Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

Simeon Kanya

born in Graz, Austria

2020

Deep Learning

on

Analog Neuromorphic Hardware

This Master thesis has been carried out by Simeon Kanya

at the

Kirchhoff Institute for Physics in Heidelberg, Germany

under the supervision of

Dr. Johannes Schemmel

Abstract

Neurons and synapses are the biological basis for information flow and processing in the brain. By forming spiking neural networks, the brain is able to learn and remember. These networks have not only provided an insight into the brain's mechanisms but offer an alternative to artificial neural networks in deep learning. When implemented on analog neuromorphic hardware, they inherit a variety of favorable properties from their biological counterpart, such as parallel and event based information processing, noise robustness and a high energy efficiency. These characteristics are of great interest, since using conventional deep learning for real-world problems requires the resources of large supercomputer clusters. However, modern learning algorithms for neuromorphic hardware still have a hard time competing with the great success of deep learning. In this thesis, the performance of two candidates for training spiking neural networks is evaluated on the analog neuromorphic hardware platform BrainScales2. First, a rate-based spiking neural network is trained using gradient descent as a standalone on-chip experiment. Second, a surrogate gradient algorithm, SuperSpike, is implemented as an chip-in-the-loop experiment.

Zusammenfassung

Neuronen und Synapsen bilden die biologische Basis für den Austausch und die Verarbeitung von Informationen im Gehirn. Mithilfe dieser spikenden neuronalen Netzen kann das Gehirn lernen und sich erinnern. Diese Netze dienen nicht nur dazu die Vorgänge im Gehirn zu veranschaulichen, sondern sind auch eine Alternative zu Deep Learning mit künstlichen neuronalen Netzen. Im Einsatz auf analoger neuromorpher Hardware übernehmen spikende neuronale Netze einige vorteilhafte Eigenschaften ihrer biologischen Vorbilder, wie zum Beispiel paralleles und eventbasiertes Verarbeiten von Informationen, eine erhöhte Fehlertoleranz sowie eine hohe Energieeffizienz. Unter dem Gesichtspunkt, dass die Ressourcen eines Höchstleistungsrechners benötigt werden, um mit klassischem Deep Learning echte Probleme lösen zu können, sind die zuvor genannten Leistungsmerkmale besonders interessant. Allerdings können sich moderne Lern-Algorithmen für neurmorphe Hardware noch nicht mit dem Erfolg von künstlichen Neuronalen Netzen auf konventioneller Hardware messen. In dieser Arbeit wird die Funktion und Performance von zwei Algorithmen auf der analogen neuromorphen Plattform BrainScales2 evaluiert. In einem ersten Experiment wird ein raten-basiertes spikindes neuronales Netz mit einem Gradientenverfahren in einer alleinstehenden "on-Chip" Implementierung trainiert. Das zweite Verfahren beruht auf einem Surrogate-Modell des Gradientenverfahrens, SuperSpike, welches als Chip-in-the-Loop Experiment durchgefhrt wird.

Contents

1 Introduction

Humankind has been striving for a conceptual understanding of the physical world, even long before Faust famously questioned his knowledge about what it is that holds the world together. With the scientific progress and gained knowledge over within the last decades, new innovation have emerged, that changed the earth into a new digitally interconnected place. Yet, the understanding of our own human brain remains elusive.

From a biological point of view, neurons are responsible for processing and transmitting information in the brain by forming spiking neural networks (SNNs) (*Gerstner et al.*, 2014). Studying these networks is promising approach to gaining better insight into the brain's mechanisms. Not only provide SNNs a guiding principle to study the brain, but are of great interest for an efficient implementation of deep neural networks on neuromorphic hardware (*Pfeiffer and Pfeil*, 2018). They inherit multiple favorable characteristics from their biological inspiration such as parallelized and event-based information processing, noise tolerance and a low power consumption.

The success of deep learning is followed by an ever growing demand of specialized computational resources (*Mayer and Jacobsen*, 2020). Besides building larger supercomputers using conventional hardware, more efficient solutions such as neuromorphic hardware have seen an increased interest. As of today, several well known tech companies have launched their own neuromorphic platforms. IBM started to work on *TrueNorth* in 2008 (*Akopyan et al.*, 2015), Intel presented the *Loihi* chip in 2018 (*Davies et al.*, 2018) and Google began selling the *Coral* dev-board in 2019 (*Cass*, 2019). Even before neuromorphic hardware caught the attention of the big industry names, academic projects had already been started. Among others, the EU's Human Brain Project (HBP) funds two promising approaches: *SpiNNaker*, a digital based neuromorphic supercomputer located in Manchester (*Furber et al.*, 2014) and *BrainScaleS (BSS)*, a mixed-signal accelerated emulation for spiking neural networks based in Heidelberg (*BrainScaleS*, 2012).

In recent years, the field of neuromorphic computing has focused on novel training algorithms for SNNs that can compete with the widely successful conventional deep learning methods. The main problem arises from the non-differentiability of individuals spikes making well proven optimization methods such as gradient descent inapplicable. The ideas for workarounds range from imitating artificial neural networks (ANNs) with rate coding to using surrogate gradients (*Tavanaei et al.*, 2019).

In this research two candidates for supervised training are presented and then implemented on different prototypes of the BrainScaleS2 (BSS2) platform. In a first approach, a SNN is emulated as a classical deep ANN on the prototype chip HICANN-DLSv2 using rate coding. The network is then successfully trained by an on-chip implementation of gradient descent, i.e. only on-chip resources are used to compute and apply parameter changes. A second experiment is conducted on a revision of the first full-size prototype of the BSS2 platform, the HICANN-Xv1. With an additional built-in observation feature of the membrane potential, a spiking variant of gradient descent called *SuperSpike* (*Zenke and Ganguli* (2018)) can be used to train a deep SNN. On the current revision a complete and high-performance on-chip implementation has not yet been feasible, due to several hardware bugs. Instead, SuperSpike is implemented by recording the analog neuron dynamics on-chip and processing them with the support of a host. Lastly, the challenges of the different learning methods and potential future improvements will be discussed.

2 Background

The required knowledge to work in the field of neuromorphic computing is broad and manifold, ranging from the biological view of the human brain further to machine learning algorithms. In the next sections, the main concepts and the physical background, upon which the presented research is based on is, are introduced. Starting with deep learning and an overview of the biological neuron, the transition to neuron models, neuronal coding schemes and their training approaches will be given before introducing the neuromorphic BSS2 platform. Throughout the thesis, vectors are indicated by an upright boldface $\mathbf{v}$ and matrices by capital letters M.

2.1 Deep Learning

The presented introduction to deep learning in this section is largely oriented on the well-known eponymous book from *Goodfellow et al.* (2016) and indicated respectively if otherwise.

Deep learning is among the most useful and powerful tools machine learning has provided to the scientific community. Image or pattern recognition are in general hard to solve for traditional computation concepts. Deep learning abstracts such task in terms of a hierarchy of concepts. Each concept is based upon a combination of simpler ones. Going down on a hypothetical ladder towards the easiest concept available, creates a deep structure with many layers. This is why it is called deep learning.

A popular example for deep learning is the multi layer perceptron (MLP), a deep feedforward network. As the name suggests, the information is forwarded from one layer to another (see fig. 2.1). At each layer l the input $\mathbf{x}^{(l)}$ is mapped to an output $\mathbf{y}^{(l)} = \Phi(\mathbf{x}^{(l)}, \theta^{(l)})$ with the activation function Φ and a set of parameters $\theta^{(l)}$. The output of the layer l then determines the input of the next layer, i.e. $\mathbf{y}^{(l)} = \mathbf{x}^{(l+1)}$, and so forth. The layer structure is inspired by biological neural networks and therefore they are often referred to as artificial neural networks (ANNs).

An ANN is trained by adapting the parameters $\theta^{(l)}$ for all layers l following a training algorithm. In machine learning, one discriminates between supervised, and unsupervised learning algorithms. Yet, drawing a consequent line to categorize machine learning methods is difficult, as the approaches are sometimes combined into hybrid forms

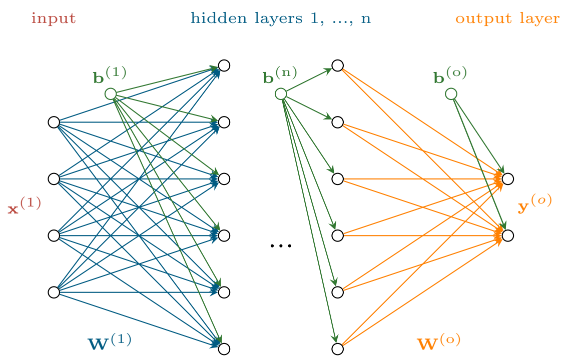


Figure 2.1: Deep artifical neural network. An artificial neural network (ANN) can n hidden layers will always have an input $\mathbf{x}^{(1)}$) and one output layer $y^{(o)}$. The sizes of the input and the individual layers can vary depending on a chosen problem.

too. Without a supervisor, an algorithm looks for structures and useful properties within the input data. To this end, unsupervised learning algorithms try to observe the underlying probability distribution of the input data. In supervised learning, on the other hand, each input vector x is associated with a target vector $\mathbf{y}^*$. In this case, the algorithm is trained to predict a target for a given input.

2.1.1 Supervised Training

The supervised training of an ANN can be divided into a *forward pass* where the output of all nodes is evaluated and a *backward pass* which computes the respective parameter updates. In the **forward pass**, the input $\mathbf{x}^{(l)}$ of a layer l sums to a net input $\mathbf{a}^{(l)}$ of

$$\mathbf{a}^{(l)} = W^{(l)}\,\mathbf{x}^{(l)} + \mathbf{b}^{(l)} + (\text{noise})\,,$$

with the weight matrix $W^{(l)}$, the bias $\mathbf{b}^{(l)}$ and an optional noise term. Depending on the task, the bias as well as the optional noise term can be vital: the individual biases for instance allow the network to adjust the dynamic range of each neuron and the injection of artificial noise can significantly increase the training performance (*An*, 1996).

The output $\mathbf{y}^{(l)}$ of the layer l is then given by the activation function ϕ, e.g. a sigmoid

$$\mathbf{y}^{(l)} = \phi(\mathbf{a}^{(l)}) = \frac{1}{1 + e^{(-\beta \mathbf{a}^{(l)})}},$$

with the slope parameter β. Popular choices for the activation function besides a sigmoid are a rectified linear unit (ReLu) or the hyperbolic tangent (tanh). In fig. 2.2a

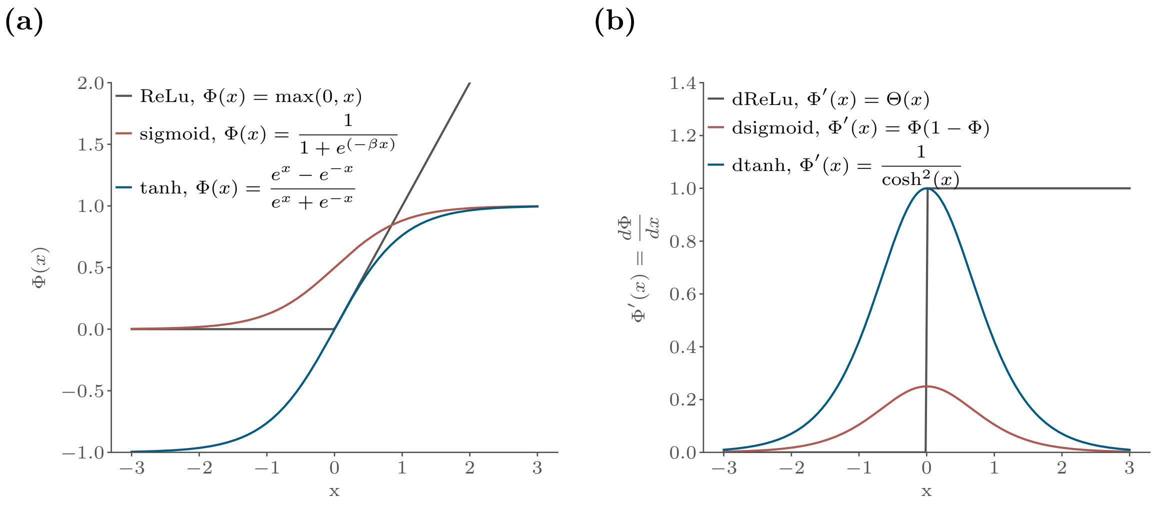


Figure 2.2: Popular shapes for activation functions in deep learning. **(a)**: Some of the most popular shapes for the activation function are a ReLu, tanh or sigmoid. **(b)**: The derivative of the different activation function gives an insight how the impact of gradient descent changes for different functions. In the case of the sigmoid and the hyperbolic tangent, either a very high or low input leads to a zero gradient and thus to a vanishing parameter update.

the various activation functions are shown for comparison. However, more important than the exact shape is that the chosen function is non-linear. A linear activation function makes any layer structure redundant, as the composition of linear linear functions yields again a linear function and therefore all layers of a deep network could be merged into a single one.

The same principle is then applied to all other layers to complete the forward pass, i.e. the result of the previous layer is the input for the current layer. The weight matrix $W^{(l)}$ connecting layer l with $l-1$ has the appropriate shape to fit the number of input nodes $n^{(\text{l-1})}_{\text{nodes}}$ and output nodes $n^{(\text{l})}_{\text{nodes}}$.

In deep learning, *gradient descent* is probably the most popular algorithm to perform the **backward pass** on a network. A differentiable loss function $\mathcal{L}(\mathbf{x}, \mathbf{y}^*, \theta)$ for a given target $\mathbf{y}^*$ is minimized. Here, a binary cross-entropy loss is chosen

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}^* \log(\mathbf{y}_i + (1-\mathbf{y}^*)\log(1-\mathbf{y}_i),$$

with a minibatch of size N and the target $\mathbf{y}^*$ scaled between 0 and 1. In combination with a sigmoid-shaped activation function, this choice becomes convenient when computing the new set of parameters θ'. This is done by moving along the negative gradient of the loss with respect to the network's parameters θ

$$\theta' = \theta - \eta \, \nabla_\theta \mathcal{L}(\mathbf{x}, \mathbf{y}^*, \theta), \tag{2.1}$$

with the learning rate η. To avoid extensive computational costs, the gradient is estimated by a uniformly drawn subset of the full training data set, a *minibatch.* In particular, if the size of the minibatch equals one, one speaks of stochastic gradient descent (SGD). As an example, the updates of the weight matrices of a single hidden layer network are computed in the next paragraphs using SGD.

First, the derivative of the cross entropy loss function in the output layer $l \equiv o$ is computed

$$\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(o)}} = -\frac{\mathbf{y}^*}{\mathbf{y}^{(o)}} + \frac{1-\mathbf{y}^*}{1-\mathbf{y}^{(o)}},$$

The gradient of the loss can then be rewritten in terms of the error $\mathbf{e}^{(o)} = \mathbf{y}^* - \mathbf{y}^{(o)}$ by using the derivative of the activation function

$$\frac{\partial \mathbf{y}^{(o)}}{\partial \mathbf{a}^{(o)}} = \frac{\partial \Phi(\mathbf{a}^{(o)})}{\partial \mathbf{a}^{(o)}} = \Phi(1-\Phi),$$

$$\Rightarrow \quad \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(o)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(o)}} \, \frac{\partial \mathbf{y}^{(o)}}{\partial \mathbf{a}^{(o)}} = \mathbf{y}^* - \mathbf{y}^{(o)} = \mathbf{e}^{(o)}.$$

According to eq. (2.1), the final update of the weight matrix is given by

$$\delta W^{(o)} = -\eta \frac{\partial \mathcal{L}}{\partial W^{(o)}} = -\eta \, \frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(o)}} \, \frac{\partial \mathbf{a}^{(o)}}{\partial W^{(o)}} = -\eta \, \left(\mathbf{e}^{(o)} \mathbf{x}^{(o),T} \right), \tag{2.2}$$

with the transpose of the input $x^{(o),T}$.

The computation for the hidden layer ($l \equiv h$) can be done in a similar fashion. Again, the gradient of the loss function is computed

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(h)}} = \mathbf{e}^{(h)} \, \frac{\partial \mathbf{y}^{(h)}}{\partial \mathbf{a}^{(h)}}.$$

The error of the output layer $\mathbf{e}^{(o)}$ is propagated backwards with the transpose of the weight matrix $W^{(o),T}$ as $\mathbf{e}^{(h)} = W^{(o),T} \mathbf{e}^{(o)}$ yielding a total update of

$$\delta W^{(\mathrm{h})} = -\eta \, \left(W^{(\mathrm{o})T} \mathbf{e}^{(o)} \right) \, \frac{\partial \mathbf{y}^{(h)}}{\partial \mathbf{a}^{(h)}} \, \mathbf{x}^{(h),T}.$$

The backward propagation of the error is eponymous for the method's name *backpropagation.* Despite the great performance for many deep learning tasks, the biological plausibility of propagating the error signal backward has been questioned ever since

(*Grossberg*, 1987). A simple but effective adjustment was suggested by *Lillicrap et al.* (2016) which is also known as *feedback alignment.* Instead of the transpose of the feedforward weight matrix, a fixed random matrix B is chosen to propagate the error backwards. Compared to the backpropagation variant from eq. (2.2) the update in the hidden layer changes to

$$\delta W^{(h)} = -\eta \left(B\mathbf{e}^{(o)}\right) \frac{\partial \mathbf{y}^{(h)}}{\partial \mathbf{a}^{(h)}} \, \mathbf{x}^{(h),T}.$$

The only constraint to B is that $\mathbf{e}^{(o)T} W^{(o)} B \mathbf{e}^{(o)} > 0$ has to be fulfilled on average, meaning that geometrically, the new feedback signal $B\mathbf{e}^{(o)}$ for the hidden layer lies within 90° of the one used by backpropagation, $W^{(o)T}\mathbf{e}^{(o)}$.

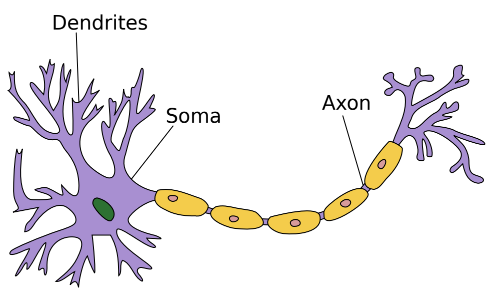
2.2 The Biological Neuron

Biological neural networks have been a great inspiration for deep learning algorithms and ANNs. It is estimated that the human brain contains around 10^{11} neurons of different shape, size and functions (*Azevedo et al.*, 2009). By the use of *synapses*, *presynaptic* neurons connect to *postsynaptic* ones and thereby create complex network structures throughout the brain. A postsynaptic neuron has usually around 10^4 presynaptic partners (*Drachman*, 2005). The connections vary from dense clusters with nearby neurons to linking distant brain regions with each other.

Most neurons can be divided into three functional parts: *dendrites*, *soma* and *axon* (c.f. fig. 2.3). At the dendrites a postsynaptic neuron receives inputs from its presynaptic partners. These inputs are converted into postsynaptic potentials (PSPs) using charged molecules corresponding to a variable graded potential in size and shape. The PSPs are then relayed to the soma where they are integrated over. Depending on the nature of synaptic connection, the stimulation has either an excitatory or inhibitory effect on the neuron, resulting in an increased or decreased membrane potential. Over time the unbalanced ion concentration in and outside the membrane is restored by the membranes permeability and additional ion pumps. In an equilibrated state the membrane potential is referred to as the *resting potential.* (*Gerstner et al.*, 2014)

Given a continuous excitatory stimulation current, the neuron will trigger a fire mechanism once a certain threshold potential is reached. The neuron's membrane becomes hyperpolarized and decreases even below the resting potential as shown in fig. 2.3b entering a *refractory period.* During this period it remains hard but not impossible for the neuron to fire again due to the ongoing hyperpolarization of the membrane. This mechanism releases an electrochemical pulse a so-called action potential or spike,

(a)



(b)

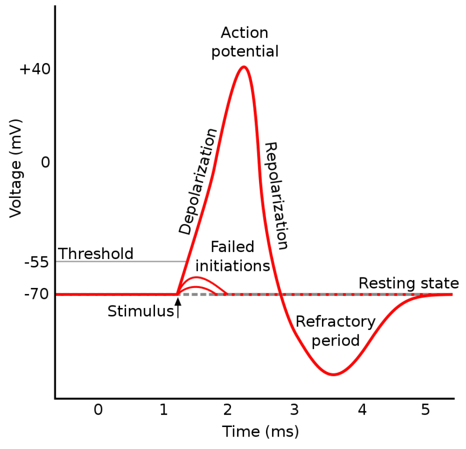


Figure 2.3: Schematics of a biological neuron and an action potential. **(a)** A biological neuron can be split into three main functional parts. The dendrites collect the inputs from presynaptic partners and relay them in the form of postsynaptic potentials (PSPs) to the soma. The soma integrates the PSPs and eventually triggers a fire response. The axon relays this response to the connected neurons at the end of the axon. Figure adapted from *Jarosz*, 2009. **(b)** The fire response of a neuron after exceeding the threshold is called action potential or spike. After a phase of depolarization, the membrane repolarizes and enters the refractory periode during which it is hard but not impossible to fire again. Figure taken from *Iberri*, 2020.

which is then relayed by the axon to the connected partners at the end of the axon. (*Gerstner et al.*, 2014)

The action potential releases various neurotransmitters to overcome a small but physical gap at the synapses, the synaptic cleft. Once a transmitter has docked to a corresponding receptor on the other side, activated ion channels convert the chemical transmission back into an electrical signal resembling the PSP. Depending on the type of neurotransmitters the excitation can be excitatory or inhibitory (*Gerstner et al.*, 2014). According to Dale's principle a presynaptic neuron always releases the same type of neurotransmitter (*Dale*, 1935). To that end, a neuron's output is either excitatory or inhibitory but not both. The input of a neuron, on the other side, is not restricted to a single type of excitation, as various presynaptic partners can be connected.

A wide-spread assumption in the field of neuroscience is that the exact shape of a spike doesn't carry any relevant information and therefore all spikes can be modeled by a stereotypical shape. However, recent research has already suggested that the small variations in the action potential contain vital information (*Debanne et al.*, 2013). For the prior assumption, the communication between neurons is reduced to a temporal and spatial dimension. Temporal information is encoded either in the frequency (*rate coding*) or the precise timing (*time coding*) of spikes whereas the spatial dimension is filled with varying populations of neurons (*Gerstner et al.*, 2014). A more detailed description of neural coding schemes is presented in section 2.4.

The brain has the ability to continuously change the topology of its synaptic wiring, to create new synapses, to alter the chemical properties of the synaptic receptors or to simply strengthen and weaken the synaptic efficacy. With these *plasticity* mechanisms the brain is able to learn and adapt as a reaction to stimulation or even damage. (*Gerstner et al.*, 2014)

One way of learning and forming memory is known as synaptic plasticity where the synaptic strength is changed over time. According to Hebb's theory "neurons that fire together wire together" (*Hebb*, 1949). An experimental proof of such activity-dependent plasticity was found by *Bliss and Lømo* (1973), where they discovered that a short but high frequency stimulation leads to a long lasting change in the synapse's efficacy. This is also referred to as Long-Term Potentiation (LTP). Reducing the stimulus to a low frequency, on the other hand, resulted in the opposite effect: Long-Term Depression (LTD). In combination they can carve out a certain region in the brain which is related to a specific stimulus and thereby create memory (*Nabavi et al.*, 2014). A better understanding of LTP and LTD was gained when Spike-Timing-Dependent Plasticity (STDP) was first observed (*Markram et al.*, 1997, *Bi and Poo*, 1998). STDP shows in principle that presynaptic activity just before a postsynaptic response leads to an increased synaptic strength. If presynaptic activity occurs right after a postsynaptic spike, the synapses are weakened.

Such plasticity mechanism are self-regulating and independent of any reward-giving structure. In analogy to the learning concepts introduced by deep learning, they are also referred to as unsupervised Hebbian learning. However, these methods fall short of successfully training the brain to master complex action-required tasks, as the outcome of a decision made by the brain is not part of the plasticity mechanism. Hence, a behavioral learning strategy requires knowledge over the outcome of taken decisions and needs to remember which decisions have lead to rewards, as the reward in real life is often delayed. (*Gerstner et al.*, 2014)

Developing biologically inspired and plausible supervised learning algorithms is a dedicated goal in the field of modern neuroscience. Before discussing two such candidates in more detail, a practicable model of the biological neuron is presented first. This model is implemented in the analog core of the neuromorphic platform BSS2 and resembles the basis of all experimental work presented in this thesis.

2.3 The Leaky Fire-and-Integrate Model

An early but successful description of the biological neuron dynamics was accomplished by the Leaky Fire-and-Integrate (LIF) neuron model, first described by *Lapicque* (1907). Despite some strong simplifications, the main dynamics of the membrane potential are well described by the model and it thus has been a popular and portable choice for neuromorphic hardware implementations.

In biology, the observation of similar shaped individual action potentials lead to the assumption that the shape of a spike does not transport any information. The Leaky Fire-and-Integrate (LIF) model is based upon this theory and thus every spike can be replaced by a stereotypical shape (*Gerstner et al.*, 2014).

Another observation in biology is that neurons vary much in their shape and size fulfilling different functions. The spatial component plays an important role for the dynamics of a neuron. For instance, the strategic positioning of certain excitatory or inhibitory inputs on the dendrites, either closer or farther away from the soma, give rise to non-linear behavior in the course of the membrane potential. However, extensive spatial dependencies are difficult and costly to implement in a model. Therefore, the LIF neuron neglects the topology of the neuron and is approximated as a point-like integrator. (*Gerstner et al.*, 2014)

In the model, the incoming spike train $S_j(t)$ from various presynaptic partners j is described by a series of spikes s at times $t_j^{(s)}$

$$S_j(t) = \sum_s \delta(t - t_j^{(s)}),$$

with the δ-function denoted as δ.

Each spike of the input spike train evokes a PSP. The impact of the PSPs depends on the individual synaptic weights w_j. For simplicity, the excitatory or inhibitory nature of the synapses is encoded by a sign in the synaptic weight as well. Summing over all input sources yields a total synaptic input current that is seen by the postsynaptic neuron

$$I_{\text{syn}}(t) = \sum_j w_j \left(\epsilon * S_j(t)\right), \tag{2.3}$$

with the convolution of a double exponential kernel ϵ with the input spike train $\epsilon * S_j$

describing the shape of a single PSP. The kernel can be defined as

$$\begin{aligned}\epsilon_{\text{double}}(t) &= \frac{1}{\mathcal{N}} \left(\epsilon_{\text{rise}} * \epsilon_{\text{fall}}\right)(t) \\ &= \frac{1}{\mathcal{N}} \exp\left(-\frac{t}{\tau_{\text{rise}}}\right) * \exp\left(-\frac{t}{\tau_{\text{fall}}}\right),\end{aligned}$$

with a rising and falling time constant τ_{rise} and τ_{fall} respectively. A constant $\mathcal{N}$ norms the kernel to unity. As the rising constant goes to zero $\tau_{\text{rise}} \to 0$ the double exponential turns into a single exponential kernel $\epsilon_{\text{single}} = \epsilon_{\text{fall}}$.

The membrane potential V_{m} changes with the continuous synaptic input causing an unbalanced ion concentration inside the membrane. Passive as well as active processes are permanently restoring the membrane potential back to its equilibrium state which is associated with the resting potential V_{leak}. In the LIF model, the temporal scale of these restoring processes is defined by the membranes capacitance C_{m} and the leakage conductance g_{leak} yielding the membrane's time constant $\tau_{\text{mem}} = \frac{C_{\text{m}}}{g_{\text{leak}}}$. The dynamics of the membrane are then given by a single differential equation

$$C_{\text{m}} \frac{dV_{\text{m}}}{dt} = -g_{\text{leak}}(V_{\text{m}} - V_{\text{leak}}) + I_{\text{syn}}. \tag{2.4}$$

As for a biological neuron, a LIF neuron triggers a spike once a certain threshold ϑ is crossed following the condition

$$V_{\text{m}}\left(t^{(s)}\right) = \vartheta \Leftrightarrow \text{neuron fires at time } t^{(s)}.$$

Then the membrane is set to a reset potential V_{reset} where it remains unchanged for a refractory period of τ_{refrac}

$$V_{\text{m}}(t) = V_{\text{reset}} \quad \forall t \in \left(t^{(s)}, t^{(s)} + \tau_{\text{refrac}}\right].$$

Unlike its biological counterpart, the modeled neuron cannot spike during the refractory period.

A LIF neuron doesn't keep track of any previous spikes once a spike is released, given that the time constant of the synaptic input is shorter than the one of the membrane potential, in particular if $\tau_{\text{mem}} > \tau_{\text{fall}}$. These limitations make it impossible for the model to correctly describe neuronal behavior such as spike bursts (*Gerstner et al.*, 2014) and led to a demand of a more detailed modeling as the Adaptive-Exponential Integrate-and-Fire (AdEx) model. The AdEx model resembles an extension to the LIF model featuring an additional adaption state variable that provides post-spike memory

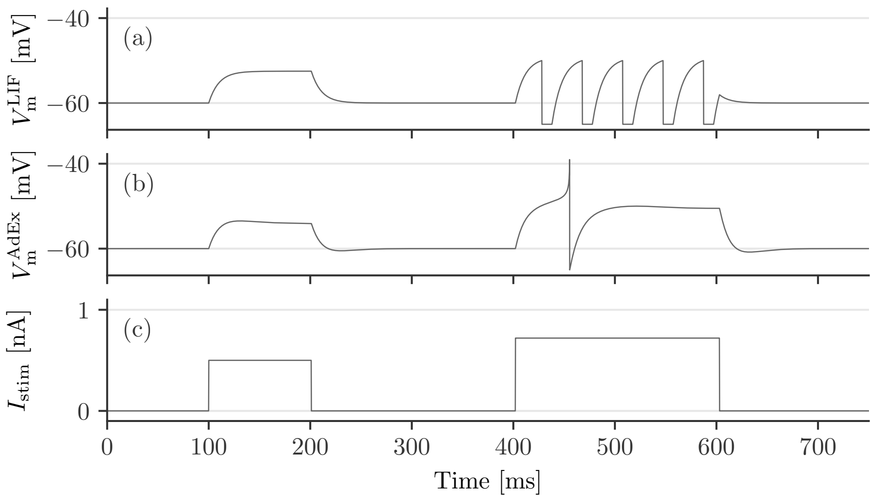


Figure 2.4: Membrane dynamics of the LIF and Adaptive-Exponential Integrate-and-Fire (AdEx) given a constant input. **(a)** Evolution of the membrane potential $V_\mathrm{m}^\mathrm{LIF}$ according to the LIF model in response to a different stimulation currents. The first current is not strong enough to trigger an action potential. A more intense stimulation yields a repetitive and equidistant spiking pattern. **(b)** Given a small box shaped current, the evolution of the AdEx neuron's potential $V_\mathrm{m}^\mathrm{AdEx}$ is similar to the LIF model. At higher inputs, a negative adaption suppresses a repetitive spiking pattern after the first spike. The peak resembles the positive exponential voltage feedback simulating an action potential. **(c)** A box-shaped stimulation current I_stim is used for both models to show the course of the membrane potential. Figure taken from *Stradmann*, 2019

to the membrane. Depending on the sign of the adaption, the neuron is either inhibited or engaged to fire again after having spiked at least once (see fig. 2.4).

For the scope of this thesis a more advanced model is not yet required. All experiments are done using the simpler LIF model. However, the use of the AdEx model for a spike-based learning rule such as SuperSpike is promising when targeting real-world applications. A respective project has already been planned in the future and will be explained in more detail in chapter 5.

A supervised learning algorithm based on a single LIF neuron will not be able to solve any difficult task yet. In the human brain millions of such neurons are split up into various areas which are each responsible to perform a certain type of work such as smell, speak or motor control. In the next section different ways of how neurons can communicate with each other in order to perform a certain task are discussed.

2.4 Neural Coding with Spiking Neural Networks

The communication among biological neurons can be modeled by spiking neural networks (SNNs) where individual neurons convey information by sending spikes to their connected partners. As mentioned before, spikes can be approximated to have a stereotypical shape, leaving the temporal to encode information. In the context of large multilayer networks, the spatial dimension of the network encodes information as well, i.e. the types of synapses, the synaptic strength and the network's topology.

In the following, the different coding schemes and supervised training approaches for a feedforward multi-layer SNNs using LIF neurons are presented.

2.4.1 Rate Coding

In an attempt to explain computational processes in the brain, the activation of an artificial neuron has been mapped onto the firing rate of a spiking neuron by *Rieke et al.* (1999). The most apparent way to define a neuron's spike rate ν is to count the number of spikes n_{spikes} fired within a period T

$$\nu = \frac{n_{\text{spikes}}}{T}.$$

From a practical point of view, this method is time consuming and can therefore not be the basis upon which fast decisions are taken by the brain. The measurement time can be reduced by shortening the period T to ΔT, but this will yield a more inaccurate firing rate. By repeating the measurement multiple times, the average rate improves the accuracy, but the total measurement duration is prolonged again. Moreover it is not feasible that the exact same input occurs multiple times in a real world problem.

To solve both issues, a population average rate ν_{pop} can be used. The averaged firing rate of a population with n neurons $\langle \nu \rangle_{\text{pop}}$ yields an accurate rate despite the reduced measurement time ΔT

$$\langle \nu \rangle_{\text{pop}} = \frac{\sum_i n_{\text{spikes},i}}{n \Delta T},$$

with the number of spikes for a neuron i within the population $n_{\text{spikes},i}$.

In the terminology of the LIF model, a presynaptic spike train S_j from a source j can be associated with a mean firing rate ν_j. This is based on the assumption that the spikes of the input follow a Poisson process and therefore the firing rate of a neuron is well described by a Poisson distribution in most cases (*Averbeck*, 2009), which in turn legitimates the use of a mean firing rate and a certain accuracy. In this way the time average of the synaptic input current for a LIF neuron can be expressed in terms of the incoming firing rates $\nu_{\text{in},j}$ and their respective synaptic weight w_j

$$\langle I_{\text{syn}} \rangle = \sum_j w_j \nu_{\text{in},j}.$$

Training with Rate Coding With a rate coding approach, a spiking feedforward network obtains properties from an ANNs. In particular, the activation function of a node in an SNN associates a given input rate ν_{in} with an output rate ν_{out}

$$\Phi(\nu_{\text{in}}) = \nu_{\text{out}}.$$

Unlike for individual spikes, the gradient of the activation function can now be formulated with the use of rates. This allows the use of typical deep learning methods such as stochastic gradient descent (SGD) for rate-based SNNs.

The choice for a sigmoidal activation function in combination with a cross entropy loss has already been motivated in section 2.1.1. In the following, the necessary adjustments to shape the response of the LIF neuron model into a sigmoidal activation function are presented.

The stimulation of a LIF neuron can be expressed by the synaptic input current I_{syn} or the corresponding input firing rate ν_{in}. In analogy to the activation of an ANN the synaptic current is split up into an input and bias term $I_{\text{syn}} = I_{\text{in}} + I_{\text{bias}}$. The input spike rate ν_{in} of the neuron scales linearly with the synaptic current and therefore $\nu_{\text{in}} \propto I_{\text{in}}$. The activation function of the LIF neuron is then approximated by

$$\frac{1}{\Phi(\nu_{\text{in}})} = \frac{1}{\nu_{\text{out}}} \approx \tau_{\text{refrac}} + \tau_{\text{mem}} \frac{\vartheta - V_{\text{reset}}}{I_{\text{syn}}}, \tag{2.5}$$

given that the input rate is high and the time constants of the membrane τ_{mem} and the synaptic input current τ_{fall} are smaller than the refractory period τ_{refrac} (c.f. *Brunel*, 2000). The output rate saturates at a maximum rate $\nu_{\text{max}} \approx {}^1/_{\tau_{\text{refrac}}}$ for a high enough input rate.

In the limit $\vartheta - I_{\text{syn}} \gg \sigma$, i.e. for low input rates, the activation function yields

$$\nu_{\text{out}} \approx \frac{(\vartheta - I_{\text{syn}})}{\tau_{\text{mem}} \sigma \sqrt{\pi}} \exp\left(-\frac{(\vartheta - I_{\text{syn}})^2}{\sigma^2}\right), \tag{2.6}$$

with the fluctuations of a single excitatory input source σ.

Without external noise, the fluctuations are small, reflecting only the variations of the input spike train and thus the activation function shows a steep incline. One way to smoothen the course of the function is to increase σ, e.g. by injecting additional

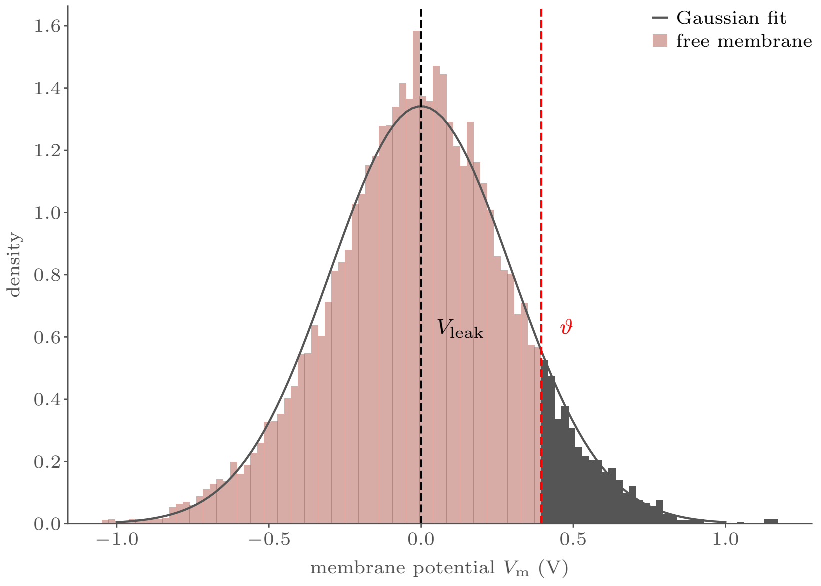


Figure 2.5: Simulation of the Gaussian free membrane potential distribution, i.e. the distribution of the membrane potential in absence of any spiking mechanisms. The distribution f_{V_m} centers around V_leak. The width of the distribution correlates to the amount and synaptic weight of injected noise spikes. The part of the distribution colored in black, exceeds the threshold potential ϑ indicated by a red vertical line and would lead to spikes.

Poisson spikes. A continuous noise stimulation with these noise spikes leads to a Gaussian free membrane potential distribution f_{V_m} which is centered around the resting potential V_leak as depicted in fig. 2.5. The term *free* refers to the absence of any spiking mechanisms of the membrane. In a naive approach, the part of the free distribution that exceeds a certain threshold potential correlates to the number of fired spikes. This neglects non-vanishing effects from the fire dynamics of the membrane which have been investigated in more detail by *Petrovici* (2012). The impact of these dynamics can be reduced by the use of short time constants for the synaptic input and the membrane.

However, despite the strongly simplified picture, this view still offers a correct intuition of how the threshold, the leak potential and the strength of the noise effect the free membrane potential and in turn change the shape of the activation function: More noise leads to a broader distribution and thus a more gentle incline of the output rate. The synaptic input moves the distribution to either a lower or a higher mean value. And changing the threshold corresponds to adding a bias term to the synaptic input. The latter has been motivated by *Petrovici et al.* (2016). The bias term in the neuron's activation can be adequately replaced by adapting the relative distance δV between the resting potential and the threshold

$$b \propto \delta V = V_{\mathrm{leak}} - \vartheta. \tag{2.7}$$

With the approximations made in eq. (2.5) and eq. (2.6) as well as a suitable choice of neuron parameters, the activation function yields an sigmoidal shape (fig. 2.6). The slope of the sigmoid can be easily adapted by changing the synaptic strength of the input spike trains. And the alignment along the x-axis can be done by the bias. Finding the appropriate neuron model parameters to create a well shaped sigmoid can be tricky. The parameters in-use are listed in table A.1 in the appendix.

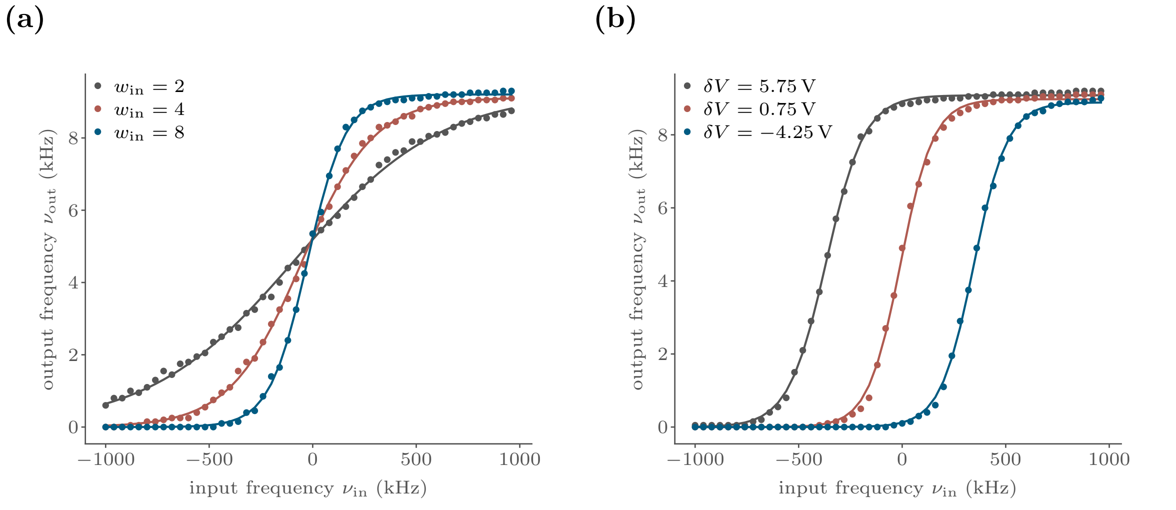


Figure 2.6: Simulation of a sigmoidal activation function using Poisson noise. The shape of the activation function Φ depends on several parameters of the LIF neuron and the Poisson input spike trains. **(a)** The input current that is seen by the neuron is directly proportional to the weight $I_{\mathrm{syn}} \propto w_{\mathrm{in}}\nu_{\mathrm{in}}$. In this way, a lower weight causes a lower input current which in turn leads to a smaller incline of the output rate (*black*) than for a higher input weight (*red* or *blue*). **(b)** The *red* curve is centered at zero input which requires a small threshold offset of 0.75 V. A positive potential difference δV shifts the sigmoid (*black*) to the left. With a lower threshold the neuron starts to fire earlier. The potential difference of the *blue* and *black* curve relative to the centered sigmoid equals $\pm 5\,\mathrm{V}$.

2.4.2 Temporal Coding

SNNs can also model more complex processes which depend on the interspike interval (ISI) or a specific spike time. In comparison to rate coding, using the exact spike time to encode information is more exposed to noise. In turn, the response time of such coding is highly increased, since there is no need to establish a certain firing rate first. The use of temporal coding has been observed in biology multiple times and it has been proven to be of high importance (*Gerstner et al.*, 1996 and *Rieke et al.*, 1999). Another advantage over rate coding is the efficiency that comes with the use of single spikes as each spike costs energy. Sparse temporal coding condenses the necessary resources to solve a task to a minimum.

So far, training SNNs with hidden units has been proven to be a difficult task (*Pfeiffer and Pfeil*, 2018). With the binary nature of spikes, the neuron's activity is non-differentiable on an individual spike level. Hence, well proven methods from conventional deep learning cannot be simply transfered to temporal coding with SNN. A promising workaround was suggested with *SuperSpike*, where a supervised learning algorithm is implemented for SNNs by using a surrogate gradient (*Zenke and Ganguli*, 2018).

Supervised Training with Temporal Coding As before, only feedforward networks on the basis of LIF neurons are considered. The training routine of SuperSpike can again be split up into a forward and backward pass.

The **forward pass** changes only slightly compared to ANNs. Instead of a continuous input and output, the formalism of the LIF is used. The presynaptic activity of neuron j is given by the spike train S_j and the postsynaptic activity of neuron i by the spike train S_i. Again the activation function Φ is determined by the dynamics of the LIF neuron.

As stated in section 2.1.1, most training approaches involve the optimization of a certain loss function $\mathcal{L}(\theta)$ that depends on the network's parameters θ. In the **backward pass** of the SuperSpike formalism the loss is given by the *van Rossum distance* (*van Rossum*, 2001) of a target spike train S_i^* and the actual output spike train S_i

$$\mathcal{L} = \frac{1}{2} \int_{-\infty}^{t} dt' \left[\left(\alpha * S_i^* - \alpha * S_i \right) (t') \right]^2 , \qquad (2.8)$$

with a smooth double exponential kernel α.

The computation of the gradient for 2.8 with respect to θ requires the derivative of a spike train S_i for a neuron i. In particular, $\frac{\partial S_i}{\partial w_{ij}}$ which is undefined for the time of a spike and zero elsewhere. SuperSpike circumvents this issue by rendering the spike train with a smooth auxiliary function $\sigma(V_{\mathrm{m},i})$ of the membrane potential $V_{\mathrm{m},i}$ for a neuron i and thus the ill-defined gradient of the spike train can be replaced by a surrogate derivative $\sigma'(V_{\mathrm{m},i})$

$$\frac{\partial S_i}{\partial w_{ij}} \quad \rightarrow \quad \sigma'(V_{\mathrm{m},i})\frac{\partial V_{\mathrm{m},i}}{\partial w_{ij}}.$$

In SuperSpike $\sigma(V_{\mathrm{m},i})$ is chosen to be a fast sigmoid

$$\sigma(V_{\mathrm{m},i}) = \frac{\beta(V_{\mathrm{m},i} - \vartheta)}{1 + \beta|V_{\mathrm{m},i} - \vartheta|}, \tag{2.9}$$

with a slope parameter β. The surrogate partial derivative yields $\sigma'(V_{\mathrm{m},i}) = \frac{\beta}{(1+\beta|V_{\mathrm{m},i}-\vartheta|)^2}$. Other auxiliary functions will work too. Common choices are for example piecewise linear or exponential functions.

At a first glance, it appears that the problem has just been shifted to computing the partial gradient of the membrane potential instead. When the potential $V_{\mathrm{m},i}$ is formulated as a spike response model for LIF neurons it again depends on the output spike train S_i (*Gerstner et al.*, 2014). However, under the assumption of a low output rate the gradient can be approximated by $\frac{\partial V_{\mathrm{m},i}}{\partial w_{ij}} \approx (\epsilon * S_j)$ with ϵ another double-exponential kernel corresponding to the shape of a PSP. Plugging in the approximation and the formulation of the gradient as a surrogate gradient yields

$$\frac{\partial w_{ij}}{\partial t} = \eta \int_{-\infty}^{t} dt' \underbrace{\left(\alpha * (S_i^* - S_i)\right)}_{=e_i \text{ (Error)}} \; \alpha * \Big(\underbrace{\sigma'(V_{\mathrm{m},i})}_{\text{Post}} \underbrace{(\epsilon * S_j)}_{\text{Pre}} \Big), \tag{2.10}$$

with the learning rate η.

The formulation for a hidden layer is similar, except for the calculation of the hidden error. For simplicity the network is reduced to only one hidden layer structure. The error signal of the i-th unit e_i^{hidd} in the hidden layer is given by backpropagation of the error e_k from the output layer

$$e_i^{\mathrm{hidd}} = \sum_k w_{ik} e_k,$$

with the feedforward weights w_{ik} between the hidden and output layer. As for the gradient descent, feedback alignment can also be used within the SuperSpike formalism by using a random weight matrix. Despite the formal restriction to a single hidden layer, the method can be easily adapted for multiple hidden layers.

2.5 Neuromorphic Hardware

The great success of deep learning in the recent years has also increased the demand for new specialized hardware that handles the huge amount of parallel computing more efficient than the conventional von Neumann architecture. Among various approaches, neuromorphic hardware is arguably the closest form of computing to the human brain. As its inspiration, it is designed to be robust to noise or malfunctioning sectors, it is highly energy efficient and remains flexible throughout the learning process. These properties are of high interest for any deep learning applications. As a consequence, several platforms[1] have been launched by big industry corporations and academia over the last years.

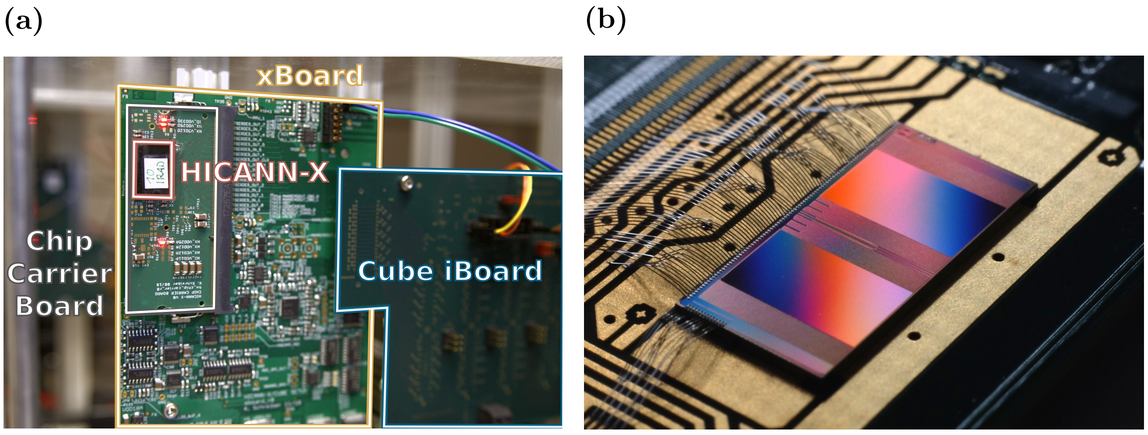


Figure 2.7: The cube setup HICANN-Xv1. **(a)** The chip (*red*) is mounted on the chip carrier board and is protected by a cover. The chip carrier board (*gray*) is then mounted on the xBoard (*yellow*) which is connected to the FPGA over the cube iBoard (*blue*). **(b)** Close-up of the newest BSS2 single chip. The analog core of the neuromorphic chip is bonded before a protective cover is placed over it. Picture taken by Müller, 2020.

In the context of this thesis, the presented learning strategies from section 2.4 are implemented on an analog neuromorphic platform called BrainScaleS2 (BSS2). This mixed-signal accelerated emulation for SNNs is based in Heidelberg and is the result of a long term cooperation with the EU, namely the Human Brain Project (HBP). The BSS2 platform is designed to perform various plasticity algorithms on-chip. The core of the platform is based upon a complete redesign of its predecessor the High Input Count Analog Neural Network (HICANN). By reducing the CMOS manufacturing process from 180 nm to 65 nm several new features could be included on the new core. One of the main renewals is a general purpose unit which can be used for any on-chip

[1] Among others, SpiNNakker and BrainScaleS by EU's Human Brain Project, Loihi by Intel and Truenorth by IBM

	Neuron Model	Neurons	Synapses
HICANN-DLSv2	LIF	32	32×32
HICANN-Xv1	AdEx	512	512×256

Table 2.1: Overview of relevant BSS2 prototypes. The HICANN-DLSv2 is produced at a reduced size to avoid unnecessary costs. The HICANN-Xv1 is the first functioning full-size chip of the BSS2 platform.

computation and a specialized vector unit that efficiently provides parallel access to observables from the analog core. The features have been implemented step by step on various prototype versions. The following paragraphs will focus on the specifications of the prototypes used for the experimental implementations of the discussed deep learning methods (c.f. table 2.1).

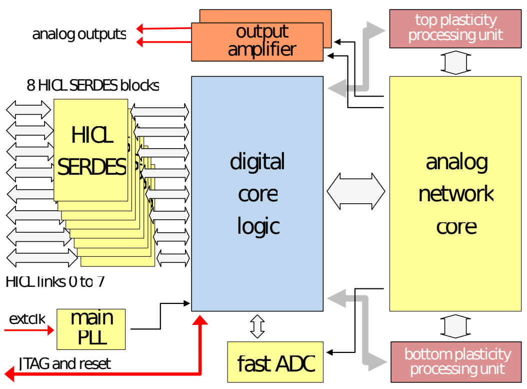
On the HICANN-DLSv2 the redesign of the LIF neuron model is implemented. Moreover, the chip features a general purpose unit and a vector unit which are summarized as the plasticity processing unit (PPU). The HICANN-DLSv2 has been manufactured at a reduced size of 32 neurons and 32×32 synapses to avoid unnecessary costs. The HICANN with Digital Learning System and Hagen eXtensions (HICANN-Xv1) is the first full-size prototype of the BSS2 platform with 512 AdEx neuron circuits and 256 possible synaptic connections per neuron. In addition to the PPU, the chip features on-chip event routing and HAGEN extension, an early realization of a neuromorphic system which basically implements an on-chip analog matrix multiplication (*Schemmel et al.*, 2020). Another renewal are dedicated noise generators, which come in handy when working with sigmoidal activation functions and rate coding (c.f. section 2.4.1).

2.5.1 Architecture of BSS2

The design of both prototypes can be divided into an *analogue* and a *digital* core (c.f. figure 2.8a). The external communication is established by a field programmable gate array (FPGA) accessing eight serial Low Voltage Differential Signaling (LVDS) links. The interface not only manages read and write instructions but handles any spike-event data in both directions.

Depending on the chip version, the analogue core contains the physical implementation of either a LIF or an AdEx neuron model (*Aamir et al.*, 2018b and *Aamir et al.*, 2018a). The analog neuron model parameters can be tuned by setting respective bias currents with a 10 bit Digital to Analog Converter (DAC). Each neuron can be controlled and adjusted individually (*Hock et al.*, 2013).

(a)



(b)

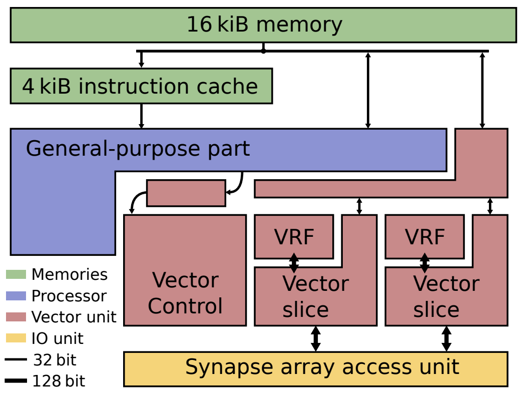


Figure 2.8: Overview of the BSS2 architecture. **(a)** The neuromorphic platform is divided into a digital and analog core which is connected to an external host via an FPGA. Two plasticity processing units provide computational power for on-chip training. Figure taken from *Schemmel*, 2017 **(b)** The plasticity processing units are divided into a general purpose part which is based on a 32 bit architecture and a vector unit. The latter enables efficient parallel data processing of analog parameters and observables. Figure taken from *Friedmann et al.* (2017)

The biological time constants of neurons and synapses are usually in the order of 1 to 100 milliseconds. The *in-silico* implementation of the neuron models causes a temporal speed-up compared to their *in-vivo* counterpart, leading to chip-time constants of a few microseconds. This acceleration is possible due to the supra-threshold dynamics of CMOS transistors.

On the full-size chip, the neurons are connected by a grid of 512×256 synapses (32×32 on the smaller one). The activity of presynaptic neurons is injected row-wise by dedicated synapse drivers as either excitatory or inhibitory spikes. Each synapse has access to a 6 bit decoder address and compares it to a 6 bit label of the incoming spikes (see fig. 2.9). If they match, the spike is relayed to the corresponding neuron at the bottom of the synapse grid. The efficacy is thereby configured by a 6 bit weight.

The analog core features several observables which are relevant for the implementation of plasticity rules. The firing rate of a neuron for instance is recorded by a 10 bit spike counter on the HICANN-DLSv2. The counter has been replaced by a smaller 8 bit version on the full-size chip to save some space. In addition, every synapse features two correlation sensors (causal and anti-causal) which are designed to record STDP traces. Other relevant observables are the traces of the synaptic input current or the membrane potential. The latter is key for the implementation of temporal-based plasticity rules such as SuperSpike.

When training a highly accelerated analog system such as BSS2 platform, a fast computation of any plasticity rule is indispensable. A Columnar Digital to Analog Converter

(a)

(b)

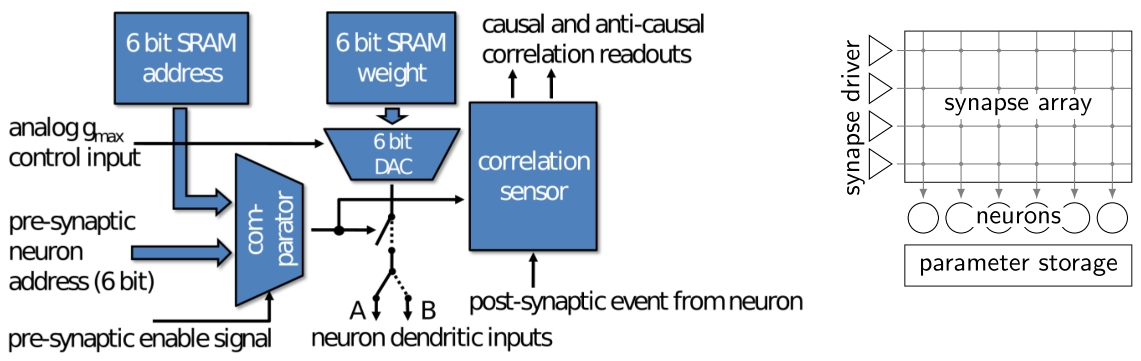

6 bit SRAM address
6 bit SRAM weight
causal and anti-causal correlation readouts
analog g_{max} control input
6 bit DAC
correlation sensor
pre-synaptic neuron address (6 bit)
com-parator
A
B
post-synaptic event from neuron
pre-synaptic enable signal
neuron dendritic inputs
synapse driver
synapse array
neurons
parameter storage

Figure 2.9: Synapse circuit overview on HICANN-DLSv2. **(a)** Synapse drivers inject the presynaptic activity row-wise as either inhibitory or excitatory spikes. The 6 bit addresses of the presynaptic neuron is compared at each synapse with a local 6 bit address. If the addresses match, the spike is relayed to the corresponding neuron at the bottom of the synapse grid. The synaptic strength can be configured by a 6 bit weight. The two correlation sensors (causal and anti-causal) record STDP traces. Figure taken from *Friedmann et al.* (2017). **(b)** Overview of the row-wise synapse drivers and the synapse array. Figure adapted from *Billaudelle et al.* (2019).

(CADC) provides row-wise parallel access to the STDP traces with a total of 2×32 respectively 2×512 CADC channels (one channel per correlation sensor per synapse row). The on-chip vector unit then guarantees an efficient access to the CADC read-out by the use of Single Instruction Multiple Data (SIMD) operations. On the newer HICANN-Xv1, the CADC routing possibilities have been extended such that the membrane potential can be accessed as well. Before this renewal the membrane potential was recorded by a fast Multiplying ADC (MADC) which accesses only one neuron at a time. With the lack of parallelization a fast in-experiment use cannot be realized and is therefore feasible for training purposes. This limitation is compensated by a better accuracy and resolution making the MADC a useful debugging and observation tool.

Apart from dedicated spike counters, the digital neuron back end registers any spiking event and transfers them to the digital core logic, where the events are merged with any activity coming from the noise spike generators or PPUs as well as from an external source. The events are then rerouted back into the synapse grid accordingly, enabling recurrent connections and multilayer network structures.

These observation features for the analog neuronal dynamics are then combined by the general purpose unit with which complex plasticity rules can be implemented on-chip (see fig. 2.8b). The HICANN-Xv1 features even a second PPUs to provide enough computational power for the increased chip size. The software capabilities of the general purpose unit for the new chip are under continuous development. For instance a fast access to an external memory will be released in near future. The current hardware and software constraints of both prototypes will be discussed in a final chapter, after the experimental implementations have been presented (chapter 3 and chapter 4).

3 Classification of the Circles Data Set on BSS2

In the first of two deep learning experiments which are implemented on the BSS2 platform, a spiking feedforward network is emulated as an ANN using rate coding. As derived in section 2.4.1 the SNN can be then trained with conventional backpropagation. In addition the experiment focuses on a fully on-chip implementation. Only for monitoring purposes of the training progress the PPU's memory is continuously read out through the FPGA interface.

Input A	Input B	Output A $\veebar$ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.1: The XOR classification follows the prediction given in the table: if the binary inputs are not equal the XOR operator returns true and false otherwise.

In deep learning solving the XOR or "exclusive or" problem (c.f. table 3.1) has become a very first benchmark to test the functionality of novel network designs and algorithms, since it requires a multi-layer network structure with a non-linear activation function to be solved (*Goodfellow et al.*, 2016). As part of a preliminary work, an on-chip implementation of stochastic gradient descent (SGD) on the prototype HICANN-DLSv2 has been successfully tested with the XOR problem.

In a next step, the difficulty of the task has been increased with the classification of the *Circles* data set. The data set consists of two classes representing an inner and outer circle as shown in fig. 3.1. A deep network then tries to place a decision boundary in between the two circles. Before discussing the details of the training process, the rate coding of the task and the implementation of the network's training method SGD are presented in the following sections.

3.1 Circles Task

The Circles data set describes a set of points $p = p(x, y)$ in a two-dimensional plane, which are in either of two disjunct rings, each representing a class

$$\text{class}(p) = \begin{cases} 0, & r_{\text{inner}}^2 < x^2 + y^2 < r_{\text{outer}}^2, \\ 1, & R_{\text{inner}}^2 < x^2 + y^2 < R_{\text{outer}}^2, \end{cases} \tag{3.1}$$

with the confining inner and outer radii of the first and second ring r_{inner}, r_{outer} and R_{inner}, R_{outer} respectively. The goal of the task is to find a decision boundary that successfully separates the both rings as shown in fig. 3.1.

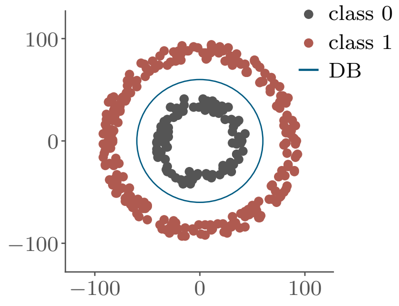


Figure 3.1: The Circles data set has two classes (*red* and *black*) which can be separated by a decision boundary (*blue*).

The Circles task is slightly modified to reduce the implementation effort on the PPU and to make it easier to find a decision boundary. The class 0 is reduced to one confining circle by setting the inner radius to zero. The outer radius of the second class is replaced by the 8 bit limitations of the input. The remaining two boundaries are set far apart to $r_{\text{outer}} = \sqrt{8000}$ and $R_{\text{inner}} = \sqrt{13000}$. The area in between the classes is where the network should place the decision boundary. The more space is available, the easier it is for the network to succeed.

With a rate-based coding, the data points need to be translated into firing rates first. A

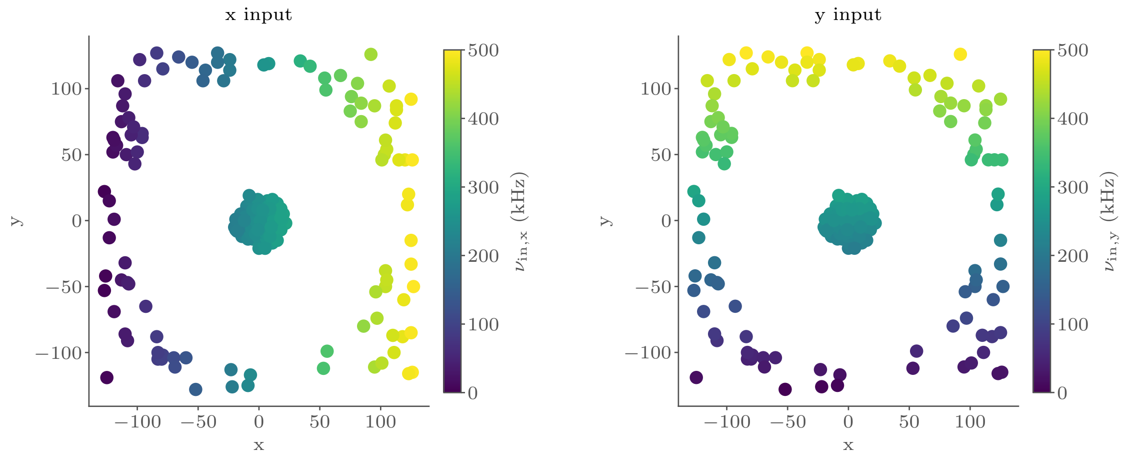


Figure 3.2: Input coding for the Circles experiment. Each point $p(x, y)$ of a representative data set with 100 points per class is mapped to two input frequencies each depending on one of the coordinates. In the left figure, the x-dependency is shown and to the right the y-dependency.

point p is represented by two signed 8 bit integers which correspond to the coordinates x and y with $x, y \in [-128, 127]$. The corresponding firing rate of a coordinate $c \in \{x, y\}$ is then given by

$$\nu_{\text{input, c}}(c) = \nu_{\max} \cdot \frac{c + 128}{255}, \tag{3.2}$$

with the maximum firing rate $\nu_{\max}$ set by a respective spike train source. A representative data set of 100 samples per class illustrates in fig. 3.2 how the points of the modified task are distributed. Moreover, the translation of a point into an input rate is shown. For instance, a point from the lower right corner p(100,100) is associated with two input rates: one depends on the x coordinate $\nu_{\text{in, x}}(100) \approx 450\,\text{kHz}$ and one on the y coordinate $\nu_{\text{in, y}}(-100) \approx 55\,\text{kHz}$.

As for a supervised learning method, the input data is labeled. To identify the class of an input pattern in the output layer, a single readout neuron is trained to yield a target rate ν^*

$$\nu^*(p) = \begin{cases} \nu_0^* = 32.6\,\text{kHz}, & \text{class}(p) = 0, \\ \nu_1^* = 93.5\,\text{kHz}, & \text{class}(p) = 1. \end{cases} \tag{3.3}$$

The mismatch between the target rate and firing rate of the readout neuron ν_{out} then determines the error of the output layer $e^{(o)}$

$$e^{(o)}(p) = \nu_{\text{out}}(p) - \nu^*(p).$$

The decision boundary, the rate separating both classes, is chosen to be the mean of both target rates

$$\nu_{\text{DB}} = \frac{\nu_0^* + \nu_1^*}{2} = 63.05\,\text{kHz}.$$

3.2 Poisson Spike Train Generator

The on-chip implementation of the experiment requires the generation of spikes for the input data and noise spikes for sigmoidal activation function. The latter relies on spike trains with an underlying Poisson distribution of the spikes. With the lack of dedicated spike generators on-chip, the PPU has to fulfill the task.

One way to numerically generate a Poisson spike train is to repeatedly perform a Bernoulli process on a short time interval Δt over a period T. The Bernoulli process is equivalent to an unfair coin flip with probability p, which is set depending on the desired firing rate ν by $p(\nu) = \nu \cdot \Delta t$. Given that the probability equals one, i.e. at every time interval a spike will be fired, the firing rate reaches its maximum at

$\nu_{\text{max}} = {}^{1}\!/_{\Delta t}$. However, the spike train is no longer Poisson-based for high firing rates but rather fires with a fixed interspike interval (ISI). Therefore, the spike generator will only be used at low frequencies for the noise generation ($\nu_{\text{noise}} \ll \nu_{\text{max}}$). The input spike trains on the other hand do not rely on pure Poisson spike trains and thus can use the full frequency range of the spike train generator.

With respect to hardware limitations, the maximum fire rate of the spike train generator $\nu_{\text{max, ppu}}$ is defined by the shortest possible ISI on the PPU which is in turn limited by the time required to generate a single spike Δt_{spike}

$$\nu_{\text{max, ppu}} = \frac{1}{\Delta t_{\text{spike}}} = \frac{1}{0.44\,\mu\text{s}} = 2.27\,\text{MHz}.$$

The duration of the spike generation Δt_{spike} on the PPU is determined by recording a constant spiking pattern with the highest possible frequency. The total number of spikes is counted by on-chip spike counters and the duration is simply measured by recording the membrane potential with an oscilloscope.

On the PPU the Bernoulli process is implemented by comparing the frequency dependent probability $p(\nu)$ with a randomly drawn number. A popular method for generating random numbers on a system with limited memory and computational resources is the *xorshift* (*Marsaglia et al.*, 2003). A random number is thereby generated by repeatedly applying the XOR operator on a seed variable and a bit-shifted version of itself.

3.3 Activation Function on Chip

The theoretical background of how the activation function of a LIF neuron becomes sigmoidal has been motivated in section 2.4.1. The main tool to create such an activation function is the continuous stimulation of the neuron with Poisson noise spike trains. Thereby a broad Gaussian distribution of the membrane potential is established, which in turn yields a sigmoidal activation function. In fig. 3.3, the recorded free membrane potential on the HICANN-DLSv2 is shown while being stimulated by excitatory and inhibitory noise spike trains with a frequency of 70 kHz and a synaptic strength of $w_{\text{noise}} = 15$.

The overall activation function depends on several parameters such as the neuron potentials, the synaptic weights and frequencies of the spike sources as well as the time constants τ_{mem}, τ_{fall} and τ_{refrac}. Most of these parameters are not required to be changed during the experiment. However, their choice has a great impact on the shape of the activation function. A suitable setting for these parameters was found

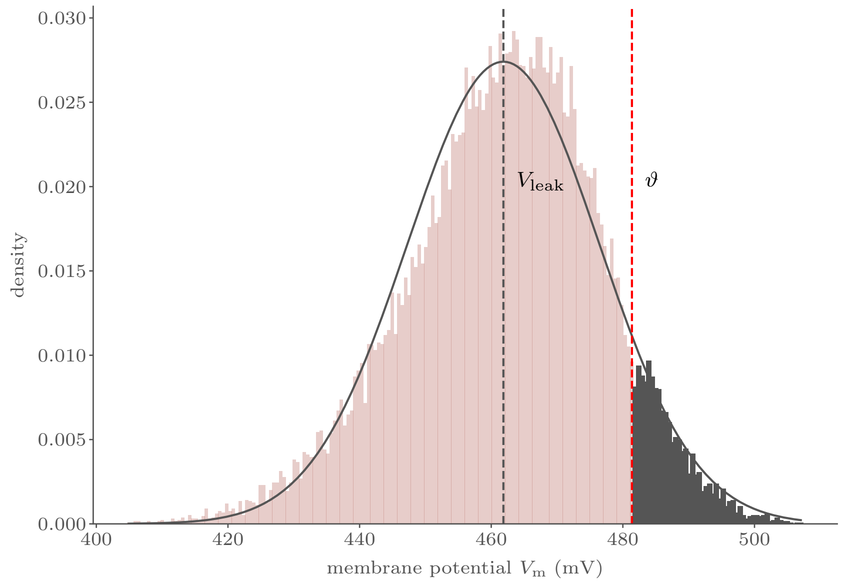


Figure 3.3: Gaussian free membrane potential distribution on HICANN-DLSv2. A broad distribution of the free membrane is achieved by sufficiently stimulating the membrane with Poisson noise spikes. By changing the frequency and impact of the noise spikes the width of the distribution can be changed. In a naive approach, the part of the distribution that exceeds a certain threshold potential correlates to the number of fired spikes (*black*).

by performing various parameter sweeps on the HICANN-DLSv2. In the appendix in table A.1 the final choice of parameters for the activation function are compared to the ones chosen for the simulation in section 2.4.1.

Calibration The manufacturing process of analog neuromorphic hardware causes systematic stochastic deviations in the neuron and synapse parameters. The thereby induced heterogeneity between neurons and synapses is referred to as *fixed-pattern noise* and is constant in time. In an uncalibrated state, the neuron circuits exhibit activation functions which do not align with regards to their maximum rate and alignment along the x-axis (see fig. 3.4a). Despite the detuned parameters, it has been shown that plasticity rules can correct the intrinsic imperfections of the analog hardware up to a certain degree (*Wunderlich et al.*, 2019). However, the dynamic range of the individual neurons on the chip is limited and to ensure properly overlapping activation functions a calibration of the respective parameters is inevitable.

A collection of simple experiment-specific PPU-based calibration routines is implemented to maintain a pure on-chip implementation. The calibration is based on a binary search algorithm to find the suitable DAC-values for the analog neuron parameters. In general, the binary search algorithm compares a target value to the middle

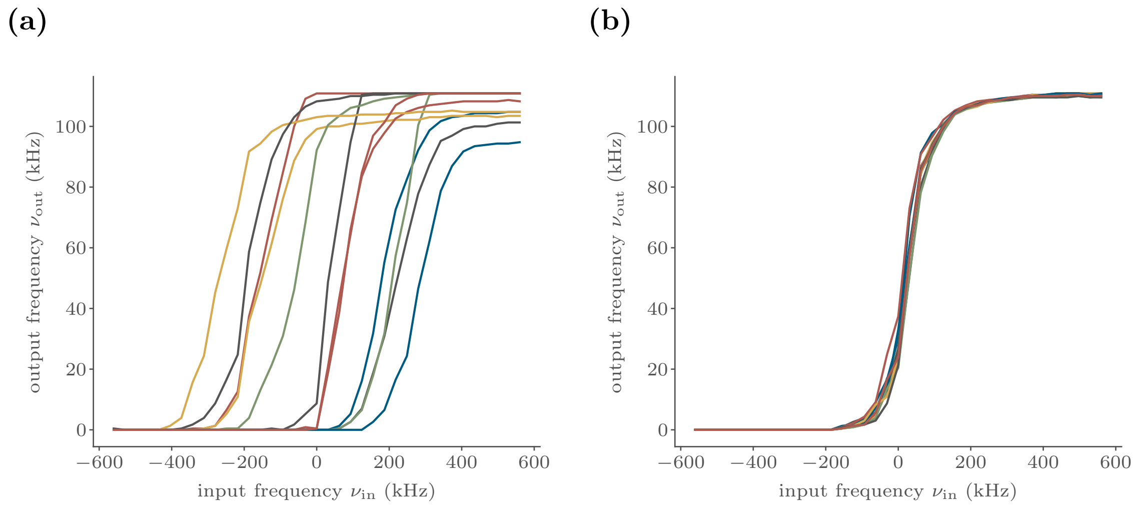


Figure 3.4: Calibration of the sigmoidal activation function on HICANN-DLSv2. **(a)**: In the uncalibrated state the maxima of the activation function and the position of the inflection point deviates due to fixed pattern noise of the hardware. **(b)**: The calibration of the refractory time and the resting potential aligns the activation functions well.

value of a given sorted array. Depending on the outcome of the comparison the lower or the upper half is eliminated from the search space and the search is repeated with new boundaries which are set by the remaining half. In a worst case scenario, the algorithm finds the target value after $\mathcal{O}(\log(n))$ repetitions, given that the searched array has n entries (*MacMahon*, 1960).

For the calibration of the activation function the sorted array is replaced with the 10 bit range of the targeted analog parameter. Instead of the direct comparison with a target value, a condition is chosen which resembles the desired calibration outcome. For instance, the maximum output frequency of the neuron is calibrated by changing the DAC value of the refractory period τ_{refrac}, since in the limit of high frequencies the output rate mainly depends on the refractory period, c.f. with eq. (2.5). As condition for the binary search, the maximum output rate ν_{out} is measured at a high input rate and compared to a target rate ν^*_{out}. Depending on the outcome, either half of the parameter range is eliminated from the search space. For the experiment, the target rate is set to 111.3 kHz which corresponds to 256 recorded spikes over a measurement period of $T = 2.3\,\text{ms}$.

The calibration of the alignment along the x-axis is slightly more complicated. The activation function is centered if its inflection point is positioned at zero. In theory, the firing rate at the inflection point equals half the maximum rate. As shown in the simulation of the activation function in section 2.4.1, the curve can be shifted along the x-axis by changing the potential difference δV between the resting potential and the

(a) **(b)**

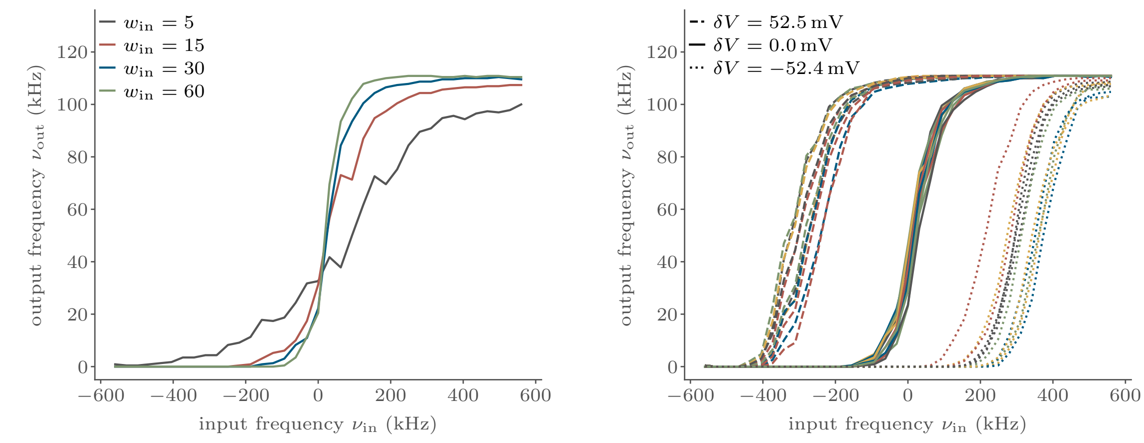


Figure 3.5: Changing the shape of the sigmoidal activation function on HICANN-DLSv2. **(a)**: The synaptic weight of the input is direct proportionality to the synaptic input current $I_{\text{syn}} \propto w\nu_{\text{in}}$ and effects the slope of the activation function. **(b)**: The activation function can be shifted along the x-axis by changing the threshold. The dotted lines represent a higher threshold and thus a negative bias whereas the dashed lines correlate to a lower threshold and a positive bias.

threshold. Since the threshold is already used as the bias parameter, the DAC-value of the resting potential will be calibrated. As condition for the binary search, the rate of the activation function is measured with no additional input except for the noise sources and is compared to the half of the maximum rate.

During the calibration of the resting potential with respect to a centered inflection point of the activation function, the positioning of the sigmoid only depends on the relative potential difference δV. The choice of a work point for the absolute threshold and resting potential should in theory be arbitrary, as the whole operating range can be shifted. On analog hardware, this does not hold true, since some circuits exhibit limited operating ranges. As a consequence the work point is fixed by a threshold potential of 0.54 V.

In the context of analog circuits it is very unlikely to find the exact target DAC-value due to a temporal variability during the measurements. However, the algorithm continuously approximates the searched value under the chosen condition and serves its purpose well. Furthermore, the calibration of the refractory period and the resting potential are repeated and interleaved several times. This is done to reduce the influence of the other yet uncalibrated parameter. In the final calibration state, the activation functions align well (see fig. 3.4b).

Synaptic Weights and Bias As shown in the simulation of the activation function in section 2.4.1, the slope of the activation function and its alignment along the x-axis can be easily modified. By lowering the synaptic weight of the input the slope declines, since the total synaptic input exhibited by the membrane is reduced. A lower synaptic weight can also be interpreted as a stretch of the x-axis in fig. 3.5a.

More importantly, the threshold can be used as the bias. Changing the potential difference δV between the threshold and the resting potential is interchangeable with adding a bias b to the synaptic input current of the neuron, c.f. eq. (2.7). In fig. 3.5b the activation function has been shifted in both directions by changing the threshold. Due to the fixed pattern noise, the shifted sigmoids do not align per neuron, but since the threshold parameter is set and trained individually the plasticity rule will correct the offset if necessary.

3.4 Experiment Setup on HICANN-DLSv2

For the classification of the Circles dataset a single hidden layer network is trained on-chip with stochastic gradient descent (SGD). The PPU executes both the forward and backward pass. This also includes generating the dataset and the input spikes. After a proper initialization routine, the training loop is started: the corresponding input rates of a randomly drawn data point $p(x, y)$ are fed into the network and the resulting output rates are evaluated by the SGD routine on the PPU. The computed parameter updates are then applied before the next iteration can start.

Forward Pass As for ANNs, the synapse must be able to seamlessly evolve from an inhibitory weight to an excitatory weight and vice versa. By design a single synapse cannot alternate between the type of the weight without changing the synapse driver's configuration which is time-consuming and can only be done row-wise. Instead of a single input row, two rows are used for the same input - one in excitatory and one inhibitory mode. In each column only one of the double-synapses is active and corresponds to the current type of the synaptic weight. As the weight changes from excitatory to inhibitory, the excitatory synapse is silenced by setting its weight to zero while the other one is reactivated.

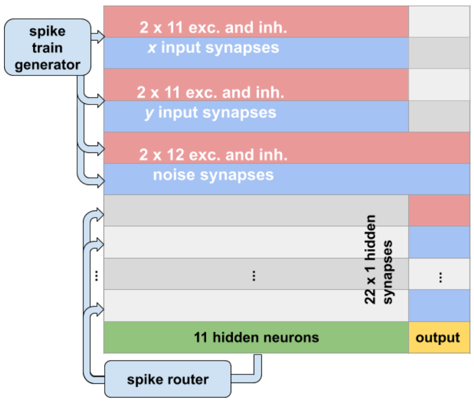


Figure 3.6: Configuration of the synapse array for the Circles data set.

As shown in fig. 3.6 the first layer of the network, the input layer, is connected to the hidden units by two rows per x and y input over eleven columns. A dedicated spike router injects the generated spikes by the hidden neurons back into the network, with each hidden neuron requiring another double-row. The feedforward connections of the hidden layer are then relayed to a single column over the 11×2 rows, where the activity is combined for a single output unit. The synaptic weights of the excitatory and inhibitory noise spike trains remain fixed throughout the experiment and require each a single row with twelve columns.

A Poisson spike train generator on the PPU provides a stream of random spikes over a certain measurement period. The provided spike resources are split into four branches, two for the excitatory and inhibitory noise source and two for both input units. Each noise branch has to supply twelve neurons with Poisson spike trains. In an attempt to reduce correlated input between the individual neurons the branches are further divided into eight channels and thus only four will share the noise with other ones.

Backward Pass In the backward pass, the output rates which have been obtained by the forward pass are used to compute the updates for weights and biases according to stochastic gradient descent (SGD). A fully on-chip implementation of the required computation comes with some compromises, such as a limited precision or not directly accessible weights. In the following, the developed workarounds are presented.

The general-purpose unit of the PPU operates with a 32 bit architecture without hardware-support for floating point types. To increase the precision of the parameter updates, the computation is bit-shifted to the left. The calculated results are then stochastically rounded and shifted back right before being applied to the network. Stochastic rounding has been proven to be a viable workaround for deep learning with parameters of limited precision (*Gupta et al.*, 2015). Typical methods of directed rounding to an integer x are rounding up $\lceil x \rceil$, down $\lfloor x \rfloor$ or to the nearest neighbor $\lceil \frac{\lfloor 2x \rfloor}{2} \rceil$. Stochastic rounding on the other hand rounds x with a probability p corresponding to the proximity to the upper or lower neighbor

$$x_{\text{stoch}}(x) = \begin{cases} \lfloor x \rfloor & \text{with } p = 1 - (x - \lfloor x \rfloor), \\ \lfloor x \rfloor + 1 & \text{with } p = 1 - \lfloor x \rfloor. \end{cases}$$

Unfortunately a direct access to the analog weight parameters from the general-purpose unit is time-consuming and computation intensive due to the lack of parallelizations. With feedback alignment, the need of the current weight information becomes irrelevant to the plasticity rule, but the computed weight update still needs to be applied to the network. This can be implemented by using the vector unit. The SIMD instructions allow parallel access to all weights and therefore it is not only used to efficiently apply the weight updates but also to silence and reactivate the respective synapses in the double-row structure of the synapse array. The *assembly* code-base for this implementation has been co-written by Sebastian Billaudelle and Benjamin Cramer.

During training the weights can max out due to their limited 6 bit range. The implementation of a simple weight regularization using the convenient parallel access of the vector unit solves this issue. The regularization of both, excitatory and inhibitory weights $w_{ij}^{\mathrm{inh}},\ w_{ij}^{\mathrm{exc}} \equiv w_{ij} \in [0, 2^6)$ is implemented using bit shifts to minimize the implementation effort and is given by

$$\mathrm{Reg}(w_{ij}) = w_{ij} - \left\lfloor \frac{\lfloor 2 \cdot w_{ij} \rfloor}{2^6} \right\rfloor .$$

The impact of the regularization is then fine tuned by applying it only with a certain probability p.

The bias is set by lowering or raising the DAC-value of the threshold potential. The resolution of the capacitive parameter memory equals roughly 1.7 mV, which in turn translates to a change of the input frequency of about 9.9 kHz. As a comparison the maximum input frequency of a neuron is around 1 MHz, but the magnitude of the typical presynaptic activity will only be at a few 100 kHz. Hence, the bias has a rather coarse resolution.

However, compared to the speed of the accelerated neuromorphic core, the Digital to Analog Converter is slow. Instead of a rapid change, a continuous and slow implementation can reduce potential negative effects of the coarse bias resolution, especially in combination with a low learning rate.

Initial Conditions and Hyperparameters The final experiment setup involves the setting of initial conditions, neuron parameters and the tuning of hyper-parameters. The latter describe a set of parameters, that is not changed during the training process but their choice often determines whether or not the training succeeds (*Goodfellow et al.*, 2016). A subset of the most relevant parameters is listed in table 3.2.

At the beginning of the experiment all weights of both layers are randomly initialized using a uniform distribution ranging from −25 to 25 LSB. A random initialization of the weights is vital to the training performance (*Goodfellow et al.*, 2016). In a worst case scenario, the initialization of the weights can even prohibit the network from learning at all.

Initially, the biases of both layers are set to zero, i.e. $\vartheta = V_{\mathrm{leak}}$. With this setting

Parameter	hidden layer	output layer
initial weights w_{ij}^{init}	$\in [-25, 25]$ LSB	$\in [-25, 25]$ LSB
initial bias b_i^{init}	$\propto \frac{1}{2} \sum_j w_{ij}$	0
learning rate of bias η_{w}	$165e4$	$5.5e4$
learning rate of bias η_{b}	$2.3e4$	$0.23e4$
resting potential V_{leak}	0.54 V	0.54 V
reset potential V_{reset}	0.01 V	0.01 V
refractory period τ_{refrac}	9 µs	0.01 V
range of threshold ϑ	0.45 V to 0.63 V	0.45 V to 0.63 V
max input frequency ν_{in}	2×500 kHz	11×111.3 kHz

Table 3.2: Initial, hyper and neuron parameters per layer. Some parameters require a different setting for the hidden and the output layer.

it has been observed that some hidden units are "silent" at the start of the training. Furthermore, these silent units remain inactive throughout the training process. As a workaround, a weight dependent term is added to the zero bias, such that $b_i^{\text{init}} \propto \sum_j w_{ij}$. This ensures an initial firing rate of every neuron at the beginning of the training and solves the issue of silently initialized hidden units.

During the tuning of the network's hyperparameters, the slope of the activation function had to be raised by a fair amount to increase the contrast of the nodes. By a steeper sigmoid, the network can create a more precise decision boundary than by a flat slope. The slope is changed by reducing the frequency of the noise spike train ν_{noise} to 1.1 kHz.

A further increase of the slope of the activation function is achieved by increasing the dynamic range of the input rates. With frequencies ranging from 0 kHz to 500 kHz almost all available resources are exploited. The maximum input rate of a hidden unit is then given by the combined rate of the x and y input. In turn, each hidden unit produces an output rate of up to 111.3 kHz, generating a potential maximum input of 11×111.3 kHz for the readout unit in the output layer.

The learning rates as chosen in table 3.2 may appear unusually high. This is mainly due to a lack of normalization during the computation of the parameter updates. In order to avoid the risk of eliminating small contributions of variables during the on-chip computation of the parameter updates, the magnitude of the individual variables has not been normalized in advance. An individual learning rate per layer and parameter copes with the different magnitudes later on. Even without the necessity to norm resulting parameter updates, it is not unusual to tune learning rates separately, since

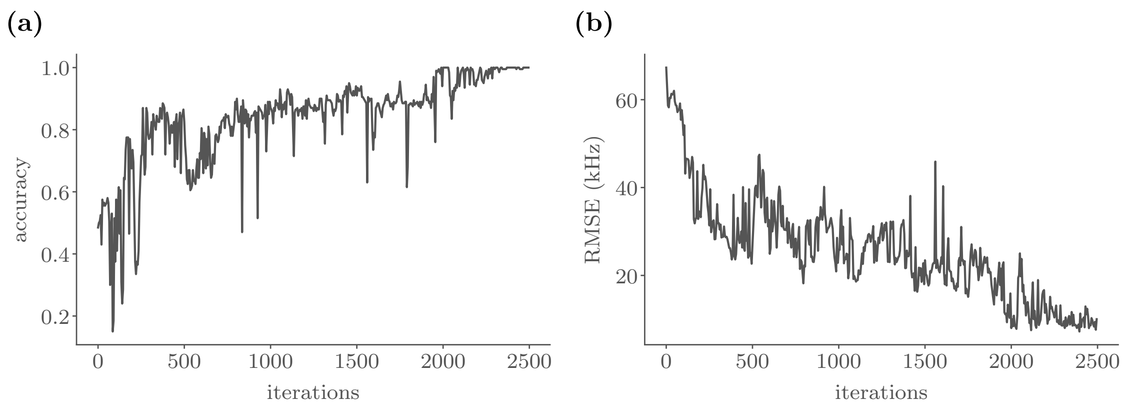


Figure 3.7: Training success of the Circles Classification. The performance is monitored by the accuracy (a) and the root mean squared error (RMSE) (b) over 2500 iterations.

different network layouts will cause varying update velocities for the individual network parameters (*Goodfellow et al.*, 2016). The final choice of the learning rates is then tuned such that a fair amount of learning activity is provided for both layers and parameters. In the appendix in appendix A.2 a detailed time evolution of the weight and bias updates per layer is shown.

3.5 Training

For a fully on-chip implementation of experiment the required input data for the training data set needs to be generated on the fly, since there is only limited memory on-chip available. In this way, each forward pass is performed with a randomly generated data point from either of the two rings from section 3.1. The backward pass is performed directly after the first training sample, i.e. the minibatch size is one.

In the following the network is trained for 2500 iterations. For a better illustration of the training process, a balanced *validation* dataset with 100 randomly drawn data points per class is chosen to validate the current state of the network after every fifth iteration. The limitation to every fifth step was chosen to reduce the measurement time to a few hours and to cope with the increasing instability of the FPGA during long measurements. To measure the performance of the training process the root mean squared error (RMSE) and the accuracy are evaluated. The RMSE is given by

$$\mathrm{RMSE} = \sqrt{\frac{\sum_p e^{(o)}(p)^2}{n_{\mathrm{points}}}}, \tag{3.4}$$

with the error $e^{(o)}$ from section 3.1 and the number of data points n_{points} in the validation dataset and the accuracy is

$$\text{Accuracy} = \frac{n_{\text{true}}}{n_{\text{points}}}, \tag{3.5}$$

with the number of correctly classified points n_{true}. Whether or not a point is correctly identified is determined by the decision boundary which was introduced in section 3.1.

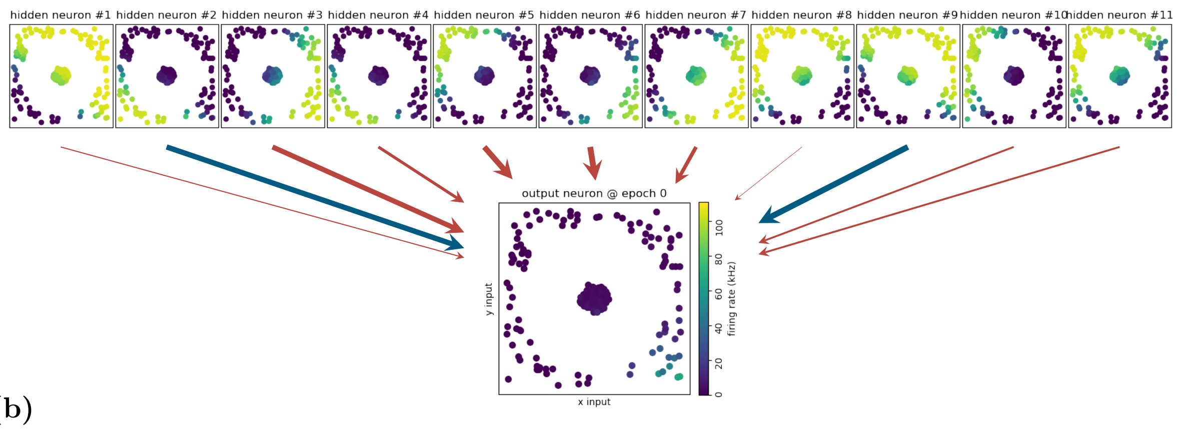
At the beginning of the training process, each hidden unit is stimulated with a linear combination of the inputs from fig. 3.2. The activation function then transforms the linear input into a non-linear output. The new representations of the input layer are again linearly combined into a net input for the output unit where another non-linear transformation is applied. The initial state of the network is shown in fig. 3.8a. At this stage of the training process the output unit cannot classify the task correctly.

The evolved network after 500 iterations is depicted in fig. 3.8b and shows first signs of improvement. The RMSE has already been reduced to half of its initial error and the accuracy has improved to roughly 80 % as shown in fig. 3.7. Less useful representations in the hidden layer are further modified by changing their bias or the connected weights. The beneficial ones are rewarded with a stronger connection to the output layer and further refinement of their bias and input weights.

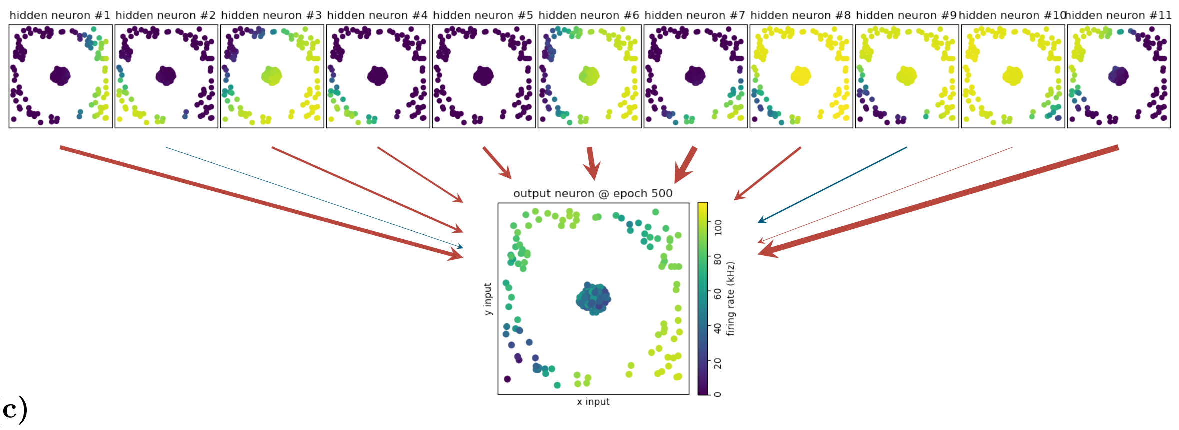
At 2500 iterations, see fig. 3.8c, the network has converged. The accuracy has also stabilized at almost 100 % (see fig. 3.7). The average over the last 500 iterations yields a mean accuracy of (98.2 ± 2.9) % and the average over the last 250 iterations improves to (99.6 ± 0.8) %. The RMSE is also reduced to a minimum, i.e. the output neuron resembles the target well. The average output error over the last 250 iterations yields (9.6 ± 1.4) kHz. To provide a better understanding of how the network evolves to improve these performance measures, the evolution and contribution of the first hidden unit is discussed in more detail.

The random initialized weights of the first hidden unit create a less beneficial representation of the input where more than half of the data points return a high output rate (fig. 3.8a). However, the initial contribution of the unit to the output unit is small. Over the next hundred iterations, the weights are adjusted such that the contour lines of the output rates are rotated clockwise by 45° (fig. 3.8b). More importantly, the bias is reduced and removes any high output rates from the inner circle. The improved representation is then rewarded with a stronger contribution to the readout unit. Until the final iteration, the nature of the representation does not change much anymore but the parameters are still fine-tuned (fig. 3.8c). The bias is slightly increased such that a greater portion of the outer circles fires with at a high rate and the orientation of the contour lines is slightly rotated back. Furthermore, the contribution of the hidden

(a)



(b)



(c)

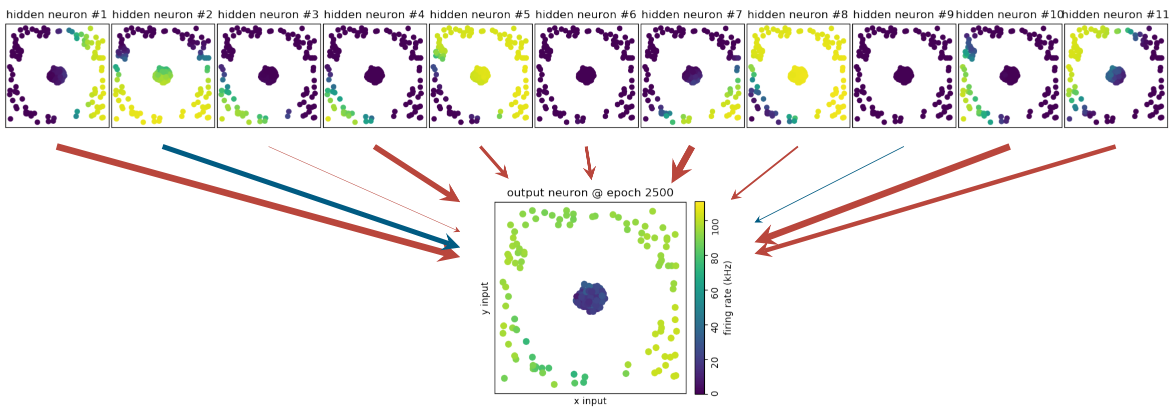


Figure 3.8: Evolution of the network's ability to separate the Circles data set. For simplicity only the hidden and output layer are displayed. The connecting arrows resemble the excitatory (*red*) and inhibitory (*blue*) synaptic connections and their thickness corresponds to the synaptic strength. **(a)** The initial untrained state of the network does not separate the inner and outer circle. **(b)** After 500 iterations the first improvements can be seen. With a decision boundary of roughly 60 kHz, the output unit starts to correctly identify the two classes. **(c)** After 2500 iterations the network has succeeded to separate both classes with an accuracy of almost 100 %.

unit is further fostered by the network.

4 SuperSpike on BSS2

The second experiment in this thesis is done on the most recent chip version of the BSS2 platform, the HICANN-Xv1. In contrast to the first experiment the neural coding scheme is changed from rate based to temporal coding. The prerequisite for learning on SNN with temporal coding is to find an appropriate plasticity rule, that copes well with the non-differential nature of individual spikes. A possible candidate, SuperSpike, was presented in section 2.4.2. The latest version of the BSS2 platform has now access to the temporal evolution of the membrane potential. This is a key element for a successful implementation of the surrogate gradient method.

To benchmark the implementation a constructed task is taken from the original publication by *Zenke and Ganguli* (2018) which is equivalent to solving the XOR problem on a time scale.

4.1 XOR-related Task

A total of 96 input units, each firing once at a fixed random spike time, is split into four overlapping collections of different size. In fig. 4.1, the various spike trains are visualized. As in the exclusive-or, the four different input patterns are assigned to two target classes which are represented by two distinct neurons in the output layer. Despite the multiple input sources, the task is by construction only two-dimensional. The first input pattern p_1 overlaps with all other patterns. The units involved in p_1 can therefore be interpreted as bias or reference units, since they will fire regardless of the active pattern. The second and third pattern do not overlap except for their bias units. In analogy to the XOR operator these patterns compare to $1 \veebar 0 = 0 \veebar 1 = 1$. Their combination then yields the fourth pattern resembling $1 \veebar 1 = 0$. In this way, p_1 and p_4 are associated with class 0 whereas the remaining two input spike trains correspond to class 1.

A feedforward pass of a single pattern spans roughly 250 µs with the random input spike times being drawn from a window of 40 µs. In a slight modification to the derivation of SuperSpike, the error metric is changed from a single target spike train S_i^* for the output i to a target time window $[t_0^*, t_1^*]$. Depending on the input pattern

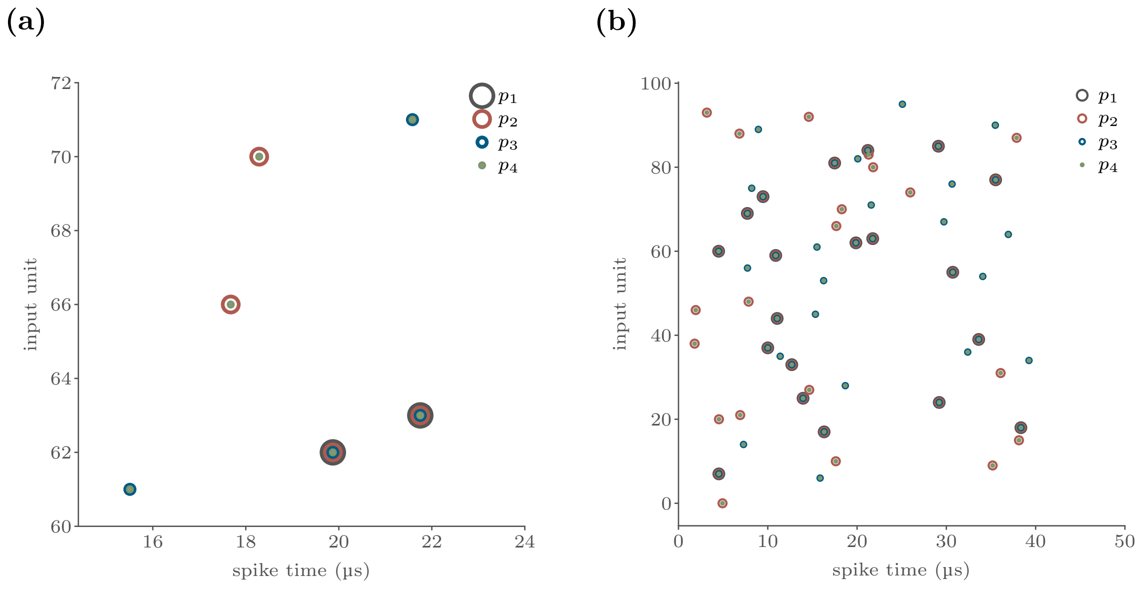


Figure 4.1: XOR-related task for SuperSpike. **(a)** A representative sector of the input data from the XOR-related task shows how the input patterns overlap. The first pattern (*black*) contains two spikes and overlaps with all other patterns. The second (*red*) and third pattern (*blue*) are disjoint apart from the bias units with four spikes each. Their combination yields the fourth input pattern (*green*) with a total of six spikes. (b) The total number of spikes in an input pattern is then multiplied by ten yielding 20, 40 or 60 input spikes respectively.

p_j, the error of the output unit i corresponding to the target class i is given by

$$e_i(t) = \begin{cases} \Big(\alpha * \big(e_{\text{outside, i}} + e_{\text{inside, i}}\big)\Big)(t), & \text{if} \quad \text{class}(p_j) == i, \\ -(\alpha * S_i)(t), & \text{else.} \end{cases} \tag{4.1}$$

In analogy to the van Rossum distance the error outside and inside the window can be written as

$$e_{\text{outside}}(t) = -S_i(t) \cdot \big(H(t_0^* - t) + H(t - t_1^*)\big) \,,$$

$$e_{\text{inside}}(t) = \begin{cases} 0, & \text{if} \quad \exists\, t_i^{(s)} \in [t_0^*, t_1^*], \\ (\epsilon * t^*)(t), & \text{else.} \end{cases}$$

with the Heaviside step function H and $H(0) = 1$ as well as the double-exponential kernels α and ϵ from section 2.4.2. Despite extending the target spike to a target time window, the error still follows the same principles as the von Rossum distance: If the target neuron spikes within the designated time span, a constant error of zero is returned. In case there occurs no spike inside the window, a target spike at time

t^* will be added to the error instead. Any spikes outside the window sum with a negative sign. Given there is spiking activity in the inactive output unit, the negative convolution of the activity with the α kernel is returned as a penalty.

4.2 Implementation on the BSS2 Platform

Unlike the previous experiment, SuperSpike is not implemented in a fully on-chip fashion. This is largely due to hardware bugs which have been discovered during the commissioning of the new HICANN-Xv1. The two most relevant issues are a reordering of the CADC channels and inaccessible spike counters from the vector unit. With the symmetric design of the chip the CADC is split into four respective quadrants, but the channels of the first and second as well as third and fourth quadrant are mingled with each other. In a scripting programming language such as Python this can be unraveled quite effortless. On the general purpose unit the programming capabilities are rather limited making it hard to develop a functioning experiment environment under these circumstances. More importantly, the software workarounds cannot be implemented on the vector unit and therefore the computation will not scale.

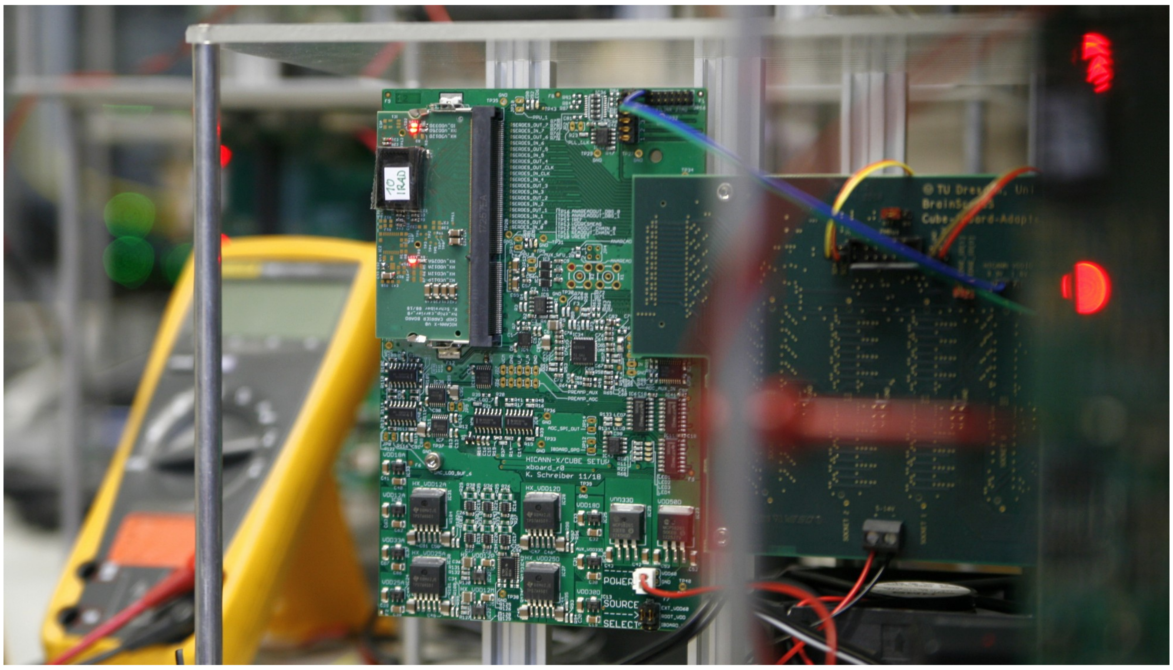


Figure 4.2: Cube setup with HICANN-Xv1 in the laboratory.

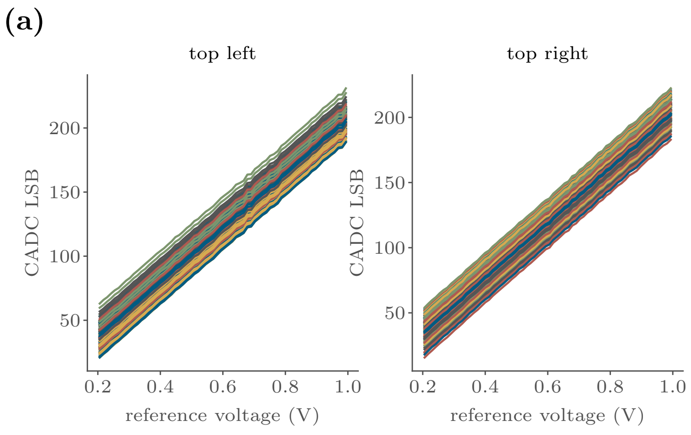
However, the main obstacle remains the lack of on-chip knowledge over spike times. In principle there are two available sources to record a spike event: the digital back end which registers every spike event precisely and the on-chip spike counters. The data transfer between the digital back end and the PPU operates at a limited speed due to a known software bug that has not yet been fixed, making its use unfeasible. The spike counters on the other hand, cannot be accessed by the vector unit since they are not properly connected. With these constraints, an efficient implementation on HICANN-Xv1 is beyond reach. Despite the handicaps, SuperSpike can still be set up as a chip-in-the-loop experiment, i.e. the weight updates are computed on the host and the forward pass is evaluated on-chip.

In the following, the details of the experiment implementation are presented. The developed software framework is based on the new software stack for the BSS2 platform, which is still under continuous development. A detailed description of the software stack's architecture is provided by *Müller et al.* (2020). Furthermore, the use of the chip is restricted to the upper half and only one PPU, to reduce the implementation effort.

CADC Readout A key element for the SuperSpike algorithm is the access to the membrane potential by the CADC. The general purpose unit instructs the vector unit to trigger the CADC and to transfer the resulting conversions back into the main memory of the PPU. At that time the software stack did not fully support the use of the vector unit. A preliminary code base has therefore been developed for the CADC readout by Aron Leibfried.

For the purpose of the experiment, the membrane traces of 128 neurons are recorded at a high sampling rate of roughly 400 kHz to ensure a suitable time resolution of the accelerated dynamics. At the same time, the available memory on the PPU restricts the maximum number of samples. Each sample uses one byte of memory and thus 100 samples of 128 neurons occupy already 12.8 kB of the available 16 kB, leaving enough space for the PPU instructions which share the same memory. In the final configuration of the CADC 100 samples are recorded per neuron with a temporal resolution of roughly 2.5 µs.

Before the CADC can be employed, its characteristic needs to be calibrated. This is done by setting various reference voltages and recording it by the CADC. Since at the minimum voltage (0.0 V) and maximum voltage (1.2 V) the CADC shows a non-linear behavior, the sweep range of the reference voltage is limited from 0.2 V to 1.0 V. Per quadrant, a slope and offset parameter of the characteristic is manually adjusted such that the full range of the LSBs is used. In addition to the per-quadrant settings, each channel has an individual offset parameter, that can be adjusted as well. Unlike the offset correction per channel, the ramp and slope parameters are not trivial to tune, since they show a sensitive cross-dependency in certain areas. A manual method was thereby preferred over an automated fit-routine. In fig. 4.3 the pre and post calibration



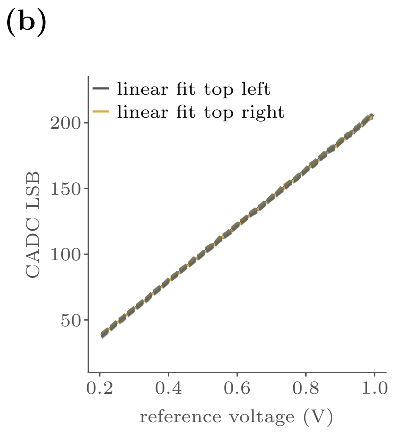


Figure 4.3: Pre and post calibration state of the CADC. **(a)** The pre calibration CADC measurement of a reference voltage ranging from 0.2 V to 1.0 V are not aligned. The slope and offset parameter differ per quadrant. Both quadrants from the upper chip half are shown. **(b)** Both quadrants are calibrated such that they share the same characteristic.

state of the characteristics are shown for both quadrants from the upper chip half. The thereby implicated conversion from DAC LSB to V will be used for most of the data shown throughout this chapter.

Calibration of LIF Neurons In chapter 3 it has already been mentioned that analog neuromorphic hardware exhibits fixed-pattern noise, which in turn can be corrected up to a certain degree by the plasticity rule in place (*Wunderlich et al.*, 2019). It is therefore a constant trade-off of between resources invested into the calibration and a potential degradation of the learning performance by less tuned analog setups.

At the time of the experiment, the development of chip-specific software for the new prototype was in an early stage. Among others, there was a lack of a calibration database and more generally, a lack of available calibrated hardware resources. This was largely due to an at the time impractical and slow calibration routine, that is based around the MADC. The MADC offers a precise measurement of an observable but does not provide any parallel readout.

The efficient parallelized readout of the CADC, on the other hand, presented itself to be a viable basis to put a more practical calibration routine into action. The main objective of the new routine is to provide a fast calibration that requires little to no interaction in order to bring a setup into a usable state with respect to the specific experiment requirements. However, the convenience of the CADC comes at the cost of the precision.

As a trade-off, only a subset of the available neuron parameters is tuned, namely the potentials of the LIF model V_{leak}, V_{reset} and ϑ. By design, the potentials of the synaptic input $V_{\text{syn, inh}}$ and $V_{\text{syn, exc}}$ have an influence on the resting potential and thus need to be considered as well. The time constants τ_{mem} and τ_{fall} are certainly important parameters in the LIF model. Motivated by the results from *Wunderlich et al.* (2019) the induced error by the misalignment of the time constants is knowingly accepted and traded for less implementation effort.

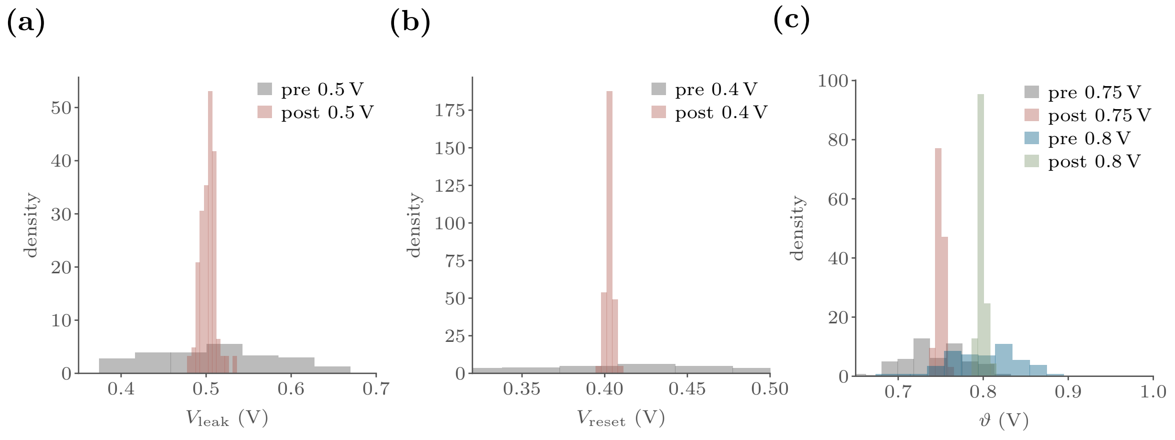


Figure 4.4: Pre and post calibration state of the analog LIF parameters. **(a)** Before calibration to 0.5 V, the resting potential shows a spread over almost 0.3 V (*gray*). **(b)** The reset potential V_{reset} is calibrated to 0.4 V (*red*). **(c)** The spread of the uncalibrated thresholds ϑ is smaller than for the leak potential (*gray* and *blue*). The potential is calibrated to 0.75 V (*red*) and 0.8 V (*green*).

Similar to the previous task, a binary search algorithm is adopted to find the proper DAC values of the analog parameters. The cross talk between individual capacitive parameter memory (capmem) cells, that arises if several cells are set to the same value, requires a workaround. By the design of the binary search algorithm, this event occurs repeatedly making the binary search a suboptimal choice for the calibration of the chip. The development of a proper gradient based calibration has already been started, but for the use case in this experiment the existing code base could be sufficiently stabilized with two minor adaptations. First, a random variation is consequently applied to all "unused" DAC-channels and second, the binary search algorithm is extended with a fall back option to the best parameter setting so far. The results of the final calibration for the LIF parameters are shown in fig. 4.4.

4.2.1 Experimental Setup

The HICANN-Xv1 is mainly controlled by a Python class called `HXBlackbox` which implements a set of convenient tools that put the chip into operation. The code has been co-written by Sebastian Billaudelle and Benjamin Cramer and is based on the new software stack for the BSS2 platform (*Müller et al.*, 2020).

In a preliminary step, a Python-based simulation frame has been developed which is oriented on the original implementation of SuperSpike by *Zenke and Ganguli* (2018). The chip-in-the-loop experiment is embedded in a new class `HXSuperSpike` combining the SuperSpike simulation with an adapted version of `HXBlackbox`. The forward pass of the simulation is then replaced by measuring the analog traces and spike times on the HICANN-Xv1 before the respective updates of the network parameters are computed in the backward pass on the host. In the following a network with a single hidden layer is set up containing 96 input sources, 30 hidden units and two output neurons which serve as target classes.

Chip-based Forward Pass The implementation of the experiment on the basis of the `HXSuperSpike` class is straight forward. The setup of the multilayer network structure is implemented by the event router and the synapse array using corresponding weights and addresses to discriminate between input and recurrent connections. The issue that arises with alternating weights for a single synapse is solved by the same double-row approach from the previous experiment implementation (section 3.4). However, in this case the workaround is already implemented within the `HXBlackbox` and the user only needs to fill a logical weight matrix W_{logical} accordingly.

$$W_{\text{logical}} = \left(\begin{array}{ccc} \left.\overbrace{\Big(I \to H \Big)}^{n_{\text{in}}}\right\} n_{\text{h}} & 0 & 0 \\ 0 & n_{\text{out}}\left\{ \overbrace{\Big(H \to O \Big)}^{n_{\text{h}}} \right. & 0 \\ 0 & 0 & 0 \end{array} \right) = \begin{pmatrix} W^{(h)} & 0 & 0 \\ 0 & W^{(o)} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The n_{in} input rows are connected to the n_{h} units of the hidden layer $(I \to H)$ by the weight matrix $W^{(h)}$. The feedforward connections of the hidden layer are mapped to n_{out} units of the output layer $(H \to O)$ by the matrix $W^{(o)}$.

The input pattern is formulated as an array containing tuples of target neuron and spike time. The forward pass is then evaluated by a designated `stimulate` function, which returns all recorded spikes at the end of the measurement. The membrane potential traces are measured by triggering the on-chip CADC readout program on the PPU as soon as the input pattern is injected by the host.

Host-based Backward Pass Within `HXSuperSpike` the analog membrane traces and recorded spike events are gathered and processed to determine the updates of the weight matrices as derived in chapter 4. In a practical approach, the integration of eq. (2.10) is done in finite temporal intervals which correspond to the length of a full CADC readout period T of roughly 252 µs. Moreover, the finite resolution of the CADC turns the integral into a sum over the time steps $dt' \approx \delta t = 2.5$ µs.

$$\begin{aligned}\Delta w_{ij}^{k} &= \eta \int_{t_k}^{t_{k+1}} dt' e_k^{(o)} \;\; \alpha * \left(\sigma'(V_{\mathrm{m},i}^{(o)}) \left(\epsilon * S_j^{(o)}\right)\right) \\ &\approx \eta \sum_{t'=0}^{T} \delta t e_k^{(o)}(t') \;\; \alpha * \left(\sigma'(V_{\mathrm{m},i}^{(o)}) \left(\epsilon * S_j^{(o)}\right)\right)(t') \qquad (4.2)\end{aligned}$$

Before the updates can be computed, it is vital that both the measured CADC traces and the recorded spike times by the digital back end are synchronized. In a respective offset measurement, the neurons are stimulated with a continuous stream of input spikes such that they fire repetitively but with a low frequency. The located peaks in the resulting membrane traces are then compared to the digitally recorded spike times, yielding an offset ΔT of (2.7 ± 1.1) µs, see fig. 4.5. In the CADC traces, the membrane potential peaks always before or exactly at the actual spike time, as the reset mechanism is triggered as soon as the threshold potential is crossed. This causes a systematic error of the evaluated peak time. A linear increase of the membrane potential would add a systematic error of half a bin width, but since the membrane potential follows rather the shape of a double exponential the systematic error is estimated to be only a quarter of a bin width. The offset is thereby corrected to $\Delta T_{\mathrm{corr}} = (2.1 \pm 1.1)$ µs.

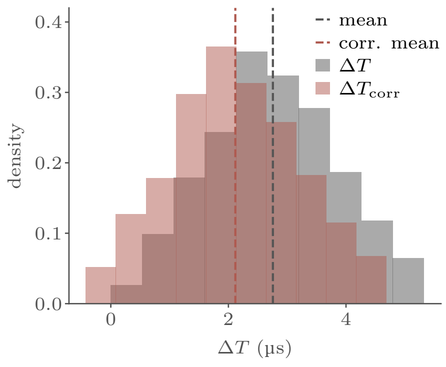


Figure 4.5: Offset measurement between CADC traces and digital spike times. The time delta between peaks in the CADC traces and the recorded spike times by the digital back end needs to be corrected.

In an attempt to optimize the performance of the Python-based simulation environment, Python bindings are used to speed up time consuming or repetitive code in C++. In particular, the stored bits of the analog CADC traces are inverted due to a hardware bug and need to be reverted before further use. Another task where C++

improves the performance is the construction of input spike trains during the forward pass. In the final setup, the overall runtime has been improved by about 25 percent.

Initial Conditions and Neuron Parameters The chosen initial conditions and neuron parameters for this experiment are inspired by the configuration used in *Zenke and Ganguli* (2018). However, some adaptations and a respective time-scaling have to be made, to cope with the analog hardware's acceleration factor of 10^3.

Parameter	HICANN-Xv1	SuperSpike[1]
membrane constant τ_{mem}	8 µs	10 ms
synaptic constant τ_{fall}	5 µs	5 ms
refractory period τ_{refrac}	30 µs	5 ms
α kernel constant τ_α	12 µs	10 ms
ϵ kernel constant τ_ϵ	12 µs	10 ms

Table 4.1: Time constants used for SuperSpike. Shown is a comparison between the tuned hardware parameters used in this thesis and the original simulation.

A LIF neuron is fully described by a set of time constants and neuron potentials. Their specific values have a significant impact on the training performance. If for instance the time constants are chosen too short, the individual parts of the learning rule in eq. (4.2) do not overlap anymore and will yield a vanishing weight update, despite an incorrect spiking behavior of the network. On the other hand, longer time constants will blur the network's temporal perception. The time constants used in the backward pass are shown in the table 4.1 and resemble an average estimate of the detuned hardware time constants of the recorded membrane traces.

In the given task, the neurons are not required to fire multiple times. The refractory period has therefore been increased to 30 µs to suppresses repetitive firing patterns. Another way of avoiding repetitive firing patterns is to set the reset potential far below the equilibrium state.

The distance between the resting potential and the threshold defines the dynamic range of a non-spiking membrane. The choice of the distance follows two main motivations. A larger gap increases the dynamic range and therefore improves the signal-to-noise ratio (SNR) of the CADC traces. At the same time, a greater potential difference between threshold and resting potential requires a stronger synaptic input current to trigger a fire response. A single input spike with a maximum excitatory weight and time constants set as in table 4.1 increases the membrane potential by approximately

[1] Simulation parameters for SuperSpike can be found in *Zenke and Ganguli* (2018)

0.3 V. The final setting is a compromise of both assuring that the 6 bit weights are not maxing out during training and setting the potentials as far apart as possible. Since the overall input activity of the output layer is reduced compared to the hidden layer, the threshold of the output unit is further lowered by 0.05 V. The choice of the neuron potentials for the hidden and output layer is listed in table 4.2.

Parameter	hidden layer	output layer
initial weights w_{ij}^{init}	$\in \mathcal{N}(0, 13)$	$\in \mathcal{N}(0, 18)$
resting potential V_{leak}	0.5 V	0.5 V
reset potential V_{reset}	0.4 V	0.4 V
threshold ϑ	0.75 V	0.7 V
learning rate η	2000	30
slope of fast sigmoid β_σ	$20\,\text{V}^{-1}$	$20\,\text{V}^{-1}$

Table 4.2: Initial, hyper and neuron parameters per layer.

In addition to a proper choice of the neuron parameters, the performance of the training depends on the initialized weights and the chosen learning rate (*Goodfellow et al.*, 2016). In analogy to the simulation of SuperSpike, the weights are drawn from a normal distribution which is centered around zero (*Zenke and Ganguli*, 2018). The learning rates are set, such that weights in both layers are adapted during the learning but at the same time do not cause rapid weight updates that would in turn require a regularization to prevent the weights from maxing out.

Another tunable parameter in the SuperSpike plasticity rule is the slope of the auxiliary function $\sigma(V)$, which is defined in eq. (2.9). The choice of the parameter depends on the distance between the threshold and the resting potential. The slope is chosen such that at rest the surrogate gradient $\sigma'(V)$ provides little to no contribution and reaches its maximum when the membrane is about to cross the threshold.

Network Observables In fig. 4.6 the relevant traces of the forward and backward pass are shown for the final parameter settings. The first column shows raster plots of the incoming spike trains for the hidden (a) and output (d) layer, respectively. Their contribution to the membrane potential depends on the type and strength of the respective synapse. The second column depicts the recorded membrane traces of the hidden (b) and output layer (e). Despite the calibration of the membrane potential, the fixed-pattern noise of the chip is still clearly visible. The input of the output layer (d) resembles the spikes generated in the hidden layer, visible from the threshold crossings and subsequent membrane reset (c).

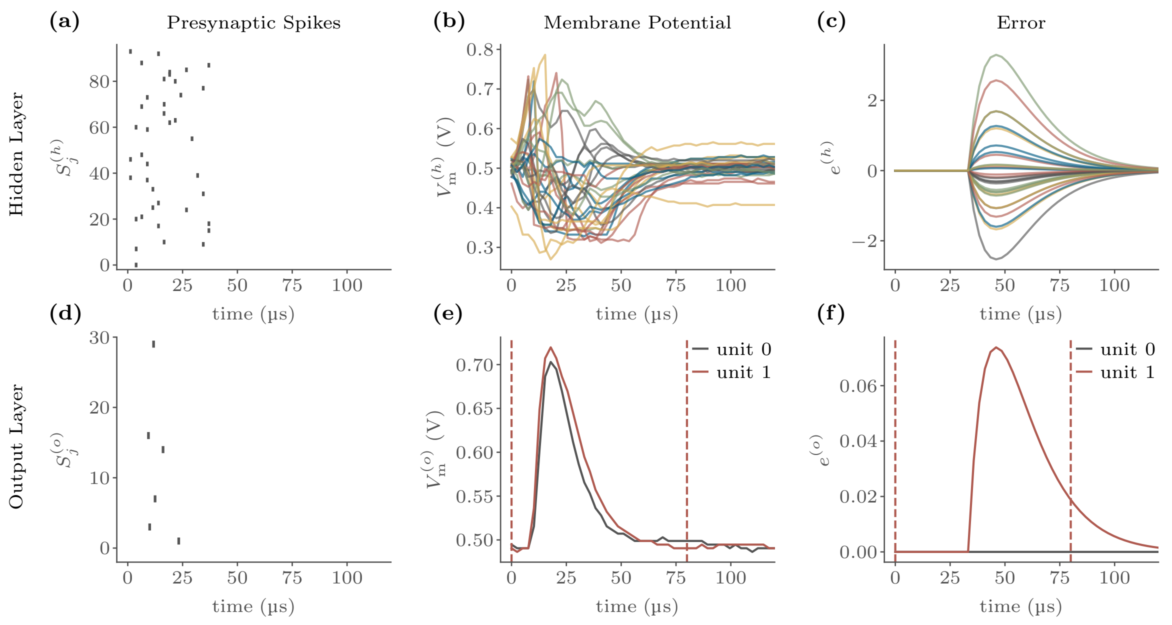


Figure 4.6: Monitoring of the network on HICANN-Xv1. Each column represents an important observable for the backward pass. The hidden and output layer are discriminated row-wise. The first column, contains the information of the presynaptic spike trains $S_j^{(l)}$, the second column shows the analog traces of the membrane potential V_m with respect to the total synaptic input generated by the incoming spikes. The third column displays the adapted van Rossum error $e^{(l)}$.

The third column shows the error as given in eq. (4.1). The dashed vertical lines in the panes (e, f) indicate the time window within which the target output unit needs to spike. The interval is set in accordance with the duration of the input pattern and the neuron's time constants from 0 µs to 80 µs. As indicated by the color of the dashed lines, unit 1 is required to fire, but the membrane potential falls short of doing so (e). In this case, a target spike t^* indicates where the silent output unit should have fired. The time is chosen to be near the end of the input pattern at 33 µs. Depending on the preferred propagation method, the error of the output unit (f) is then relayed to the hidden unit (c) either by backpropagation or feedback alignment.

As already introduced in section 2.4.2, the final weight update depends on the error, a postsynaptic and a presynaptic term. The post and presynaptic term can be combined into *eligibility* traces $\lambda_{ij}^{(o)}$

$$\lambda_{ij}^{(o)} = \alpha * \Big(\underbrace{\sigma'(V_{\mathrm{m},i}^{(o)})}_{\text{Post}} \underbrace{(\epsilon * S_j^{(o)})}_{\text{Pre}} \Big),$$

which can be interpreted as a memory over presynaptic events and how close the membrane is to spike . This memory then influences the update of the weight by how

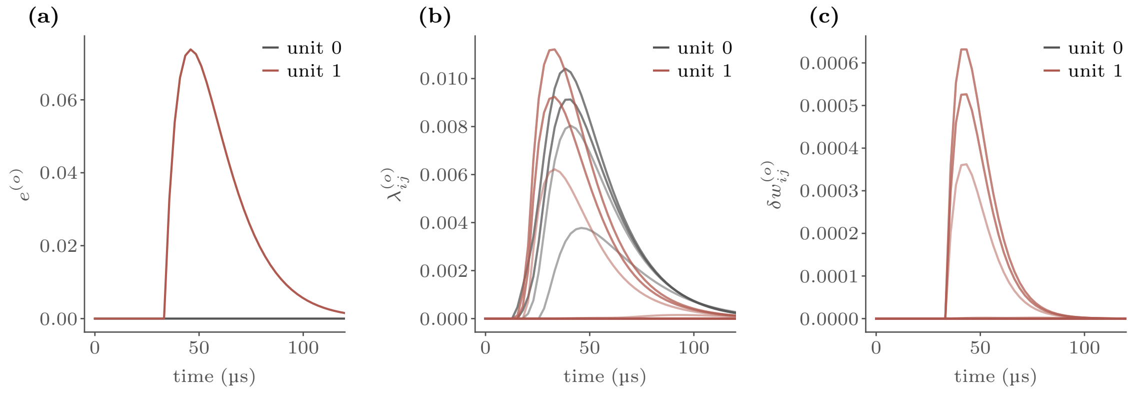


Figure 4.7: Weight update in the backward pass using SuperSpike. **(a)** The first contribution to the weight update is given by the adapted von Rossum error $e^{(o)}$. **(b)** The convolution of the surrogate gradient with the presynaptic spike activity yields the eligibility traces $\lambda_{ij}^{(o)}$. **(c)** The final weight update $\Delta w_{i,j}^{(o)}$ is then given by the integral over $\delta w_{i,j}^{(o)}$ which is the product of (a) and (b).

"eligible" it is (c.f. *Sutton and Barto*, 2011). The composition of the weight update for the given example in fig. 4.6 is shown in fig. 4.7

4.3 Training and Results

A hidden layer network is trained on the classification task with SuperSpike using backpropagation or feedback alignment as propagation methods in the backward pass. Each method is trained for ten trials with a balanced training data set consisting of 2000 minibatches.

A minibatch is created by randomly drawing eight patterns from a set of four fixed input patterns with a uniform probability (c.f. fig. 4.8). The individual batches are not necessarily balanced, but the overall training data set will be. A single input pattern spans over 40 µs, but the evaluation of the pattern in the forward pass takes at least 250 µs. After the CADC traces have been read out on the host, the next pattern can be injected. As soon as all patterns of a minibatch have been processed, the gathered information is evaluated in the backward pass yielding the respective weight updates for the synaptic connections of the network. A trial is completed once all minibatches have been processed. The next trial will then be started with a new seed for the random initial parameters.

The performance of the different propagation methods is measured by using a validation data set. After every minibatch, the updated network is evaluated by testing each pattern exactly one time. As in the previous experiment the accuracy is determined by

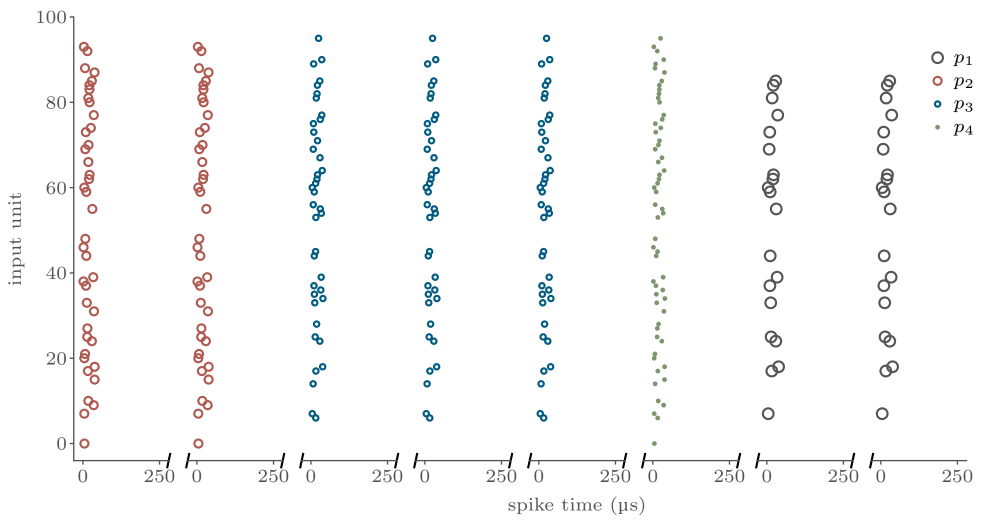


Figure 4.8: Example of a training minibatch. The chosen batch-size of the minibatches is eight. The break in the x-axis indicates the time needed to transfer the recorded CADC data from the PPU memory and the spike times from the digital back end to the host after an input pattern has been processed by the forward pass.

the fraction of correctly identified inputs n_{true} over the total number of inputs n_{inputs} in the data set. In the following accuracy-figures, the overall accuracy is determined by the 0.5-quantile over the various trials, i.e. the median. Moreover, the upper and lower error bounds are set by the 0.2-quantile and 0.8-quantile, respectively, covering 60 % of the data. For better displaying the displayed data is further resampled from 2000 to 500 samples using a polyphase method (c.f. *SciPy*, 2020).

Backpropagation vs. Feedback Alignment First, the standard backpropagation method is compared to feedback alignment, where instead of the transpose of the connecting weight matrix a random matrix is taken to propagate the error from the output to the hidden layer (c.f. chapter 4). The random matrix is drawn from either a uniform distribution ranging from -25 to 25 or from a centered normal distribution $\mathcal{N}(0, 20)$.

The performance of the training aligns well with the performance of the chosen validation data set (see fig. 4.9). As expected, backpropagation converges significantly faster than feedback alignment. Interestingly the choice of the underlying distribution of the random matrix does not have a great impact on the performance of the training, at least on the setup in use. Both distributions converge at the same velocity, despite an observed sensitivity on the range of the distributions during the parameter tuning.

(a) **(b)**

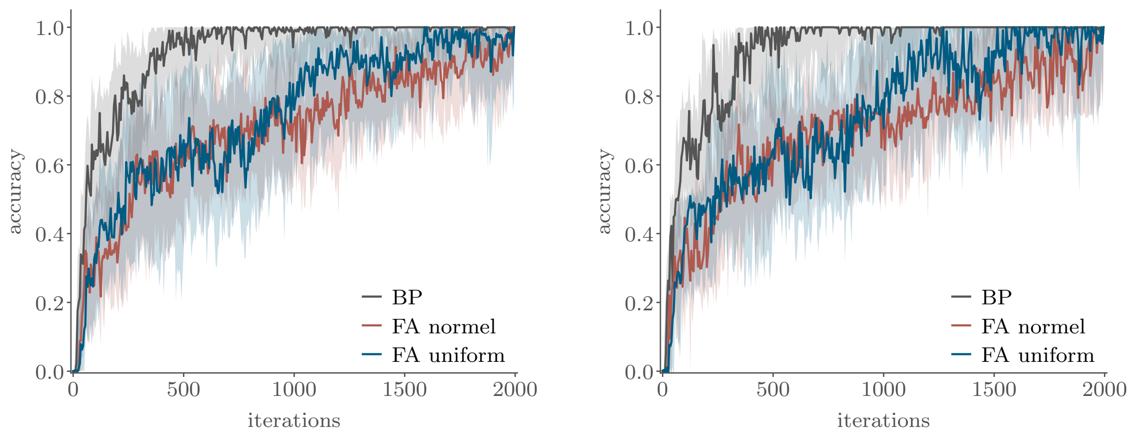


Figure 4.9: Train and validation accuracy of SuperSpike using backpropagation (BP) and feedback alignment (FA). **(a)** The training accuracy for backpropagation (*black*) converges faster than for feedback alignment (*red* and *blue*). The underlying distribution of the randomly drawn feedback weights, normal (*red*) versus uniform (*blue*), does not effect the performance. **(b)** The accuracy for the chosen validation data set does not differ much from the training accuracy.

The Necessity of Hidden Learning As a sanity check, the learning rate of the hidden layer is set to zero. This validates, that the initial state of the hidden layer is not solving the task through the random weight initialization and thus the setup does not even require hidden learning. In a more drastic approach the whole hidden layer is removed. According to the theory, the network should not be able to perform under these circumstances as the XOR-related classification task specifically requires a hidden layer structure with a non-linear activation function (c.f. *Goodfellow et al.*, 2016).

The performance of the setup with the applied limitations for the hidden layer is shown in fig. 4.10. With the inability to train the hidden layer, the accuracy does not improve by much. The initial conditions are set in a way that only little activity is generated by the hidden layer. In the following, the output layer receives only few spikes and fails to build a fire response upon them. The median accuracy settles around 25 % which reflects the correct classification of one out four input patterns. In case of the removed hidden layer, the median accuracy also settles around the same accuracy. The upper error bound is limited to 50 % which reflects the nature of the XOR-related task. In the best case only two out of four patterns are correctly identified without a hidden layer. All in all this confirms the necessity of hidden learning for the chosen classification task. Not only is a deep network structure required to solve the task, but also training activity within the hidden layer.

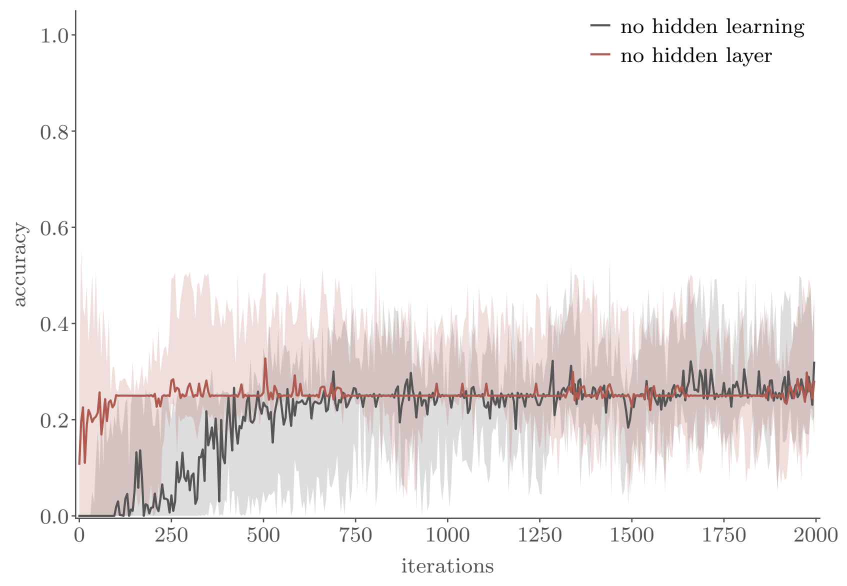


Figure 4.10: Performance of SuperSpike without hidden learning or hidden layer. The accuracy without the possibility to train the hidden layer remains low, as the initial state does not allow many spikes to pass the hidden layer. For a missing hidden layer, the accuracy settles around 25 %. The upper bound is limited to 50 % due to the nature of the XOR related task.

Transferability between Chips Throughout the development process of the experimental setup for SuperSpike several HICANN-Xv1 setups have been in use. During this phase, the differences between the individual chips could be noticed to some extent, especially when hand tuned calibration routines had to be redone for a new setup. In a direct comparison of the calibrated membrane traces, the second chip is less pre-

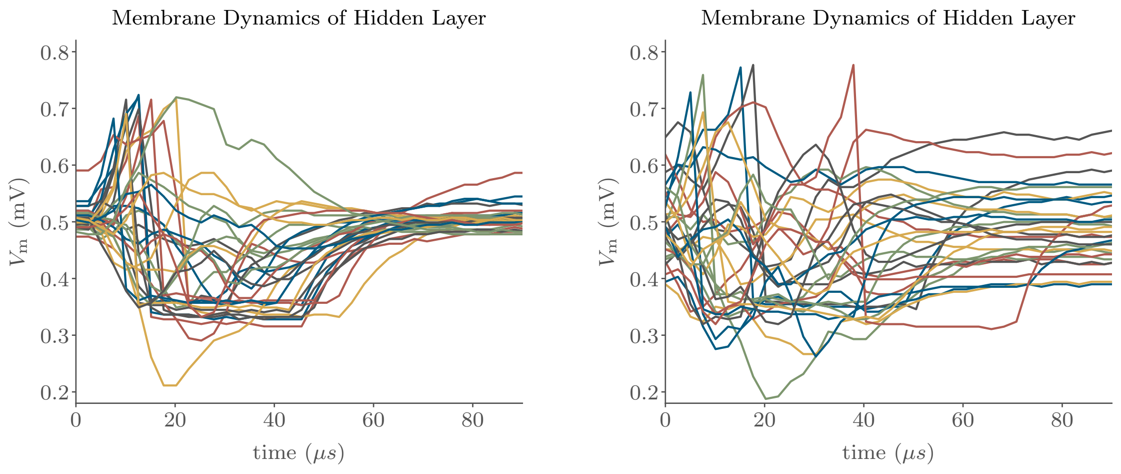


Figure 4.11: Membrane trace comparison between HICANN-Xv1 setups. The calibration of the resting potential for the setup “63” (right) did not work as well as for “73” (left).

HICANN-Xv1	average Accuracy (25 steps)	average Accuracy (100 steps)
no. 73	$100.0^{+0.0}_{-5.2}$ %	$100.0^{+0.0}_{-5.8}$ %
no. 63	$100.0^{+0.0}_{-11.2}$ %	$100.0^{+0.0}_{-17.1}$ %

Table 4.3: Performance of SuperSpike using backpropagation. The average validation accuracy is evaluated by averaging the median accuracy over the last 25 respectively 100 steps. The uncertainty bounds are given by the mean 0.1-quantile in the subscript and by the mean 0.9-quantile in the superscript.

cise. Specifically the resting potential is not well aligned as the membrane potential spans over almost 0.3 V at rest (fig. 4.11). As a consequence, the threshold level in the hidden and output layer has been increased by 0.05 V. Overall, the performance appeared to not suffer too much. To verify this, the presented data from above is remeasured on a different setup. In fig. 4.12 the individual performance of each setup is compared per propagation method.

Given the fair amount of spread in the resting potential, the second chip performs still reasonably well. As shown in table 4.3, both setups achieve the same median accuracy with backpropagation, but the lower error bound of the “73” setup is clearly smaller than for the other setup. In addition, the uncertainty increases only marginally on “73” if more steps are taken into account, whereas on “63” the average lower bound over 100 steps is at 82.9 %. The lower precision of the worse calibrated setup coincides with a visibly lower convergence speed as shown in fig. 4.12a.

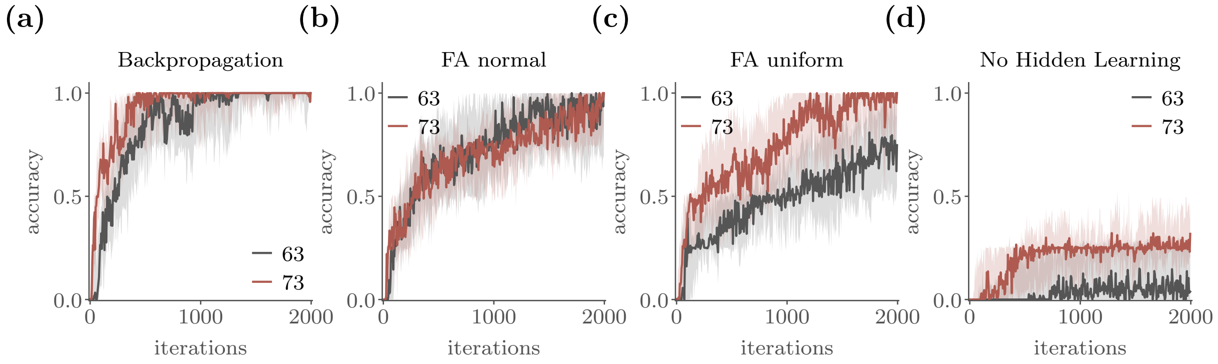


Figure 4.12: Setup transferability for SuperSpike. The HICANN-Xv1 setups “63” and “73” are compared to each other. The performances of various propagation methods such as backpropagation (a), feedback alignment with a normal distribution (b) and feedback alignment with a uniform distribution (c) are compared. In the last pane (d) the performance of no hidden learning is shown for both setups.

The performance of a random propagation matrix based on a normal distribution (fig. 4.12b) is almost identical for both chips. If feedback alignment is performed with a uniformly drawn random matrix (fig. 4.12c) the detuned chip converges slower. For this specific task, normally distributed random weights are more robust to fixed-pattern noise than uniformly drawn ones.

All in all, the transferability between setups can not only be subjectively observed, but actually works without causing too many issues. One should keep in mind, that the calibration routine in use is heavily exposed to the cross-talk problematic, which can differ from setup to setup. Furthermore, the reduction of the fixed-pattern noise is limited to the neuron potentials. With the use of a more extensive and robust calibration routine, the performance and transferability will most certainly be increased.

5 Discussion and Outlook

Within this thesis, the implementation of supervised deep learning methods for SNNs were demonstrated on analog neuromorphic hardware. A rate-based approach was successfully executed as an on-chip experiment on the HICANN-DLSv2, classifying the Circles data set with an accuracy of $(99.6 \pm 0.8)\%$. A second temporal-based method, SuperSpike, was implemented on a full-size chip version of the BSS2 platform and performed well at solving an XOR-related problem on two distinct HICANN-Xv1 setups.

Rate Coding with SGD An on-chip realization of gradient descent for a rate-based SNN was developed for the HICANN-DLSv2 chip. Despite continuing stability issues of the FPGA and imperfections of the analog hardware, the Circles dataset could be classified with an accuracy of $(99.6 \pm 0.8)\%$. This experiment resembles the first successful on-chip implementation of a deep learning task requiring a hidden layer structure and a non-linear activation on the analog BSS2 platform. However, transferability of the results to other setups has not yet been achieved, since an extensive fine-tuning of the new setup would have been required first.

A large improvement of the implementation could be achieved by transferring the experiment to the most recent chip version. The HICANN-Xv1 features dedicated on-chip noise-generators with which a sigmoidal activation function can easily be realized. Additionally, the difficulty of the task can be increased: with the newly available neuron resources, 512 instead of 32, the network can be scaled up with respect to the number and size of the individual layers. With the promising prospects given by the new chip no further improvements are planned for the current implementation on the HICANN-DLSv2. Moreover with the arrival of the HICANN-Xv1 the research focus within the group has shifted. To that end, the software development for the HICANN-DLSv2 has been halted. The older chip is still capable of performing remarkably well, but its successor outperforms it in several regards.

Temporal Coding with SuperSpike Unlike the previous experiment, SuperSpike was not implemented in a fully on-chip fashion. This is largely due to hardware bugs in the current chip revision which have been discovered only recently during the commissioning of the manufactured chip. The two most relevant issues are an incorrect wiring of the CADC channels and not directly accessible spike counters from the vector unit. The mingled CADC channels cannot be unraveled using the vector unit. A software workaround on the general purpose unit, on the other hand, will not scale and thus greatly affect the processing speed of the backward pass.

However, the main obstacle remains the lack of spike times on the PPU. In principle there are two possibilities to record spike events: the digital back end and the on-chip spike counters. The access of the digital back end by the PPU operates at a limited speed due to a known software bug and has not yet been fixed, making its use unfeasible. On the contrary, the spike counters cannot be directly accessed by the vector unit. As before a solution based around the general purpose unit does not scale and simply will not provide the necessary time resolution of the spike times. The aforementioned problems will be fixed in a future tape-out. With the new revision a fully on-chip implementation of SuperSpike will be possible. In the meantime a chip-in-the-loop approach is taken.

The host-supported implementation of SuperSpike on the HICANN-Xv1 successfully solved an XOR-related benchmark with an average test accuracy of $100.0^{+0.0}_{-5.2}\,\%$ using backpropagation. Motivated by the results from *Wunderlich et al.* (2019) only the neuron potentials have been calibrated. The performance was reproduced on another setup with a similar test accuracy of $100.0^{+0.0}_{-11.2}\,\%$ despite significantly detuned neuron potentials. In addition, the convergence of SuperSpike using feedback alignment was also shown for both setups. It can be concluded that the SuperSpike learning rule is a robust supervised training method for deep SNNs on analog neuromorphic hardware. In particular it has been shown that the algorithm does not require either calibrated time constants nor perfectly aligned neuron potentials for the given task.

Future Projects In the meantime, the developed framework for the hardware implementation of SuperSpike has been embedded in a PyTorch environment by Sebastian Billaudelle and Benjamin Cramer. Early results have shown a competitive test accuracy of over 96 % for the MNIST dataset. A more profound investigation of these preliminary results will be published in the next months.

Furthermore, the collaboration with Friedemann Zenke from the Friedrich Miescher Institute for Biomedical Research in Basel, author of the SuperSpike learning rule, will be continued. In a follow-up project the application of SuperSpike for real world problems such as speech recognition will be investigated by training recurrent SNNs. In an attempt to bridge the gap between biological time constants of a neuron (milliseconds) and the time constants of sensory input data (seconds), the slow neuronal adaption variable provided by the AdEx neuron model will be used.

References

Aamir, S. A., P. Müller, G. Kiene, L. Kriener, Y. Stradmann, A. Grübl, J. Schemmel, and K. Meier, A mixed-signal structured AdEx neuron for accelerated neuromorphic cores, *IEEE Transactions on Biomedical Circuits and Systems*, *12*(5), 1027–1037, doi:10.1109/TBCAS.2018.2848203, 2018a.

Aamir, S. A., Y. Stradmann, P. Müller, C. Pehle, A. Hartel, A. Grübl, J. Schemmel, and K. Meier, An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(12), 4299–4312, doi:10.1109/TCSI.2018.2840718, 2018b.

Akopyan, F., et al., Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip, *IEEE transactions on computer-aided design of integrated circuits and systems*, *34*(10), 1537–1557, 2015.

An, G., The effects of adding noise during backpropagation training on a generalization performance, *Neural computation*, *8*(3), 643–674, 1996.

Averbeck, B. B., Poisson or not poisson: differences in spike train statistics between parietal cortical areas, *Neuron*, *62*(3), 310–311, 2009.

Azevedo, F. A., L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain, *Journal of Comparative Neurology*, *513*(5), 532–541, 2009.

Bi, G. Q., and M. M. Poo, Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type., *The Journal of neuroscience : the official journal of the Society for Neuroscience*, *18*(24), 10,464–10,472, 1998.

Billaudelle, S., et al., Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate, *arXiv preprint arXiv:1912.12980*, 2019.

Bliss, T. V., and T. Lømo, Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path, *The Journal of physiology*, *232*(2), 331–356, 1973.

BrainScaleS, Research project, `http://brainscales.kip.uni-heidelberg.de/public/index.html`, 2012.

Brunel, N., Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons, *Journal of computational neuroscience*, *8*(3), 183–208, 2000.

Cass, S., Taking ai to the edge: Google's tpu now comes in a maker-friendly package, *IEEE Spectrum*, *56*(5), 16–17, 2019.

Dale, H., Pharmacology and nerve-endings, 1935.

Davies, M., et al., Loihi: A neuromorphic manycore processor with on-chip learning, *IEEE Micro*, *38*(1), 82–99, 2018.

Debanne, D., A. Bialowas, and S. Rama, What are the mechanisms for analogue and digital signalling in the brain?, *Nature Reviews Neuroscience*, *14*(1), 63–69, 2013.

Drachman, D. A., Do we have brain to spare?, 2005.

Friedmann, S., J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier, Demonstrating hybrid learning in a flexible neuromorphic hardware system, *IEEE Transactions on Biomedical Circuits and Systems*, *11*(1), 128–142, doi:10.1109/TBCAS.2016.2579164, 2017.

Furber, S. B., F. Galluppi, S. Temple, and L. A. Plana, The spinnaker project, *Proceedings of the IEEE*, *102*(5), 652–665, 2014.

Gerstner, W., R. Kempter, J. L. Van Hemmen, and H. Wagner, A neuronal learning rule for sub-millisecond temporal coding, *Nature*, *383*(6595), 76–78, 1996.

Gerstner, W., W. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics*, Cambridge University Press, 2014.

Goodfellow, I., Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, `http://www.deeplearningbook.org`, 2016.

Grossberg, S., Competitive learning: From interactive activation to adaptive resonance, *Cognitive science*, *11*(1), 23–63, 1987.

Gupta, S., A. Agrawal, K. Gopalakrishnan, and P. Narayanan, Deep learning with limited numerical precision, in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

Hebb, D. O., *The organization of behavior: a neuropsychological theory*, J. Wiley; Chapman & Hall, 1949.

Hock, M., A. Hartel, J. Schemmel, and K. Meier, An analog dynamic memory array for neuromorphic hardware, in *Circuit Theory and Design (ECCTD), 2013 European Conference on*, pp. 1–4, doi:10.1109/ECCTD.2013.6662229, 2013.

Iberri, D., Schematic of an action potential, 2020.

Jarosz, Q., Schematic figure of a neuron, 2009.

Lapicque, L., Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization, *Journal de Physiologie et Pathologie General*, *9*, 620–635, 1907.

Lillicrap, T. P., D. Cownden, D. B. Tweed, and C. J. Akerman, Random synaptic feedback weights support error backpropagation for deep learning, *Nature communications*, *7*(1), 1–10, 2016.

MacMahon, P. A., *Combinatory analysis*, repr. ed., Getr. Zählung pp., Chelsea Publ., New York, repr. d. Ausg. Cambridge Univ.-Pr., 1915 - 1916, 1960.

Markram, H., J. Lübke, M. Frotscher, and B. Sakmann, Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps, *Science*, *275*(5297), 213–215, 1997.

Marsaglia, G., et al., Xorshift rngs, *Journal of Statistical Software*, *8*(14), 1–6, 2003.

Mayer, R., and H.-A. Jacobsen, Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools, *ACM Computing Surveys (CSUR)*, *53*(1), 1–37, 2020.

Müller, E., C. Mauch, P. Spilger, O. J. Breitwieser, J. Klähn, D. Stöckel, T. Wunderlich, and J. Schemmel, Extending brainscales os for brainscales-2, *arXiv preprint*, 2020.

Nabavi, S., R. Fox, C. D. Proulx, J. Y. Lin, R. Y. Tsien, and R. Malinow, Engineering a memory with ltd and ltp, *Nature*, *511*(7509), 348–352, 2014.

Petrovici, M. A., *PhD thesis*, University of Heidelberg, in preparation, 2012.

Petrovici, M. A., J. Bill, I. Bytschok, J. Schemmel, and K. Meier, Stochastic inference with spiking neurons in the high-conductance state, *Physical Review E*, *94*(4), doi:10.1103/PhysRevE.94.042312, 2016.

Pfeiffer, M., and T. Pfeil, Deep learning with spiking neurons: opportunities and challenges, *Frontiers in neuroscience*, *12*, 774, 2018.

Rieke, F., D. Warland, R. D. R. Van Steveninck, W. S. Bialek, et al., *Spikes: exploring the neural code*, vol. 7, MIT press Cambridge, 1999.

Schemmel, J., Brainscales 2: A novel architecture for analog accelerated neuromophic computing including hybrid plasticity, *internal document*, 2017.

Schemmel, J., S. Billaudelle, P. Dauer, and J. Weis, Accelerated analog neuromorphic computing, *arXiv preprint arXiv:2003.11996*, 2020.

SciPy, Website, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample_poly.html`, 2020.

Smithson, M., *Confidence Interval*, pp. 283–284, Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-04898-2_183, 2011.

Stradmann, Y., Verification and commissioning of mixed-signal neuromorphic substrates, Master's thesis, Ruprecht-Karls-Universität Heidelberg, 2019.

Sutton, R. S., and A. G. Barto, Reinforcement learning: An introduction, 2011.

Tavanaei, A., M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, Deep learning in spiking neural networks, *Neural Networks*, *111*, 47–63, 2019.

van Rossum, M. C. W., A novel spike distance, *Neural Computation*, *13*(4), 751–763, 2001.

Wunderlich, T., et al., Demonstrating advantages of neuromorphic computation: A pilot study, *Frontiers in Neuroscience*, *13*, 260, doi:10.3389/fnins.2019.00260, 2019.

Zenke, F., and S. Ganguli, Superspike: Supervised learning in multilayer spiking neural networks, *Neural computation*, *30*(6), 1514–1541, 2018.

List of Acronyms

ADC	Analog to Digital Converter.
AdEx	Adaptive-Exponential Integrate-and-Fire.
ANN	artificial neural network.
BSS	BrainScaleS.
BSS1	BrainScaleS1.
BSS2	BrainScaleS2.
CADC	Columnar Digital to Analog Converter.
DAC	Digital to Analog Converter.
HICANN-DLSv2	HICANN with Digital Learning System.
FPGA	field programmable gate array.
HICANN	High Input Count Analog Neural Network.
HICANN-Xv1	HICANN with Digital Learning System and Hagen eXtensions.
ISI	interspike interval.
LIF	Leaky Fire-and-Integrate.
LTD	Long-Term Depression.
LTP	Long-Term Potentiation.
MADC	Multiplying ADC.
MLP	multi layer perceptron.
PPU	plasticity processing unit.
PSP	postsynaptic potential.

ReLu rectified linear unit.

RMSE root mean squared error.

SGD stochastic gradient descent.

SIMD Single Instruction Multiple Data.

SNN spiking neural network.

SRAM Static Random Access Memory.

STDP Spike-Timing-Dependent Plasticity.

List of Figures

List of Tables

A Appendix

A.1 Activation Function Parameters

Parameter	Simulation	Hardware
noise weight w_{noise}	8	15
input weight w_{in}	4	30
input rate ν_{in}	−1000 kHz to 1000 kHz	−560 kHz to 560 kHz
noise rate ν_{noise}	240 kHz	70 kHz
resting potential V_{leak}	0 V	0.54 V
reset potential V_{reset}	−200 V	0.01 V
threshold ϑ	−4.25 V to 5.75 V	0.45 V to 0.63 V
refractory period τ_{refrac}	100 µs	9 µs
synaptic time constant τ_{fall}	30 µs	5 µs
membrane time constant τ_{mem}	3 µs	5 µs

Table A.1: Parameter setting for a sigmoidal activation function of the LIF neuron. The different parameters for the simulated and on hardware recorded activation function are shown. The choice of the unit for the simulated potentials is arbitrary.

A.2 Monitoring Plots

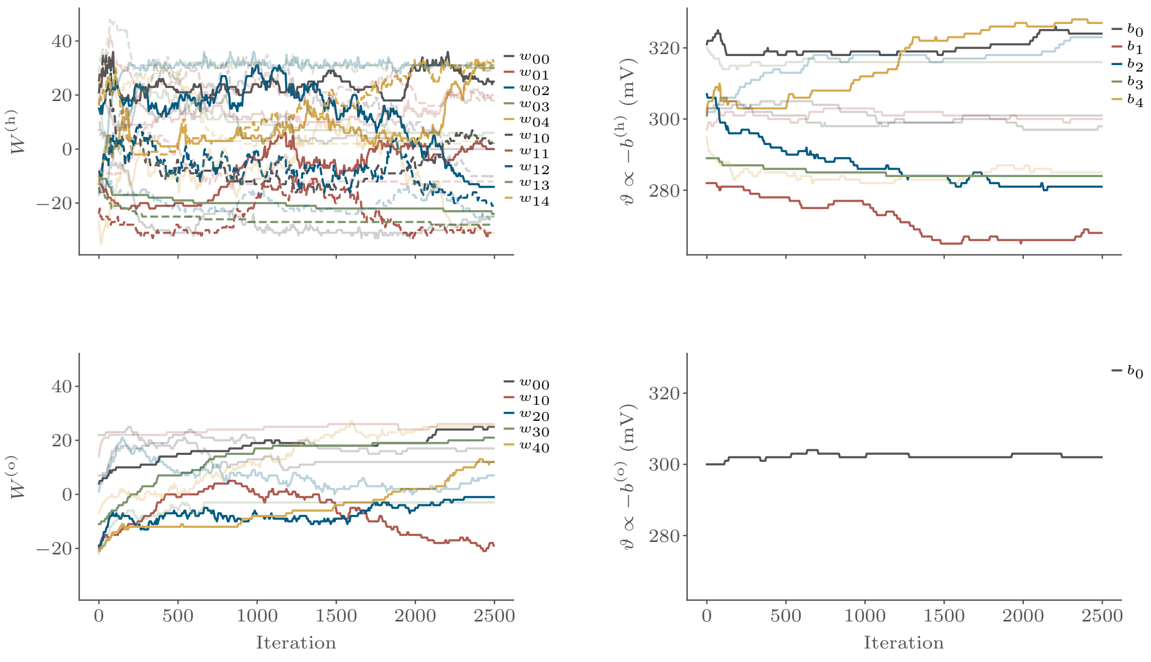


Figure A.1: Monitoring of the training process. The first and second row describes the hidden and output layer respectively. The weights are depicted in the first column and the biases in the second.

Acknowledgments

And finally, after many lines of writing I would like to express my gratitude to all the people who supported me over the last year to make this thesis possible. In particular,

Johannes for leading the Electronic Vision(s) through tough times and showing the group a promising perspective.

Professor Hausmann for agreeing to be the second advisor of the thesis.

Benjamin and Sebastian for supervision, for the selfless patience to share your extensive knowledge about neuromorphic computing and for being reachable at almost all hours of the day.

Benjamin, Christoph, Frederik, Laura, Laura, Sebastian, Sebi and Yannik for proofreading and the good feedback.

Friedemann for the insight to SuperSpike and the motivating video conference even before Corona.

Christian, Eric, Oliver, Philipp and Yannik for selflessly fixing and explaining any software related problem.

Benjamin, Sebastian, Tobias and Yannik for the entertaining time in the office.

Everybody from the Electronic Vision(s) Group for the awesome time during and after work.

Christoph, David and Klaus for distracting me from work by taking me on awesome mountain adventures.

My flatmates Johannes and Sophie for allowing me to skip the cleaning schedule in the final phase of writing.

Laura and my Family.

Statement of Originality (Erklärung)

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

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

Heidelberg, May 13, 2020 ....