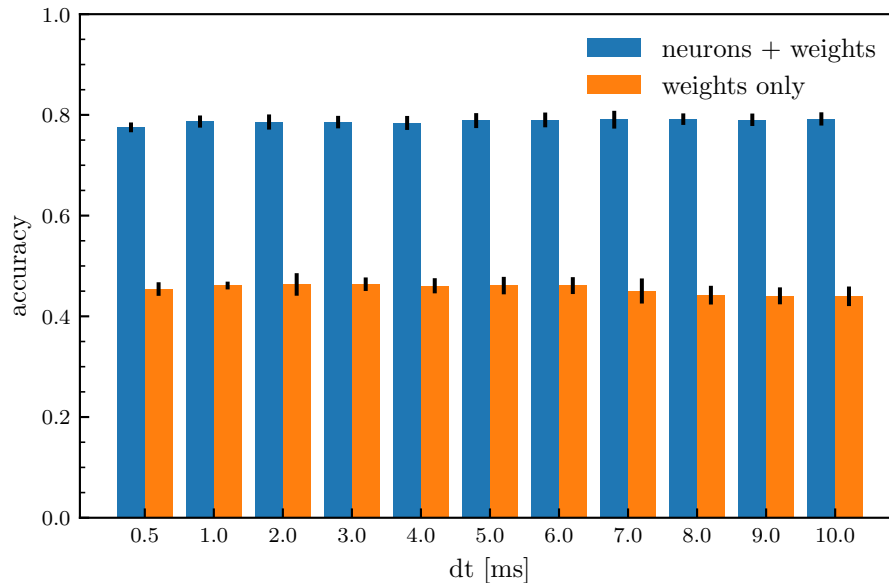


Figure 4.7: Comparison of the mean validation accuracy for different time resolutions when training weights (orange) and when training neuron parameters in addition to the weights (blue). The standard deviation across 10 seeds is shown as the black error bar.

The distributions of time constants are shown in Figure 4.8. The distribution of $\delta t = 2$ ms is chosen as this is the chosen simulation time step for experiment on BSS-2 (c.f. Section 4.3.4). Once again this time, for larger time steps, a shift to larger time constants for the neurons in the readout layer can be observed. From these simulation results it can be concluded, that any time step in the range of 0.5 ms to 10 ms can be chosen, while still achieving the same validation accuracy. Choosing a larger simulation time step has a few advantages especially when training on BSS-2. For one, the time step can be chosen such that it approximately matches the sampling frequency of the CADC recording, so that only minimal interpolation on the simulation time grid must be

performed. Additionally, a larger time step significantly reduces the computational cost of the simulation for the calculation of gradients, leading to the training being much faster.

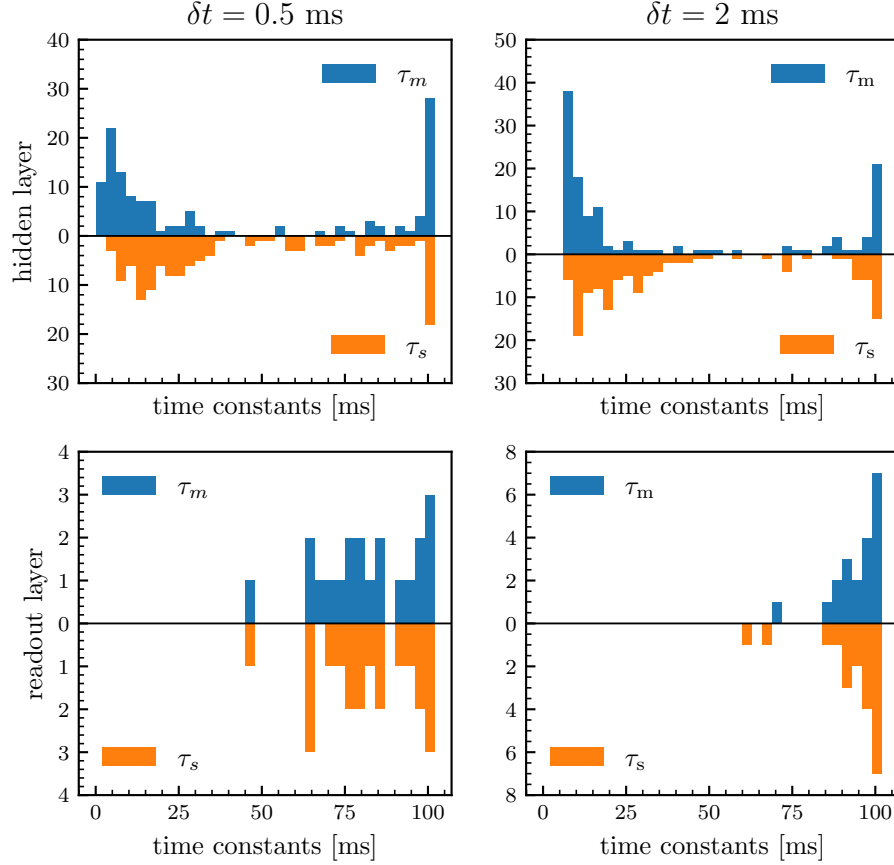


Figure 4.8: Comparison of learned time constant distributions after training between different time steps for one seed, shown for the hidden and readout layer separately.

4.3 Emulation on the SHD Dataset

This section outlines the process of training the model on BSS-2 using in-the-loop training introduced in Section 2.5. First, the challenges encountered and their corresponding work-arounds are discussed. Finally, the achieved results are presented.

4.3.1 Shifting Leak Potential

As already described in Section 3.2.1, the membrane time constant can be trained using two hardware parameters - the leak conductance and the membrane capacitance. Even though training using the membrane capacitance does not require any software model adjustment, initially the leak conductance is used for training of the membrane time constant. There are two main reasons that support this decision. First, the leak conductance can be set using a 10 bit CapMem value. Additionally, switches that multiply or divide the leak conductance by a certain factor can be used to enlarge the achievable range. These configuration options can set a large range of time constants, without compromising the resolution. Secondly, if the membrane capacitance is set to its maximum value, the recording of a membrane trace has the best signal-to-noise ratio (SNR). Apart from the scaling of inputs, which can be accounted for by the simulation model in training, varying the leak conductance has another side effect: not only does it alter the membrane time constant, but it also influences the leak potential. This change in the leak potential is induced by offset currents in the OTA controlling the leakage. When calibrating the chip, the offset currents are usually compensated for by shifting the potential provided for the non-inverting input of the OTA (Billaudelle, 2022). This however is only possible, if the transconductance is held constant. When changing the transconductance of the OTA to vary the membrane time constant, the output currents of the OTA and therefore also leak potential shifts. An extensive analysis on the interplay between leak potential and the transconductance setting has been performed in parallel to this work in Hinterding (2025). Hinterding (2025) further elaborates on how this problem can be handled using a multi-dimensional parameter transformation model. Unfortunately, this is still a work in progress at the time of writing this thesis. It has to be noted that the shift in the leak potential is only a problem when using LIF neurons, as the leak potential is shifted relative to the threshold and reset potential. When using LI neurons, as in Section 3, this shift does not matter as long as the CADC or membrane capacitor do not saturate. Since spiking neural networks require spiking neurons, the membrane capacitance is used for training the membrane time constant for the final results (Section 4.3.4). Consequently, the trainable range of the membrane time constant is fixed by

the calibration of the chip and has to be adjusted according to the problem at hand. In this case, a maximum membrane time constant of $100\text{ }\mu\text{s}$ is required for the experiment. However, the calibration framework, calix, allows only for values up to $60\text{ }\mu\text{s}$ for the membrane time constant. To circumvent this, the chip is calibrated to have a membrane time constant of $50\text{ }\mu\text{s}$ and two neuron circuits are short-circuited to double the capacitance of a neuron, increasing the range of time constants up to $100\text{ }\mu\text{s}$. This also enables a higher fan-in for each neuron and therefore using a recurrent model.

4.3.2 CapMem Crosstalk

A second issue involves the analog values being stored in the capacitive memory. Both the bias current defining the leak conductance and the bias current defining the synaptic time constant are stored as analog CapMem values. As described in Section 2.4, a CapMem stores a digital value as analog voltage using a periodically cycling voltage ramp. For each quadrant on BSS-2 there is one voltage ramp responsible for all its CapMems. If many CapMem values are identical, all of these capacitors are connected to the voltage ramp at the same time leading to a momentary voltage swing of the voltage ramp. This voltage swing causes the CapMems to store a voltage which is smaller or higher than the desired value. This crosstalk effect then leads to the neuron circuits exhibiting undesired behavior. When using a calibrated chip, the crosstalk does not pose a problem, as the calibration routine does not typically set many identical CapMem values, due to the circuit mismatch introduced by manufacturing. However, since a calibration is too time consuming one ideal transformation is used for all neurons in the network, which becomes a problem. First, initializing the network homogeneously leads to all neurons having the same CapMem values. This can be circumvented by either initializing the model values, that are represented by CapMem values on BSS-2, heterogeneously. In the following results (Section 4.3.4), a small amount of noise is added to the affected CapMem values during initialization. Secondly, the trained time constants are capped to biologically plausible values (c.f. Section 2.7). During training with a feedforward architecture using the reduced amount of inputs, presented in Section 4.2, it can be observed that in simulation especially the time constants in the output layer saturate at the upper

bound of the limit. This also happens, when performing in-the-loop training on BSS-2, causing CapMem crosstalk. A general approach to circumvent this would again be to add random noise onto each CapMem value. However, noise should only be added when the saturation of multiple time constants occurs to prevent additional mismatch between simulation and emulation on BSS-2. This makes implementation challenging. Additionally, at the time of writing this thesis, a fast lookup-based calibration, which would mitigate this problem, is on its way (Hinterding, 2025). For this reason, this work focused on other difficulties that arose during emulation on BSS-2. However, this in turn means that in the results (Section 4.3.4), CapMem crosstalk still poses a problem.

4.3.3 Saturation of the Readout Neurons

The third problem occurring during training runs on BSS-2 involves saturation of the CADC. The saturation can be observed when comparing the membrane trace of a simulation to the recorded trace during the emulation on BSS-2. The main reason for this is the use of a max-over-time decoding scheme in combination with the cross entropy loss. The cross entropy loss is minimized by the divergence of the difference between the output of the correct target neuron and the other neurons. In the case of the max-over-time decoding scheme, this happens when the maximum of the membrane trace of the target readout neuron

$$v_{\text{readout}, t} \rightarrow \infty, \quad (4.1)$$

approaches infinity, while the maxima of all other readout neurons

$$v_{\text{readout}, \bar{t}} \rightarrow -\infty \quad (4.2)$$

approach negative infinity. Therefore, the loss function generally pushes the membrane traces of the output layer towards extreme values. This can also be observed when training in simulation. If no regularization is applied the loss pushes the membrane traces of the readout neurons to values up to 40. When emulating on hardware, the leak potential is calibrated to a CADC value of 80 LSBs, while the threshold is chosen at a CADC value of 150 LSBs. With the simulation leak at 0 and the threshold at 1, this results in a trace

scale $s_{\text{trace}} = \frac{1}{70 \text{ LSBs}}$, according to equation (2.17). As the CADC has a 8 bit resolution, CADC saturation occurs in normalized simulation values at

$$v_s = (v_{\text{max}} - v_{\text{shift}}) \cdot s_{\text{trace}} \approx 2.65, \quad (4.3)$$

where v_{max} corresponds to the maximal CADC output of 256 LSBs, while $v_{\text{shift}} = 70 \text{ LSBs}$ is the leak potential shift to 0. Therefore without any further modifications, eventually the CADC of many readout neurons saturate in training on BSS-2, leading to possibly wrong predictions. Again, there are a few ways to minimize the problem. The first solution that comes to mind is adjusting the calibrated threshold value such that the maximum normalized CADC measurement is 40. This would however strongly reduce the dynamic range and resolution on BSS-2, as this would require $s_{\text{trace}} \approx \frac{1}{5 \text{ LSBs}}$ and therefore also only 5 LSBs between leak and threshold on hardware. For this reason this idea is not used. This also means that the entire dynamic range of the software representation cannot be represented with a desirable resolution. The readout layer needs another representation on hardware than the hidden layer. Therefore, a second approach is to simply use the trace scale required for these high normalized values in the readout layer, while using the actual trace scale only in the hidden layer. This approach was implemented and tried, but unfortunately it did not prove beneficial, as the saturation of readout neurons still has a strong influence on the achieved accuracy. The trace scaling of the readout neuron can also be adjusted in combination with the weight scale. Dividing the weight scale with the same factor as multiplying the trace scale of the readout neurons leads to an effective way to scale the synaptic efficacy of readout layer neurons against hidden neurons. This approach is able to significantly improve the accuracy on the training task and is therefore used for the final results. Two other approaches involve changing the loss function by adding regularizing terms. Instead of adjusting the hardware to the model, the training process is adjusted to accommodate for the dynamic range of the hardware. An extra loss term

$$l_{\text{readout}} = \frac{\rho_r}{N_{\text{readout}}} \sum_{i=1}^{N_{\text{readout}}} (\max_t v_{r,i}[t])^2 \quad (4.4)$$

penalizes high readout membrane traces. The hyperparameter ρ_r controls how

strong the maximal values of the readout membrane traces are penalized. This extra loss term is able to significantly push the maximal readout traces down to lower values with a small loss in accuracy in simulation. Implementing the loss term when training on BSS-2 did not improve accuracy, for reasons that could not yet be resolved. Lastly, another loss term in the hidden layer is introduced: a loss on the mean spike rate in the hidden layer z_h :

$$l_{\text{hidden}} = \rho_h \max(0, \sum_{i,t} z_{h,i}[t] - \zeta) \quad (4.5)$$

which penalizes spike numbers higher than the threshold ζ , therefore favoring sparse firing activities, a crucial benefit regarding power efficiency of SNNs. Furthermore, it reduces the amount of spikes being forwarded to the readout layer, affecting the maximal values of the membrane trace. In simulations this regularization term did not harm accuracy significantly, while providing the mentioned benefits. Therefore, it is implemented in the final model. Note, that saturation occurs also at the lower limit. This does affect the loss value in the end, it is however not relevant for the correct prediction using a max-over-time decoder. Unfortunately, these adjustments of the model and the hardware configuration are only able to mitigate but not fully solve the saturation problem.

4.3.4 Results on BrainScaleS-2

As discussed in the previous sections, the original model from Perez-Nieves et al. (2021) is not suited for emulation. Thus, the model is simplified in many aspects. As a starting point, two feedforward networks implementing a single hidden layer with 128 neurons are trained on the SHD task. One model only trains the synaptic weights (homogeneous model), while the second model trains the synaptic and the membrane time constant in addition (heterogeneous model). Afterwards, the same is done for a recurrent model that has one layer with 128 neurons. The number of inputs is reduced to 70 instead of using the original 700 input channels from the SHD dataset. Similar to Cramer et al. (2022) a simulation time step of $2\mu\text{s}$ is chosen, while a speedup factor of 2000 is employed. This means that the input from the SHD dataset is time-gridded with a time bin-width of 4 ms. Using a speedup factor of 2000 also means that the time constants must be clipped at the upper limit of

50 μs to maintain the corresponding original limit of 100 ms. Derived from the simulation time step, the lower limit of allowed time constants is 6 μs (c.f. Section 2.7). The weights are clipped as well to accommodate for the 6 bit weight resolution on BSS-2. As described earlier, a scaling factor $s_{h \rightarrow r} = 10$, which scales the readout layer against the hidden layer is employed to mitigate the problem of saturation in the readout neurons. Additionally, the regularization term introduced in equation (4.5) is implemented. All model parameters are summarized in Table 2. The target parameters that are used for the calibration of the chip when training the time constants are shown in Table 3, otherwise the nightly calibration “spiking” is used.

Table 2: Used model network parameters on BrainScaleS-2.

Parameter	Value	Description
δt	2 μs	Simulation time step
Δt	4 ms	Bin size for SHD
τ_m	10 μs	Initial membrane time constant
τ_s	10 μs	Initial synaptic time constant
ϑ	1	Membrane threshold in simulation
v_l	0	Leak potential in simulation
v_r	0	Reset potential in simulation
ρ	100	Surrogate steepness
optimizer	Adam	Chosen optimizer
η	1×10^{-3}	Learning rate
ζ	500	Threshold of rate regularization
ρ_h	1.5×10^{-3}	strength of rate regularization
n_{hidden}	128	Number of hidden neurons

Feedforward Model The validation accuracy over 70 epochs of training the feedforward model for 13 different seed is shown in Figure 4.9. It can be observed that the model where time constants are trained in addition to the weights clearly outperforms the model, where only weights are trained. At the last epoch, the heterogeneous model achieves a validation accuracy of $(65.0 \pm 2.5)\%$, while the homogeneous model reached a validation accuracy of $(40.9 \pm 3.0)\%$, with the dataset having a chance level of 5%.

Unfortunately, both models are not able to achieve the same results as in simulation. This could have multiple reasons. For one, through the scaling

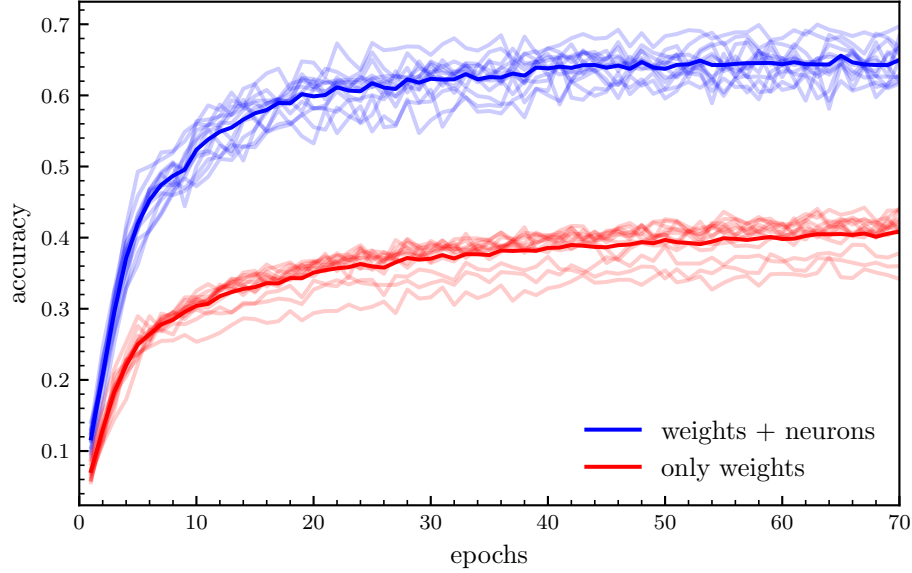


Figure 4.9: Mean validation accuracies across 13 different seeds, when training weights only (red) and when training weights and time constants (blue). The training of each seed is shown as a shaded line.

factor $s_{h \rightarrow r}$, mismatch that cannot be compensated by the in-the-loop training may be introduced. Additionally, the described problem of the saturation of readout neurons could not be solved completely, especially when training the heterogeneous model. Interestingly, a correlation between the maximum voltage in the readout layer and the validation accuracy at the last epoch training can be observed (see Figure 4.10). This suggests that the handling of saturation is either insufficient or induces further effects.

The distribution of time constants is depicted in Figure 4.11. Once again it can be observed that many time constants in the readout layer reach the upper limit. These long time constants in the readout layer in combination with a max-over-time loss essentially reproduce a sum-over-time loss, which integrates the membrane trace instead of using the maximum operation. As the sum-over-time decoder is inherently better suited for the SHD classification task (Nowotny et al., 2025), it makes sense that the heterogeneous model outperforms the homogeneous model. In the training on BSS-2, the time constants in the hidden layer also shift strongly towards the upper, but also the

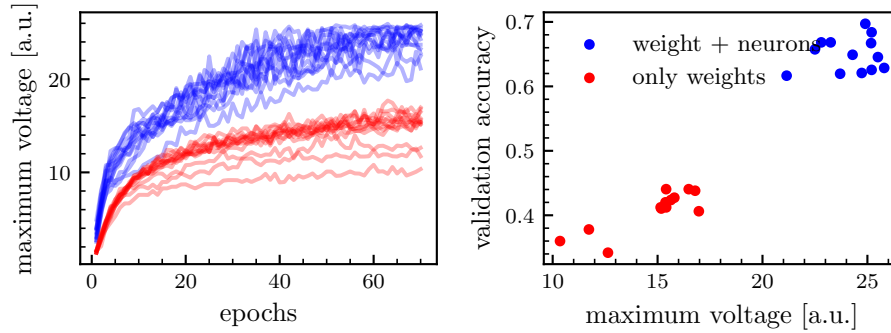


Figure 4.10: Maximum voltage of the membrane in the readout layer over all training epochs for the different seeds (left) and a scatter plot of the maximum voltage and the validation accuracy at the last epoch of training for all seeds (right). The maximum voltage rises during training for both models. Especially for the homogeneous network a correlation between the maximal voltage and the validation accuracy can be observed.

lower, limit. As, however, both time constants in the hidden and in the readout layer are rather homogeneous after training, CapMem crosstalk, discussed in Section 4.3.2, remains a problem with the use of the same transformation function for all neurons. Additionally, as the time constants are rather homogeneous after training it is not clear that the training of time constants is really beneficial. For this reason, it is investigated whether a homogeneous model with the membrane time constant initialized at the upper limit of $50\text{ }\mu\text{s}$ and the synaptic time constant at the maximum allowed value of calix at $30\text{ }\mu\text{s}$ can perform similarly. Achieving an accuracy of $(48.7 \pm 1.4)\%$, the model with long time constants shows a clear improvement over the variant with short time constants. Nonetheless, it does not match the performance of the model where time constants are optimized during training.

The weight distribution in the respective layers are shown in Figure 4.12. Both distributions approximately follow a gaussian distribution. However, especially in the hidden layer a large peak at the lower limit can be observed. This strongly inhibits the neurons in the hidden layer and most probably stems from the rate regularization term in equation (4.5).

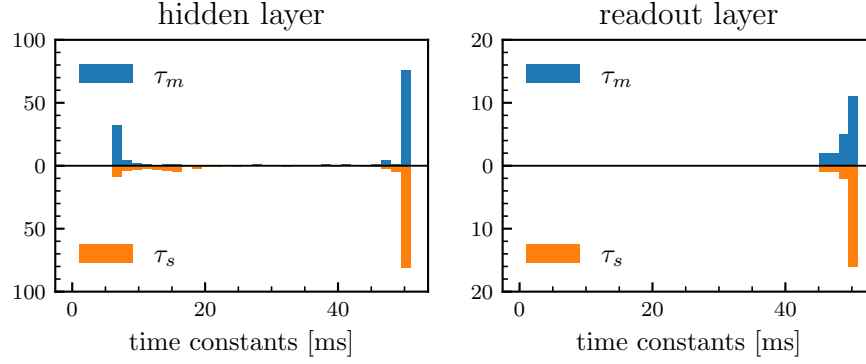


Figure 4.11: Distribution of time constants after training in the hidden as well as the readout layer for one seed.

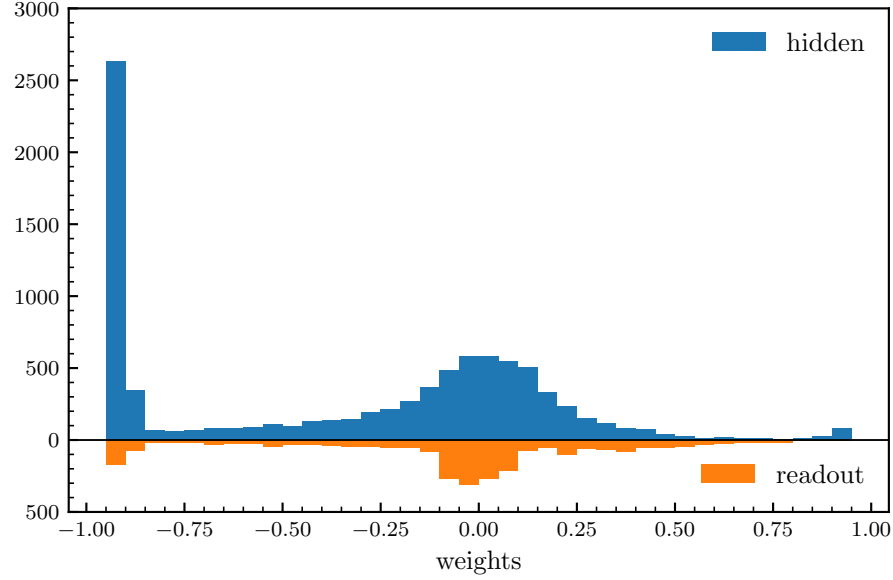


Figure 4.12: Distribution of weights after training in the hidden as well as the readout layer for one seed.

Recurrent Model Since Perez-Nieves et al. (2021) used a recurrent model, a recurrent model is also run on BSS-2. The validation accuracy of the recurrent model across 10 different seeds is shown in Figure 4.13. At the last epoch, the model with time constant training reaches a validation accuracy of $(57 \pm 14)\%$, while the model where only synaptic weights are trained achieves $(57 \pm 7)\%$. Comparing this to the results of the feedforward network, it can be seen that the model implementing only weight training strongly benefits

from the recurrent connections even though training the recurrent model on BSS-2 is very unstable. Independent of whether time constants are trained or not, some seeds exhibit accuracy drops during training. Similar drops can be observed in simulation (Figure 4.4).

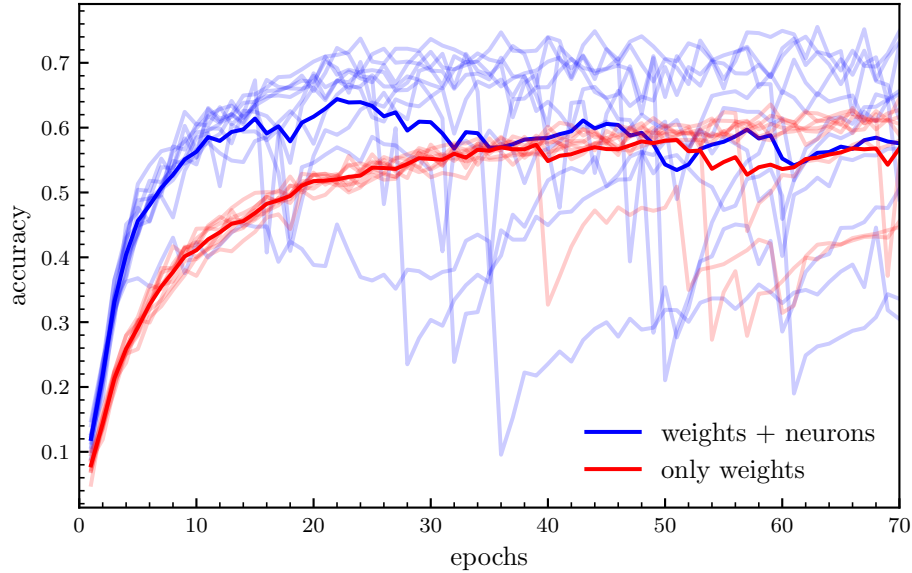


Figure 4.13: Mean validation accuracy of the recurrent model for 10 different seeds, when training weights only (red) and when training weights and time constants (blue). The training of each seed is shown as a shaded line.

When looking at the validation accuracy in combination with the maximum observed membrane voltage in the readout layer for a seed where these drops can be observed (Figure 4.10), it can be seen that these drops in accuracy are heavily correlated with the maximum voltage in the readout layer. Even though this correlation is observed, at the time of writing this thesis, the reasons for these drops in accuracy are not yet understood.

The time constant distribution after training for one seed is depicted in Figure 4.15. Large peaks at both limits are still observed. Both time constants in the readout layer are trained towards the maximum value. This again suggests that, despite the explicit recurrent connections, the max-over-time decoder in combination with the long time constants in the readout layer rather functions

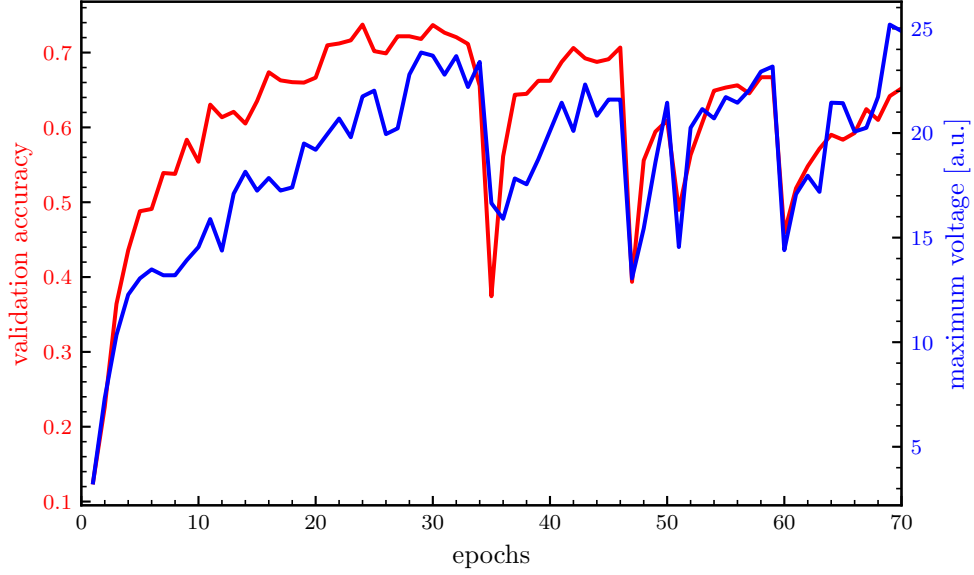


Figure 4.14: Validation accuracy and maximum voltage in output layer for one seed where accuracy drops can be observed.

as a sum-over-time decoder.

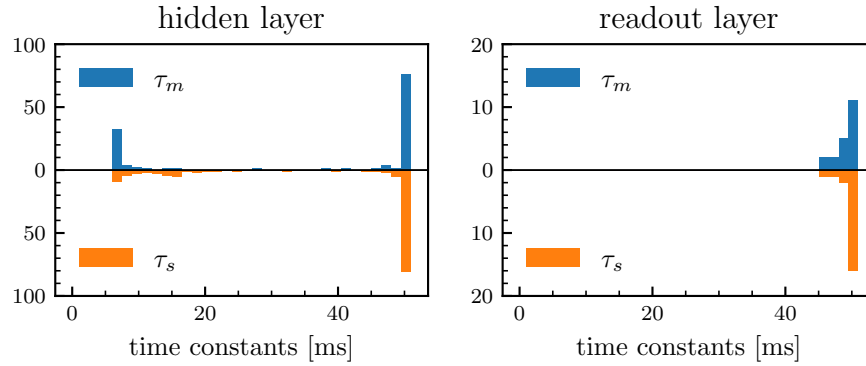


Figure 4.15: Distribution of time constants after training the recurrent model in the hidden as well as the readout layer for one seed.

5 Discussion

This work explores the possibility of gradient-based learning not only the synaptic weights, but also the neuron parameters on the mixed-signal neuromorphic system BSS-2. The training framework `hxtorch` (Spilger et al., 2023), that is based on PyTorch (Paszke et al., 2017) can be used to train models on the BSS-2 system as well as only in simulation.

This work extends the framework to enable the training of any neuron parameter. The experiment presented in Section 3.1.1, which involves identifying the correct parameterization of a target neuron trace, demonstrates the successful learning of the synaptic and the membrane time constant in simulation. With this confirmation, the results from Perez-Nieves et al. (2021) are reproduced. They demonstrate the benefits of learning both the synaptic and membrane time constants in addition to weights across different datasets. However, the reported test accuracy of $(82.7 \pm 0.8)\%$ on the SHD dataset is not reached for the model that trains the time constants in addition to the weights. After 50 epochs of training this work reaches a test accuracy of $(75.9 \pm 2.5)\%$ for this model. A reason for this may be, that the data augmentation used by Perez-Nieves et al. (2021) is not implemented in this work and data augmentation proves very useful for the test accuracy on the SHD dataset (Nowotny et al., 2025; Cramer et al., 2022). The results are in line with the results presented by Nowotny et al. (2025), who instead of surrogate gradients use the Event-Prop algorithm (Wunderlich and Pehle, 2021) to determine the gradients. The membrane time constant distribution after training looks qualitatively similar to Perez-Nieves et al. (2021), leading to the assumption that the training of time constants also works for a machine learning task in the `hxtorch` framework.

After this result, first steps towards emulation of the model on BSS-2 are performed. The model that is used on BSS-2 has to be adjusted to meet the hardware limitations. The fan-in of the hidden neurons must be reduced to enable the use of 128 neurons in the hidden layer. In order to do so, the recurrent model is simplified to a feedforward model. This reduces the fan-in of each hidden neuron and additionally reduces the complexity of the network. The simulation results show that this can be done without a large loss in accuracy, when learning weights and time constants on the SHD task. Additionally, the

benefit of learning time constants is even more apparent when using a feedforward network. Next, the input channels stemming from the SHD dataset are subsampled from 700 to only 70 remaining channels. This enables the placement of a feedforward, as well as a recurrent network with 128 neurons in the hidden layer on a single BSS-2 chip. Despite the accuracy loss induced by input subsampling, heterogeneity of the time constants proves effective in narrowing the resulting performance gap. Lastly, simulation tests regarding the time resolution are performed. From these results, it can be concluded that the chosen simulation time step and the binning interval of the spike grid do not have an impact on the performance of the model. For this reason, these two hyperparameters are chosen similarly to Cramer et al. (2022) for the training on BSS-2.

To further be able to train in-the-loop on BSS-2, the synaptic and the membrane time constant are characterized, since recalibrating the system every training step would not be feasible. The characterization of the membrane time constant includes measuring the individual membrane time constants for different leak conductances or capacitances on one chip. Then, a function is fit to the measurements to enable a fast mechanism to set the correct hardware parameter for the corresponding model value. Varying the leak conductance, unfortunately has side-effects on the leak potential due to imperfect manufacturing of the circuits. There is a solution that still uses the leak conductance as a hardware parameter to train the membrane time constant. This involves a two-dimensional translation function, incorporating both the CapMem value for the leak conductance as well as the CapMem value for the leak potential. This is being explored in parallel to this thesis in a greater context of a fast look-up based calibration (Hinterding, 2025). The other solution involving the characterization of the membrane capacitance does not exhibit such side-effects and is used in training.

The measurements of the synaptic time constant exhibit a huge inter-neuronal spread especially for low CapMem values corresponding to low conductances and large synaptic time constants. The ideal translation therefore cannot fit all neurons on the chip, which may be one of the problems that is responsible for the difference in validation accuracy between simulation and emulation on BSS-2 when training time constants on the SHD task. Additionally, the model

values are capped to stay within a certain range. Using the ideal translation, this means that the CapMem values, responsible for the synaptic time constant are also capped. As high synaptic time constants in the readout layer seem to be important for the validation accuracy, this poses a problem for example when neurons on BSS-2 only exhibit a time constant of $30\text{ }\mu\text{s}$ instead of the $50\text{ }\mu\text{s}$ used in simulation.

The second challenge that, unfortunately, is not completely solved, is the saturation of the membrane trace in the readout neurons on BSS-2. These high membrane traces are primarily caused by the max-over-time loss that is used. Another option would be to use a sum-over-time loss, which was already used when training on BSS-2 using the SHD dataset (Cramer et al., 2022). Without data augmentation and a model with 186 recurrently connected hidden neurons, Cramer et al. (2022) achieved a test accuracy of $(76.2 \pm 1.3)\%$ on BSS-2, which is significantly higher than the achieved validation accuracy of $(65.0 \pm 2.5)\%$ for the feedforward network with neuron training in this work. Here it must be noted, that this work uses the validation accuracy to estimate the accuracy on the test set, as the test set should only be used once at the very end to estimate how your final model will perform on completely new data. The sum-over-time loss has a few advantages over the max-over-time loss. Nowotny et al. (2025) show that the sum-over-time loss is inherently better suited for the SHD task. This can also be concluded from the results in this work: when training time constants, primarily in the case of the feedforward network, both time constants of all neurons in the readout layer shift towards the upper limit. This allows the membrane of each readout neuron to integrate all incoming spikes without much leakage. Combining this with taking the maximum value over the whole integration time, effectively replicates a sum-over-time loss. A second advantage of the sum-over-time loss is that the membrane traces in the readout layer are not necessarily pulled towards these high values as in the case of the max-over-time loss. Consequently, the membrane voltages in the hidden and readout layer stay within the same range. This allows to use similar configurations of all neurons without any further modification on BSS-2. This direction is not further explored in this work, as the motivation is not to reach state-of-the-art performance on the SHD task, but rather to enable learning of neuron parameters on BSS-2 and confirming

the result presented by Perez-Nieves et al. (2021) using the max-over-time loss. Despite the fact that neural heterogeneity improves performance on the SHD task, Perez-Nieves et al. (2021) further show that neural heterogeneity also improves robustness of the network. The robustness of networks with this kind of neural heterogeneity is further supported by Golmohammadi and Tetzlaff (2024). Unfortunately, the robustness of the networks is not investigated on BSS-2 in this work, as it requires a stable training that reaches accuracies that are comparable to those reached in simulation.

In conclusion, this work enables the learning of neuronal parameters on BSS-2. This is demonstrated by training the synaptic and the membrane time constant using an imperfect ideal translation function in a simple learning example on a neuron trace (Section 3). Furthermore, the benefits of training time constants on the SHD task are demonstrated. Several simulation experiments, analyzing the effects of model simplification, highlight the performance improvements achieved when both time constants are trained. Even though both models on BSS-2 are not able to reach the performance achieved in simulation, training neuron parameters proves beneficial on BSS-2 as well.

6 Outlook

This thesis should be considered as a first investigation of learning neuron parameters on the BSS-2. Therefore, many aspects presented in this thesis have the potential to be improved upon. This section discusses feature wishes for `hxtorch`, that help debugging a model when training on BSS-2. Additionally, it provides an outlook on ongoing work within the Electronic Vision(s) research group that is able to improve parameter learning, as well as explores additional capabilities enabled by learning parameters.

Even though `hxtorch` provides an accessible approach to gradient-based training of SNNs on the neuromorphic BSS-2 chip, it is still difficult to train a new model on BSS-2 without expert knowledge. This work involved many steps on figuring out the transition from a working model in simulation to a working model on BSS-2, as the large configurability of the hardware also comes with a large amount of parameters that influence this transition. Despite extensive investigation, the discrepancy between the performance in simulation and on BSS-2 could not be explained completely. As the model parameters are used for the calculation of gradients during backpropagation and the model in simulation trains correctly, it is best if the hardware emulation matches the simulation. Therefore, it is useful to investigate the discrepancy between hardware emulation when debugging the model. For this reason, the following part will explain typical debugging steps when training a SNN on BSS-2 and then continue with features that would facilitate debugging the training on BSS-2. The first step involves plotting both traces for each neuron exposed to a single spike to investigate whether they are identical, i.e. the trace and weight scaling are chosen correctly, according to the hardware configuration. This step does not have to happen manually, but can also be automatized for a provided hardware configuration before each training. This is already a work in progress. Assuming that weight and trace scaling are correct, this comparison of emulation and simulation trace is also useful during training. From this, three things can be observed directly: CapMem crosstalk, which causes both traces to no longer match anymore, saturation of the membrane traces on BSS-2 and side-effects, when changing a hardware parameter affects another parameter. An example of the latter is the shift of the leak potential due to a

change in the leak conductance which can be seen in a LIF neuron that fires, as there is a relative change between leak, threshold and reset potential. Additionally, it can be interesting to plot the voltage in the synaptic integrator circuits to see if these saturate. This can happen especially for long synaptic time constants. Unfortunately, it is not possible to record both the membrane trace and the synaptic input at once. Therefore, a slower debugging mode for training on BSS-2 is proposed. This records and plots all mentioned traces in multiple hardware runs to give feedback to the user. Another interesting feature for hxtorch would be an automatic-range neuron, that configures the hardware parameters according to the simulation. In this work a considerable amount of effort was invested into trying to scale the readout layer against the hidden layer on BSS-2. This was motivated by the fact that in simulation, without any regularizing terms, the maximum membrane potential in the readout layer is approximately 40 times higher than the threshold voltage in the hidden layer at the end of training. In the used hardware configuration, without further modifications, a factor of approximately 2.5 is possible on BSS-2. The synaptic strength on BSS-2 is however not only influenced by the 6 bit weight, but also by two other hardware parameters. Instead of using the trace and weight scale to adjust the range on hardware, as it is done in this work, these two parameters can be adjusted in combination with the trace scale, to define the range of readout neurons according to the simulation. This would enable users to utilize the entire dynamic range of the system with the highest possible resolution without detailed knowledge about the hardware.

As already mentioned in earlier sections, there is an on-going investigation in a fast lookup-based calibration. Hinterding (2025) investigates the dependencies between the leak potential and leak conductance, creating a multi-dimensional transformation from model value to hardware configuration. Other parameters will follow. In future work, this can be used instead of the imperfect translation function, presented in this thesis. As the leak conductance could be used without side-effects on the training, the capacitor could be held at maximum capacity for the lowest SNR, while still allowing for the parameter range defined by the capabilities of the hardware. Furthermore, this would decrease the parameter mismatch between hardware emulation and simulation and therefore also negate the problem of synaptic time constants being trained towards

the upper limit. With this fast calibration, it would then also be interesting to train a best-effort model on SHD, including training of the time constants on BSS-2. From the results presented in this work, this would imply using the sum-over-time loss instead of the max-over-time loss, as it is better suited for the task and the hardware limitations. Nonetheless, this direction was not pursued further, as the focus was on reproducing the results of Perez-Nieves et al. (2021) on BSS-2. Another interesting direction for future work would be to apply neuron parameter learning to other tasks with temporal structure. The DVS128 Gesture dataset (Amir et al., 2017), featuring visual stimuli from a neuromorphic vision sensor, represents a suitable benchmark for such investigations.

This thesis primarily focused on demonstrating the benefits of training neuron parameters on a machine learning task. However, there are additional scenarios where learning parameters can be advantageous. In particular, further extending the set of learnable parameters in BSS-2 would enable the system to be calibrated through gradient-based learning. Examples of how such a calibration can be performed are demonstrated in this thesis by learning time constants from a given neuron trace. Since this approach involves finding an appropriate parameterization for a specific neuron response, gradient-descent could also be applied to more complex, i.e. experimentally recorded, neuron traces (Vanier and Bower, 1999). A parameter search for traces of an AdEx neuron has already been carried out in the Electronic Vision(s) group using simulation-based inference (Kaiser et al., 2023; Huhle et al., 2024).

Acknowledgements

I would like to thank:

- Prof. Dr. Johannes Schemmel for giving me the opportunity to write my thesis in the Electronic Vision(s) Group.
- Dr. Eric Müller for the close supervision of my thesis and welcoming me into the group. I really enjoyed all the Type-2-Fun conversations about all the Type-1-Fun activities and the Type-2-Fun activities, which are actually more fun.
- Philipp Spilger for his unparalleled supervision throughout this entire time. Thank you for helping me with all my questions and ideas in such a supportive way. Thank you, for all the interesting on- and off-topic conversations. Your supervision made this thesis not only an inspiring academic experience but also a very enjoyable journey.
- Jakob Kaiser, Elias Arnold and Yannik Stradmann for all your ideas and support in solving software- and hardware-related issues.
- the entire group for a welcoming work atmosphere.
- Tim Auberer, Amani Atoui and Ronja Hinterding for being great desk mates, so that it was always a pleasure to come to the office.
- all the darts players that made the lunch break so much fun. I would like to especially thank Simon Rosenkranz for all the after-work try-hard sessions.
- Florian Fischer and Tim Auberer for all the fun “Geheimmissionen” we completed.
- my family, for always being there for me. Thank you, for your never ending support and for encouraging me to study physics: “Hauptsache gesund”.
- Catha for everything you give me in life. I am deeply grateful for your unconditional support, for your patience in listening to my struggles, and for your uplifting encouragement throughout.

The work carried out in this Master's thesis used systems, which received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 720270, 785907 and 945539 (Human Brain Project, HBP) and Horizon Europe grant agreement No. 101147319 (EBRAINS 2.0).

References

- Edgar D Adrian and Yngve Zotterman. The impulses produced by sensory nerve-endings: Part ii. the response of a single end-organ. *The Journal of physiology*, 61(2):151, 1926.
- Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.
- Elias Arnold, Georg Böcherer, Florian Strasser, Eric Müller, Philipp Spilger, Sebastian Billaudelle, Johannes Weis, Johannes Schemmel, Stefano Calabrò, and Maxim Kuschnerov. Spiking neural network nonlinear demapping on neuromorphic hardware for IM/DD optical communication. *Journal of Lightwave Technology*, 41(11):3424–3431, 2023. doi: 10.1109/JLT.2023.3252819.
- Amani Atoui, Jakob Kaiser, Sebastian Billaudelle, Philipp Spilger, Eric Müller, Jannik Luboeinski, Christian Tetzlaff, and Johannes Schemmel. Multi-timescale synaptic plasticity on analog neuromorphic hardware. *arXiv preprint arXiv:2412.02515*, 2024.
- Bruce P Bean. The action potential in mammalian central neurons. *Nature Reviews Neuroscience*, 8(6):451–465, 2007.
- Sebastian Billaudelle. *From transistors to learning systems: circuits and algorithms for brain-inspired computing*. PhD thesis, Universität Heidelberg, 2022.
- Sebastian Billaudelle, Johannes Weis, Philipp Dauer, and Johannes Schemmel. An accurate and flexible analog emulation of adex neuron dynamics in silicon. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, 2022. doi: 10.1109/ICECS202256217.2022.9971058.

- Alexandre Bittar and Philip N Garner. A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, 16:865897, 2022.
- Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642, 2005.
- JS Coombs, DR Curtis, and JC Eccles. The interpretation of spike potentials of motoneurons. *The Journal of Physiology*, 139(2):198, 1957.
- B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke. The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2020. ISSN 2162-2388. doi: 10.1109/TNNLS.2020.3044364.
- Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022.
- Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham China, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.
- Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.
- Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.
- Arash Golmohammadi and Christian Tetzlaff. Robust computation with intrinsic heterogeneity. *arXiv preprint arXiv:2412.05126*, 2024.
- Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter

- Senn, Johannes Schemmel, et al. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature machine intelligence*, 3(9):823–835, 2021.
- Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. *arXiv preprint arXiv:2306.17670*, 2023.
- Trevor Hastie. The elements of statistical learning: data mining, inference, and prediction, 2009.
- Ronja Hinterding. Towards a multidimensional calibration of neuromorphic hardware using a parameter transformation model. Bachelor’s thesis, Universität Heidelberg, 2 2025.
- Matthias Hock. *Modern semiconductor technologies for neuromorphic hardware*. PhD thesis, Universität Heidelberg, July 2014.
- Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.
- Jakob Huhle, Jakob Kaiser, Eric Müller, and Johannes Schemmel. Reproduction of adex dynamics on neuromorphic hardware through data embedding and simulation-based inference. *arXiv preprint arXiv:2412.02437*, 2024.
- David Iberri. File:action potential.svg, 2007. URL https://upload.wikimedia.org/wikipedia/commons/4/4a/Action_potential.svg. Licensed under CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>).
- Jakob Kaiser, Sebastian Billaudelle, Eric Müller, Christian Tetzlaff, Johannes Schemmel, and Sebastian Schmitt. Emulating dendritic computing paradigms on analog neuromorphic hardware. *Neuroscience*, 489:290–300, 2022. ISSN 0306-4522. doi: <https://doi.org/10.1016/j.neuroscience.2021.08.013>. URL <https://www.sciencedirect.com/science/article/pii/S0306452221004218>. Dendritic contributions to biological and artificial computations.

- Jakob Kaiser, Raphael Stock, Eric Müller, Johannes Schemmel, and Sebastian Schmitt. Simulation-based inference for model parameterization on analog neuromorphic hardware. *Neuromorphic Computing and Engineering*, 3(4):044006, 2023.
- Eric R. Kandel, John D. Koester, Sarah H. Mack, and Steven A. Siegelbaum. *Principles of Neural Science*, 6e. McGraw Hill, 2021.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- L Lapicque. Recherches quantitatives sur l’excitation électrique des nerfs. *Journal de Physiologie et Pathologie General*, 9:620–635, 1907.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL <https://doi.org/10.1038/nature14539>.
- Eric Müller, Elias Arnold, Oliver Breitwieser, Milena Czierlinski, Arne Emmel, Jakob Kaiser, Christian Mauch, Sebastian Schmitt, Philipp Spilger, Raphael Stock, et al. A scalable approach to modeling on accelerated neuromorphic hardware. *Frontiers in neuroscience*, 16(884128), may 2022. doi: 10.3389/fnins.2022.884128.
- 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. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. doi: 10.1109/MSP.2019.2931595.
- Thomas Nowotny, James P Turner, and James C Knight. Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *Neuromorphic Computing and Engineering*, 5(1):014001, 2025.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric

- Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022.
- Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *Nature communications*, 12(1):5791, 2021.
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL <https://doi.org/10.1038/323533a0>.
- Johannes Schemmel, Sebastian Billaudelle, Philipp Dauer, and Johannes Weis. Accelerated analog neuromorphic computing. In *Analog Circuits for Machine Learning, Current/Voltage/Temperature Sensors, and High-speed Communication: Advances in Analog Circuit Design 2021*, pages 83–102. Springer, 2021.
- Sebastian Schmitt, Johann Klähn, Guillaume Bellec, Andreas Grübl, Maurice Güttler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, Vitali Karasenko, Mitja Kleider, Christoph Koke, Alexander Kononov, Christian Mauch, Eric Müller, Paul Müller, Johannes Partzsch, Mihai A. Petrovici, Stefan Schiefer, Stefan Scholze, Vasilis Thanassoulis, Bernhard Vogginger, Robert Legenstein, Wolfgang Maass, Christian Mayr, René Schüffny, Johannes Schemmel, and Karlheinz Meier. Neuro-morphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234, 2017. doi: 10.1109/IJCNN.2017.7966125.
- David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

- Philipp Spilger, Elias Arnold, Luca Blessing, Christian Mauch, Christian Pehle, Eric Müller, and Johannes Schemmel. hxtorch. snn: Machine-learning-inspired spiking neural network modeling on brainscales-2. In *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference*, pages 57–62, 2023.
- Frédéric Theunissen and John P Miller. Temporal encoding in nervous systems: a rigorous definition. *Journal of computational neuroscience*, 2:149–162, 1995.
- US-Federal. File:neuron, langneutral.svg, 2006. URL https://upload.wikimedia.org/wikipedia/commons/0/08/Neuron%2C_LangNeutral.svg. Licensed under CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>), Originally taken from US National Cancer Institute’s Surveillance, Epidemiology and End Results (SEER) Program.
- Michael C Vanier and James M Bower. A comparative survey of automated parameter-search methods for compartmental neural models. *Journal of computational neuroscience*, 7:149–171, 1999.
- Johannes Weis. Inference with artificial neural networks on neuromorphic hardware. Master’s thesis, Universität Heidelberg, 9 2020.
- Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):12829, 2021.
- Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6):1514–1541, 06 2018. ISSN 0899-7667. doi: 10.1162/neco_a_01086. URL https://doi.org/10.1162/neco_a_01086.

Acronyms

AdEx adaptive exponential leaky integrate and fire. 1, 16, 69

ANN artificial neural network. 1, 10, 22

API application programming interface. 19, 25

BP backpropagation. 13, 14

BPTT backpropagation through time. 13, 14

BSS-2 BrainScales-2. i, iii, 1–4, 14–22, 25, 27–29, 33, 38, 41, 43, 46, 50, 51, 53–58, 60, 61, 63–69, 86, 87

CADC columnar analog to digital converter. 15, 20–22, 43, 50, 52, 54, 55

CapMem capacitative memory. 17, 18, 30–32, 38, 39, 52–54, 59, 64, 65, 67, 83, 84

DRAM dynamic random access memory. 16, 43, 44

FPGA field programmable gate array. 15, 16, 43

HD Heidelberg Digits. 22

LI leaky integrator. 8, 17, 19, 23, 25, 30, 39, 52

LIF leaky integrate and fire. 8, 9, 16–19, 21, 23, 52, 68, 82

LSB least significant bit. 20, 54, 55

MADC membrane analog to digital converter. 15, 30

MSE mean squared error. 27

OTA operational transconductance amplifier. 16–18, 37, 52

PSP postsynaptic potential. 6, 7

RNN recurrent neural network. 11–13

RSNN recurrent spiking neural network. 23, 24, 42, 84

SHD Spiking Heidelberg Digits. i, iii, 2, 3, 22–24, 41–46, 48, 49, 56–58, 63–66, 69, 84

SIMD single instruction, multiple data. 15, 16, 43

SNN spiking neural network. i, iii, 1–3, 12, 13, 22, 23, 31, 56, 67, 82

SNR signal-to-noise ratio. 52, 68

SRAM static random access memory. 17, 43

Appendices

A Hardware Parameters

The experiments with time constant training conducted in the Section 4 used the target parameters shown in Table 3 for the underlying calibration. They used the setup W70F0.

Table 3: Target parameters used for calibration on BrainScaleS-2 as well as calibration specific parameters used in training.

Parameter	Value
<code>tau_mem</code>	50 μ s
<code>tau_syn</code>	10 μ s
<code>threshold</code>	150
<code>leak</code>	80
<code>reset</code>	80
<code>i_synin_gm</code>	50
<code>synapse_dac_bias</code>	1000
<code>membrane_capacitance</code>	63
<code>refractory_time</code>	2 μ s
<code>trace_scale</code>	1×10^{-3}
<code>weight_scale</code>	500

B Software State

Experiments conducted in this thesis used the software state in Table 4. The used container is given in Table 5.

Table 4: Software state

Repository	Commit-Hash
hxtorch	699e4a9fd42a0fc12498c45482e17865eed42341
hxcomm	8fa8bdb90076249e639b6ec06c7754161f69d14b
haldls	8a58083ad384ceea6a026a20ad07a4395252db3f
grenade	5702a628eea32b9dd01474949626ddb288f4e4ce
hate	35b3cb211cabbbc5c01036ae7878a73e338166c4
calix	a706868c6ba285b1f8fd7cdef1a19d7328e02912
sctrltp	1d854f953f7e8c8ead44406a22bb80421ca3857c
rant	53199ee94cae1e1c2e4db10e88d570a761b14e0f
hwdb	21643473ceafa07f38eb2f40e9312fb6a520cdb9
code-format	f4ef916fde2ca6d67e27fe9b3b5807ba69704e93
logger	73dadb3ce413c521845ef7d36f818073eee4fefaf
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3
flange	28e729d59df3b4ff380f84351c40d4da3086bed8
lib-rcf	000185eb11db4d54cb6b12b09af54cf742741036
bss-hw-params	b7be7827b51536804f0bda76f8ba4be693df23a8
halco	d03f343231644067e41bdd0fda267c4070123664
fisch	f83c5f658eebf574a738e0d00441f66c27ad9cbe
linux	31b76437ff4754bcdaaaecffc3d24aecdb3b9fa5a
pywrap	5e2af30e9593882b471d3cd02df00b93f13ff479
ztl	c2d4faee05f497010ee55e35bf9c9607eeebf884
lib-boost-patches	136c5b41cb046afe2c726aa4646928bf5190622e

Table 5: Container

Path	/containers/stable/2025-02-19_1.img
Fingerprint	64d91623-6dcd-43b5-886a-2d5eca67eff6
App	dls

List of Figures

- 2.1 (a) The schematics of a neuron with “a” being the dendrites, “b” representing the representing the soma including the cell nucleus “c”. The axon begins at “d” and spreads out to axon terminals “h” at the end. The axon is covered by a myelin sheath “e” of the Schwann cell “f”. Each small part where the axon is not covered by myelin is called a node of Ranvier “g”. (b) In case a current stimulus depolarizes the membrane potential, causing it to pass a certain threshold, an action potential is generated. Typically, this is followed by a refractory period, where the cell is not excitable. 5

2.2	Process behind a signal transmission of a chemical synapse. Figure taken from Kandel et al. (2021). Copyright © McGraw-Hill Education. Reprinted with permission.	7
2.3	Schematic of a fully connected single layer feedforward architecture. The inputs \mathbf{x} are weighted and summed over using the weight matrix \mathbf{W}_1 . After applying the activation function, the outputs of the hidden layer are forwarded to the output layer using the the weights \mathbf{W}_2 , where again an application function is applied.	11
2.4	Computational graph of an SNN in discrete time. The input spikes z_0 are propagated through the network from bottom to top. The synaptic temporal updates of the synaptic current I and membrane potential v are enrolled on the horizontal axis for two time steps. The topmost layer, represented by the dashed boxes can either be an additional hidden layer or a readout layer. The latter is categorized as a single hidden layer spiking neural network.	12
2.5	Overview of BrainScaleS-2 system. A : Neuromorphic chip on a carrier board. B : Abstract schematic of the chip design. C : Simple schematic of individual neuron circuits and connections. Figure kindly provided by Jakob Kaiser from (Kaiser et al., 2022).	15
2.6	Block schematic of a neuron circuit implementing the LIF model. Figure adapted from Billaudelle et al. (2022)	16
2.7	Transformed audio of a spoken “eins”, a spoken “one” in German, into spike times.	23

3.1	Training run of time constants on a neuron trace: Neuron trace at different training epochs (left). Loss-landscape with the time constant evolution during training (right). For the training, a batch of 100 different target membrane traces are simulated using the parameters $\tau_s = 20 \mu\text{s}$ and $\tau_m = 10 \mu\text{s}$, of which one trace is shown on the left. In the right figure, these target parameters are represented by the green dot. At epoch 0, the training starts with parameters $\tau_s = 17 \mu\text{s}$ and $\tau_m = 3 \mu\text{s}$, represented by the black dot on the right. During training the time constants are then optimized according to equation (3.1).	27
3.2	Measurement of the membrane time constant τ_m using an offset current. The membrane potential is offset by the offset current until it is switched off at $50 \mu\text{s}$ leading to an exponential decay of the membrane potential.	31
3.3	Measurement of the membrane time constant τ_m for a leak conductance sweep with and without enabling conductance division. When division is enabled longer time constants for the same CapMem value can be observed.	32
3.4	Measurement of the membrane time constant for a leak conductance sweep with and without enabling conductance division for all 512 neurons. A fit is performed on the measurements of all 512 neurons to find the ideal translation.	33
3.5	Training of membrane time constant on neuron trace using the leak conductance: neuron trace at different training epochs (left) and the evolution of the model value of the membrane time constant (right).	34
3.6	Measurement of the membrane time constant for a membrane capacitance sweep for a single neuron (left) and all 512 neurons (right). This is done for two different calibrations of the membrane time constant.	35
3.7	Training membrane time constant on trace with membrane capacitance: membrane trace for different epochs (left). Membrane time constant evolution over epochs (right).	36

3.8	Measurement of the synaptic time constant τ_s . A spike event happens at $8\mu\text{s}$, after which the capacitor relaxes towards its resting potential, following an exponential decay with the synaptic time constant τ_s . An exponential decay is fit to the measurements.	38
3.9	Measurement of the synaptic time constant τ_s for different conductance configurations of a single synapse. The inhibitory and excitatory circuit exhibit approximately the same synaptic time constant for the same CapMem values.	39
3.10	Measurement of the synaptic time constant τ_s for different conductance configurations of all synapses with corresponding ideal translation: excitatory synapses (left) and inhibitory synapses (right).	40
3.11	Training of the synaptic time constant on hardware: membrane trace at different epochs (left) and evolution of the synaptic time constant over the training epochs with the target model value (right).	40
4.1	SHD classification mean test accuracies during training of a RSNN when training only the weights (red curve) and when training weights and neuron parameters (blue curve) across 10 seeds. The shaded area represents the standard deviation. The reported results achieved by Perez-Nieves et al. (2021) of the corresponding models are shown as dashed lines in the respective color.	42
4.2	Histograms of the time constant distribution in the hidden and the readout layer after training for both the membrane time constant τ_m and the synaptic time constant τ_s for one random seed.	42
4.3	Validation accuracy during training on the SHD task for a feed-forward (red) and a recurrent (blue) network when training only weights (top) and when training weights and time constants (bottom) in simulation. The mean validation accuracy of runs across 10 different seeds is represented by the curve, while the shaded area around the curve corresponds to the standard deviation.	45

4.4	Validation accuracy during a training of a recurrent network when training weights and neuron parameters for each of the ten seeds separately.	47
4.5	Comparison of learned time constant distributions after training between recurrent and feedforward network architecture for one random seed.	48
4.6	Comparison of the validation accuracy between different numbers of inputs (70 and 700) when training only weights (red, yellow) and when training weights and neuron parameters (blue, green).	49
4.7	Comparison of the mean validation accuracy for different time resolutions when training weights (orange) and when training neuron parameters in addition to the weights (blue). The standard deviation across 10 seeds is shown as the black error bar. .	50
4.8	Comparison of learned time constant distributions after training between different time steps for one seed, shown for the hidden and readout layer separately.	51
4.9	Mean validation accuracies across 13 different seeds, when training weights only (red) and when training weights and time constants (blue). The training of each seed is shown as a shaded line.	58
4.10	Maximum voltage of the membrane in the readout layer over all training epochs for the different seeds (left) and a scatter plot of the maximum voltage and the validation accuracy at the last epoch of training for all seeds (right). The maximum voltage rises during training for both models. Especially for the homogeneous network a correlation between the maximal voltage and the validation accuracy can be observed.	59
4.11	Distribution of time constants after training in the hidden as well as the readout layer for one seed.	60
4.12	Distribution of weights after training in the hidden as well as the readout layer for one seed.	60

4.13	Mean validation accuracy of the recurrent model for 10 different seeds, when training weights only (red) and when training weights and time constants (blue). The training of each seed is shown as a shaded line.	61
4.14	Validation accuracy and maximum voltage in output layer for one seed where accuracy drops can be observed.	62
4.15	Distribution of time constants after training the recurrent model in the hidden as well as the readout layer for one seed.	62

List of Tables

1	Network parameters from Perez-Nieves et al. (2021).	24
2	Used model network parameters on BrainScaleS-2.	57
3	Target parameters used for calibration on BrainScaleS-2 as well as calibration specific parameters used in training.	80
4	Software state	81
5	Container	81

List of Listings


1	Construction of an arbitrary spiking feedforward network using <code>hxtorch</code> . The construction works similar to PyTorch: The model inherits from <code>torch.nn.Module</code> and initializes all required layers in the constructor. The forward method defines, how inputs are propagated through the network. When called, the first four lines of code create a graph representation of the network in the experiment instance, required for execution on BSS-2. Calling <code>hxsnn.run(...)</code> , then emulates the network on BSS-2 and or simulates the dynamics the network.	19
2	Class definition of an <code>HXBaseParameter</code> . The <code>HXBaseParameter</code> is initialized with its BSS-2 and model representation and holds these as properties.	20

3	Extension of <code>HXBaseParameter</code> for training parameters in software. The <code>HXBaseParameter</code> inherits from <code>torch.nn.Module</code> so that trainable parameters are recognized by the optimizer. The <code>make_trainable()</code> instantiates the model parameter as <code>torch.nn.Parameter</code>	26
4	Example on how to achieve trainability of neuron parameters using the model from listing 1. After initializing the model, the <code>make_trainable()</code> function can be called on the parameter that should be trained. In this case the membrane time constant is trained. This call can already happen in the model initialization.	26
5	Extension of <code>HXBaseParameter</code> for training parameters on BSS-2. The ideal translation is provided by the user in form of the <code>set_on_chip</code> function, that sets the configuration corresponding to the parameter translation.	29

Declaration

I confirm that the work for this Master thesis was solely undertaken by myself and that no help was provided from other sources as those allowed. All sections of the work that use quotes or describe an argument or concept developed by another author have been referenced, including all secondary literature used, to show that this material has been adopted to support my thesis.

Heidelberg, April 15, 2025

A handwritten signature in black ink, reading "S. Jonscher". The signature is written in a cursive style with a small square highlight behind the end of the name.

Simon Jonscher