Thomas Limbacher, BSc

Supervised Learning Algorithms for Spiking Neuromorphic Hardware

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn. Robert Legenstein

Institute for Theoretical Computer Science
Graz University of Technology, Austria

Graz, February 22, 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, February 22, 2017

(signature)

Abstract

Supervised learning algorithms for spiking neuromorphic hardware

Analog neuromorphic hardware allows highly accelerated and, in terms of dissipated power, efficient emulations of spiking neural networks (SNNs) compared to simulations on conventional computers. This thesis presents a supervised learning algorithm for hardware emulated feed-forward SNNs with limited resolution of their synaptic efficacies. Training is done in conjuction with an abstract software model of the network which is used to perform the parameter updates based on the recordet network ativity of the SNN. Despite inherently present noise in analog circuitry, the introduced algorithm was succesfully tested on two classification problems—on the classical XOR problem and on a subset of the MNIST dataset—where a good classification performance could be achieved. The performance could, however, be improved if the variations of the hardware would be on a smaller scale, as it has been shown in a software simulation of the SNNs.

Überwachte Lernalgorithmen für spikende neuromorphe Hardware

Analoge neuromorphe Hardware ermöglicht, im Vergleich zu Simulationen mit konventionellen Computern, eine sehr energieeffiziente und schnelle Emulation von spikenden neuronalen Netzwerken (spiking neural networks, SNNs). In der vorliegenden Arbeit wird ein überwachter Lernalgorithmus für spikende neuronale Netzwerke, welche auf neuromorpher Hardware emuliert werden, präsentiert. Das Lernen der Netzwerke wird in Verbindung mit einem abstrakten Softwaremodell des Netzwerkes ausgeführt, welches basierend auf der Aktivität im Netzwerk die Parameter Aktualisierung durchführt. Trotz inhärent vorhandenem Rauschen in analogen Schaltkreisen konnte der vorgestellte Algorithmus erfolgreich auf zwei Klassifizierungsprobleme angewendet werden – auf das klassische XOR-Problem und auf einen Teil des MNIST Datensatzes – bei welchen eine gute Performance erzielt wurde. Wie eine Softwaresimulation des spikenden neuronalen Netzwerkes gezeigt hat, könnte diese jedoch verbessert werden, wenn sich die Variabilität zwischen den einzelnen Hardware Komponenten in einem kleineren Bereich befände.

Contents

List of Figures

List of Tables

1 Introduction

Artificial neural networks (ANNs) achieve human-level performance on a wide range of classification tasks [1]. This, however, requires large models with multiple layers of computational units between their input and output layer. Training of such large networks is a highly computationally intensive task which requires large training sets and a vast amount of training time. To do so, they require power-hungry computing hardware, such as graphics processing units (GPUs). In contrary, the human brain seems to perform such tasks with ease while requiring extremely low power.

In the last years, a new type of computing hardware which shares properties with networks of neurons in the brain was developed. Similar to neurons in the brain, these so-called neuromorphic chips use discrete events—comparable to spikes—for computation and communication. Although the concept of networks which use spiking neurons is not new, in fact these networks known as spiking neural networks (SNNs) have been successfully applied to various computational tasks and have at least the same computational power as sigmoidal neural networks of the same size [2], emulating them on hardware brings several advantages over simulating them on conventional computers, particularly in terms of speed and energy consumption. The low energy consumption is due to their event based communication—they consume energy only when necessary. In contrast to the fully digital spike-based neuromorphic design of e.g., the *TrueNorth* chip of IBM [3], mixed-signal neuromorphic chips like the one integrated in the *Spikey* neuromorphic hardware use digital event-based communication while implementing neurons and synapses by analog circuits in continuous time. This allows the design of highly accelerated chips since the characteristic time constants (e.g., membrane and synapse time constants) can be chosen much smaller than typical corresponding biological values.

While artificial neural networks (ANNs) are typically trained by backpropagation, which is a supervised learning algorithm which propagates high precision error values through the layers of the network, a similar powerful learning algorithm for networks of spiking neurons is not known. A quiet big step in this direction was recently made by Esser, Merolla, Arthur, *et al.* [4]. They demonstrated, by introducing two constraints into the learning rule—binary-valued neurons with approximate derivatives and trinary-valued synapse—the successful training of neural network on the *TrueNorth* chip through the backpropagation algorithm. The algorithm was tested on several difficult benchmark data sets for classifying visual or auditory inputs where close to state-of-the-art performance could be achieved. The *TrueNort* chip is a fully digital spike-based

neuromorphic chip, which means that each parameter, activations of neurons, and their gradients are accessible through an exact software model.

Courbariaux, Bengio, and David [5] have showed—although they aimed at a different goal, that is the reduction of fixed-point multiplications in GPUs, it is worth to mention here—that deep neural networks (DNNs) can achieve nearly state-of-the-art results on several benchmark tests when trained with binary activation functions and weights. They claim that this acts as a variant of Dropout, in which instead of randomly setting a number of activations to zero, activations and weights are binarized.

In contrast to fully digital neuromorphic hardware, the neurons and synapses in analog hardware are not as precisely controllable and, in the case of the *Spikey* system, the internal state of the neurons is not accessible through measurements (to be precise, it is only accessible for one neuron at a time). This makes the exact mapping between the network emulated in hardware and its software model difficult. Moreover, neurons and synapses on analog neuromorphic hardware are subject to various types of noise which makes the mapping even more challenging.

This work demonstrates the successful training of a feed-forward spiking neural network which is emulated on analog neuromorphic hardware. The used hardware is the *Spikey* neuromorphic system. This system is a mixed-signal neuromorphic architecture with analog neuromorphic circuits and digital, event based communication. Training is done in conjunction with an abstract model of the feed-forward spiking neural network. More precisely, each spiking neuron is approximated by an artificial neuron with a binary activation function. In each training step, the network activity is recorded in hardware and is then used to perform the parameter updates in the software model via backpropagation—using high precision values. The high precision values are then quantized at 4 bit to match the low-precision synaptic weights in hardware.

This thesis is basically divided into three parts. Chapter 2 introduce the reader to spiking neuron models, models of synaptic plasticity. In Chapter 3 the utilized neuromorphic hardware system *Spikey*, which was developed within the FACETS[1] project by members of the Electronic Vision(s)[2] group at the Kirchhoff Institute for Physics in Heidelberg, is presented and the deviations of some neuron and synapse parameters, caused by variations of electronic components and imperfections in the production process, are stated and analyzed. In Chapter 4 the author's contribution to this topic is presented. In particular, a supervised learning algorithm for hardware emulated feed-forward SNNs is implemented and tested on two classification problems.

[1] FACETS: Fast Analog Computing with Emergent Transient States
[2] `http://www.kip.uni-heidelberg.de/vision/`

2

Spiking Neuron Models

In a attempt to capture fundamental aspects of neural computation in the brain, McCulloch and Pitts [6] proposed a neuron model—which is today referred to as McCulloch-Pits neuron—over seventy years ago. The first generation of neural networks is based on this rather simple neuron model. The neuron can only give two output signal: a binary "high" signal if the sum of its weighted input signals is above a certain threshold value and a binary "low" if this sum is below the threshold value. Therefore this neuron is often referred to as threshold gate. Despite their simplicity, these neurons have been successfully applied in powerful ANNs like multi-layer perceptrons. For example, every Boolean function can be computed by a multi-layer perceptron with a single hidden layer [7].

More biologically realistic neuron models use continuous functions, e.g., sigmoid and hyperbolic tangent, to compute their output signals. The functions allow for both continuous input and output signals which enables networks comprising of these neurons to approximate any analog function arbitrary well. Just like networks of the first generation they are also universal for digital computations if one applies thresholding at the output layer [8]. Typical examples of ANNs consisting of neurons of this type are feed-forward and recurrent neural networks.

First- and second-generation neuron models do not send out short pulses as real neurons do but their output signal is typically bounded between 0 and 1. These signals can be interpreted as the normalized firing frequency of a biological neuron within a certain time period, where higher firing rates correspond to higher output signals. Since real spikes can be seen as a binary event, i.e., there is a spike, or there is no spike, this so-called rate coding requires an averaging mechanism. It is known that real neurons fire at various frequencies which lie between their minimum and maximum frequency. Continuous activation functions can model these intermediate output frequencies. Hence, neurons of the second generation are more powerful and biologically realistic than neurons of the first generation [9].

Changing the strength of the connections between neurons can alter the information flow through a neural network. If the strengths of incoming signals are altered it is likely that the output signal will also change. This is a fundamental basis for learning. A simple but still powerful learning algorithm is the backpropagation algorithm. This algorithm requires that the activation function of each neuron is differentiable which again emphasizes the superiority of the second-generation neuron models.

The third generation of neuron models are even more biologically realistic. These neuron

models use individual spikes which allows the integration of spatial-temporal information in communication and computation, just like real neurons do [10]. The transmitted information of these neurons is usually encoded in the spike rate (rate coding) and/or in the timing of the spikes (pulse coding). If the spike timings are known, the average firing rate can be computed. Hence, rate coding can be considered as a special case of pulse coding. Thorpe, Delorme, and Van Rullen [11] showed that humans can analyze and classify visual input within 100 ms. It takes at least 10 synaptic steps from the retina to the temporal lobe which leaves about 10 ms processing time per neuron. They pointed out that this time window is too short to allow an averaging mechanism like rate coding and that, when speed is an issue, pulse coding is favored over rate coding.

The following sections give some information about the biological background of real neurons (Section 2.1), introduce the reader to the leaky integrate-and-fire (LIF) model (Section 2.2) and to the spike response model (SRM) (Section 2.3), and present some aspects of synaptic plasticity (Section 2.4).

2.1 Biological Background

In general, incoming action potentials from various pre-synaptic neurons alter the internal state, i.e., the membrane potential, of a post-synaptic neuron. The postsynaptic neuron sends out an action potential itself at the time instance its membrane potential exceeds a threshold value. Due to the form and nature of these action potentials, i.e., a short-lasting event in which the membrane potential of the neuron rapidly rises and falls, they are commonly termed as spikes. A spike train consists of a temporal sequence of spikes. The membrane potential of a neuron rapidly decreases immediately after the firing of an output spike. This is known as repolarization [12]. After this repolarization phase the neuron's membrane potential is held at a value lower than the resting potential for a certain amount of time, known as refractory period. This is known as hyperpolarization [12]. This makes it difficult for the neuron to emit another spike during the time period of the hyperpolarization phase.

The generated spike travels through the soma (or cell body) and traverses down a fiber called the axon. Spikes cannot just hop from one neuron to neuron to the other. They have to be processed by a quite complicated part of the neuron: the synapse [13], which is formed by the pre-synaptic end of the axon (the axon terminal), the synaptic cleft, and the first part of a dendrite. In short, when a spike arrives at the axon terminal it causes the rapid opening of calcium ion channels in the membrane of the axon which allows calcium ions to flow inward across the membrane. The increase in calcium concentration causes synaptic vesicles filled with a neurotransmitter to fuse with the axon's membrane and empty their contents into the extracellular fluid which fills the synaptic cleft. The neurotransmitter then binds to a matching receptor on the post-synaptic side of the cleft and activates it. Its activation leads to an opening

of ion-specific channels of the post-synaptic membrane through which cations and anions flow according to their concentration gradient [12]. This process induces a postsynaptic potential (PSP) which can either be positive and called excitatory postsynaptic potential (EPSP) or negative and called inhibitory postsynaptic potential (IPSP). A post-synaptic neuron usually receives spikes from multiple pre-synaptic neurons, each with multiple spikes, which induces multiple PSPs over time. These PSPs are integrated over time which results in the overall post-synaptic potential of the neuron.

In contrast to action potentials, which are all quite similar, postsynaptic potentials differ in their amplitude. This is caused by various biological processes including ion channel mechanisms, in short, the synaptic efficacy [12], [14].

2.2 Leaky Integrate-and-Fire Model

The LIF neuron model [14] is probably one of the simplest spiking neuron models, but still quite popular due to the ease with which it can be simulated and analyzed. In its simplest form it can be modeled by a circuit consisting of a capacitor C in parallel with a resistor R which is driven by some current $I(t)$. The current $I(t)$ can be written as sum of the current which passes through the resistor R and the current which charges the capacitor C, therefore one can write $I(t) = I_R(t) + I_C(t)$. The current $I_R(t)$ is given by Ohm's law as $I_R(t) = \frac{V(t)}{R}$ where $V(t)$ is the voltage across the resistor. The second term $I_C(t)$ can be defined as the rate of flow of charge $Q(t)$, i.e.,

$$I_C(t) = \frac{dQ(t)}{dt} = C\frac{dV(t)}{dt}. \tag{2.1}$$

Therefore one gets with $\tau_\mathrm{m} = RC$ as the membrane time constant of the neuron the standard form of a "leaky integrator" that is

$$\tau_\mathrm{m}\frac{dV(t)}{dt} = -V(t) + RI(t), \tag{2.2}$$

where $V(t)$ is referred to as the membrane potential.

In this neuron model the form of the action potential is not described explicitly. Instead, when the membrane potential $V(t)$ reaches a certain threshold V_th, it is instantaneously set to the so-called reset potential $V_\mathrm{reset} < V_\mathrm{th}$. For $t > t^{(f)}$, where $t^{(f)}\colon V(t^{(f)}) = V_\mathrm{th}$ is the firing time, the dynamic is again given by Eg. 2.2 until the next threshold crossing occurs. The combination of leaky integration and reset defines the basic LIF model [15].

In its general form, an absolute refractory period t_refrac is considered. The neuron's dynamics given by Eg. 2.2 are interrupted at $t = t^{(f)}$, i.e., the time at which the neuron's membrane potential reaches the threshold V_th, and restarted after t_refrac at $t^{(f)} + t_\mathrm{refrac}$ with the new initial condition V_reset.

The time course of the membrane potential $V(t)$ as a function of any time varying input current $I(t)$ can be expressed for $t > \hat{t} + t_{\text{refrac}}$ by

$$V(t) = V_{\text{reset}} \exp\left(-\frac{t - \hat{t} - t_{\text{refrac}}}{\tau_{\text{m}}}\right) + \frac{1}{C_{\text{m}}} \int_0^{t-\hat{t}-t_{\text{refrac}}} \exp\left(-\frac{s}{\tau_{\text{m}}}\right) I(t-s)\, ds, \tag{2.3}$$

where $\hat{t}$ is the time of the last output spike. The function is valid until $V(t)$ crosses the threshold. At this point the membrane potential is reset to V_{reset} and the integration restarts after the refractory period t_{refrac}.

Assuming that the input current $I(t)$ of a post-synaptic neuron i is a sum of post-synaptic currents caused by pre-synaptic spikes, that is

$$I(t) = \sum_j w_{i,j} \sum_f \alpha\left(t - t_j^{(f)}\right), \tag{2.4}$$

where $\alpha\left(t - t_j^{(f)}\right)$ is the input current pulse and $w_{i,j}$ is the synaptic efficacy from neuron j to the post-synaptic neuron i, Eq. 2.3 can be written as

$$V_i(t) = V_{\text{reset}} \exp\left(-\frac{t - \hat{t}_i - t_{\text{refrac}}}{\tau_{\text{m}}}\right) + \sum_j w_{i,j} \sum_f \frac{1}{C_{\text{m}}} \int_0^{t-\hat{t}_i-t_{\text{refrac}}} \exp\left(-\frac{s}{\tau_{\text{m}}}\right) \alpha\left(t - t_j^{(f)} - s\right) ds. \tag{2.5}$$

Typical choices for α include the alpha-shaped function, i.e.,

$$\alpha(t) = \frac{t}{\tau} \exp\left(\frac{-t}{\tau}\right) H(t), \tag{2.6}$$

found in e.g., Shelley, McLaughlin, Shapley, *et al.* [16] and Kumar, Schrader, Aertsen, *et al.* [17] or the bi-exponential function, i.e.,

$$\alpha(t) = \frac{\tau_2}{\tau_2 - \tau_1} \left[\exp\left(\frac{-t}{\tau_1}\right) - \exp\left(\frac{-t}{\tau_2}\right)\right] H(t), \tag{2.7}$$

where τ, τ_1, and τ_2 are the time constants of the synapse and $H(t)$ is the Heaviside step function.

Stimulation by a Constant Input Current

Assuming a constant input current $I(t) = I_0$ and $\hat{t}_i = 0$ as time of the last spike of the neuron the time t_2 of the next spike can be derived from Eg. 2.3 as follows:

$$\begin{aligned} V_i(t) &= V_{\text{reset}} \exp\left(-\frac{t - t_{\text{refrac}}}{\tau_{\text{m}}}\right) + \frac{I_0}{C_{\text{m}}} \int_0^{t-t_{\text{refrac}}} \exp\left(-\frac{s}{\tau_{\text{m}}}\right) ds \\ &= V_{\text{reset}} \exp\left(-\frac{t - t_{\text{refrac}}}{\tau_{\text{m}}}\right) + \frac{I_0}{C_{\text{m}}} \left[-\tau_{\text{m}} \exp\left(-\frac{s}{\tau_{\text{m}}}\right)\right] \Bigg|_0^{t-t_{\text{refrac}}} \\ &= \exp\left(-\frac{t - t_{\text{refrac}}}{\tau_{\text{m}}}\right) \left(V_{\text{reset}} - \frac{I_0 \tau_{\text{m}}}{C_{\text{m}}}\right) + \frac{I_0 \tau_{\text{m}}}{C_{\text{m}}}. \end{aligned}$$

The next spike of the neuron will occur at $t = t_2$, when the membrane potential $V_i(t)$ reaches the threshold potential V_{th}, hence

$$\begin{aligned} V_{\text{th}} &= \exp\left(-\frac{t_2 - t_{\text{refrac}}}{\tau_{\text{m}}}\right) \left(V_{\text{reset}} - \frac{I_0 \tau_{\text{m}}}{C_{\text{m}}}\right) + \frac{I_0 \tau_{\text{m}}}{C_{\text{m}}} \\ -\frac{t_2 - t_{\text{refrac}}}{\tau_{\text{m}}} &= \ln\left(\frac{C_{\text{m}} V_{\text{th}} - I_0 \tau_{\text{m}}}{C_{\text{m}} V_{\text{reset}} - I_0 \tau_{\text{m}}}\right) \end{aligned}$$

Solving the last equation for t_2, the time of the next spike can be expressed by

$$t_2 = -\tau_{\text{m}} \ln\left(\frac{C_{\text{m}} V_{\text{th}} - I_0 \tau_{\text{m}}}{C_{\text{m}} V_{\text{reset}} - I_0 \tau_{\text{m}}}\right) + t_{\text{refrac}}. \tag{2.8}$$

Due to the constant input current I_0, the integrate-and-fire neuron spikes periodically with a period $T = t_2$ given by Eq. 2.8.

2.3 Spike Response Model

The SRM is a generalization of the leaky integrate-and-fire model [14]. The evolution of its membrane potential depends on the time since the last output spike and can be expressed as an integral over the past. Let $\hat{t}_i$ be the time since the last spike of neuron i, then the evolution of its membrane potential $V_i(t)$ is given by

$$V_i(t) = \eta\left(t - \hat{t}_i\right) + \sum_j w_{i,j} \sum_f \epsilon_{i,j}\left(t - \hat{t}_i, t - t_j^{(f)}\right) + \int_0^\infty \kappa\left(t - \hat{t}_i, s\right) I_{\text{ext}}(t - s)\, ds, \tag{2.9}$$

where $t_j^{(f)} \in F_j$ is the spike time of pre-synaptic neuron j and $w_{i,j}$ is the synaptic efficiency from neuron j to the post-synaptic neuron i. The spike times (or spike train) of a pre-synaptic neuron j can be formally written as

$$F_j = \left\{t_j^{(1)}, t_j^{(1)}, \ldots, t_j^{(n)}\right\}, \tag{2.10}$$

with $t_j^{(n-1)} < t_j^{(n)} < t$. The two sums in Eq. 2.9 run over all pre-synaptic neurons and over all firing times. One has to note that each term depends on the time of the last output spike of neuron i, that is $\max(F_i)$. The reset kernel η models the time course of the action potential and the after-potential. The response kernel ϵ models the postsynaptic potential (PSP) and the κ kernel models the linear response of the membrane potential to an input current $I_{\text{ext}}(t)$.

Contrary to the threshold value V_{th} in the LIF model, the threshold in the SRM is not constant and may also be a function of the last output spike. The neuron fires whenever the membrane potential $V_i(t)$ reaches the dynamic threshold $V_{\text{th}}(t - \hat{t}_i)$ from below. The points in time at which the neuron i spikes can therefore be defined as the set

$$F_i = \left\{ t \;\middle|\; V_i(t) = V_{\text{th}}(t - \hat{t}_i), \frac{dV_i(t)}{dt} > 0 \right\}. \tag{2.11}$$

While the leaky integrate-and-fire model is due to its simplicity more suitable for simulations (in particular in the context of large networks), the spike response model has its advantages in its level of biological realism. However, several characteristics of real neurons are not considered in this model. As an example, the opening of many ion channels leads to a smaller membrane resistance which results in smaller membrane time constants, see [18].

2.4 Models of Synaptic Plasticity

Synaptic plasticity, i.e., the synapse's ability to change its strength, is a fundamental property for learning and information processing. Many of today's models of synaptic plasticity, which describe how connections between neurons should be modified, are based on Donald O. Hebb's postulate [19]:

> "When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

This theory is often summarized by the famous phrase: "Cells that fire together, wire together." [20]. Which is, however, quite loosely formulated since Hebb [19] emphasized that a cell needs to—take part—in firing another cell. Which is, in terms of causality, not possible if they fire at the exact same time. This temporal aspect was nicely shown by experiments of Bi and Poo [21]. They found that the change in direction and in the absolute value of the synaptic strength depends crucial (on a millisecond time-scale) on the relative timing of pre- and post-synaptic spikes. The change in synaptic efficacy only takes place if the pre-synaptic spike at time $t_j^{(f)}$ is sufficiently close to the post-synaptic spike at time $t_i^{(f)}$. The synaptic connection is strengthened if the pre-synaptic spike precedes the post-synaptic one, i.e., if $t_i^{(f)} - t_j^{(f)} > 0$. On the other hand, if $t_i^{(f)} - t_j^{(f)} < 0$ the synaptic efficacy between pre- and post-synaptic neurons is weakened.

Spike-Timing Dependent Plasticity

This spike-timing dependent plasticity (STDP) [22], which specifies the change $W(\Delta t)$ of the synaptic weight, is commonly modeled by a so-called learning window of the form

$$W(\Delta t) = \begin{cases} A_+ \exp\left(\frac{-\Delta t}{\tau_+}\right) & \text{if } \Delta t > 0, \\ \\ -A_- \exp\left(\frac{-\Delta t}{\tau_-}\right) & \text{if } \Delta t < 0, \end{cases} \tag{2.12}$$

where the positive constants A_+ and A_- scale the strength of potentiation and depression, respectively. The time constants τ_+ and τ_- define the width of the positive and negative part of the learning window. The time difference between pre- and post-synaptic spike is given by $\Delta t = t_i^{(f)} - t_j^{(f)}$. The resulting change in the synaptic weight $w_{i,j}$ between a pre-synaptic neuron j and a post-synaptic neuron i for a certain pre-synaptic spike train F_j and post-synaptic spike train F_i is usually modeled by the application of the learning window given in Eg. 2.12 to all spike pairings, that is

$$\left[\frac{d}{dt} w_{i,j}(t)\right]_{\text{STDP}} = \int_0^\infty W(\tau) S_j(t) S_i(t-\tau)\, d\tau + \int_0^\infty W(-\tau) S_j(t-\tau) S_i(t)\, d\tau, \tag{2.13}$$

where $S_i(t)$ and $S_j(t)$ are given by

$$S_i(t) = \sum_{t_i^{(f)} \in F_i} \delta\left(t - t_i^{(f)}\right) \quad \text{and} \quad S_j(t) = \sum_{t_j^{(f)} \in F_j} \delta\left(t - t_j^{(f)}\right), \tag{2.14}$$

respectively [23].

Short-Term Plasticity

Changes in synaptic strengths induced by STDP are usually permanent. Another plasticity mechanism which acts on the timescale of hundred milliseconds is known as short-term plasticity (STP) [24], [25]. It is distinguished between short-term depression and short-term facilitation. In the latter, the synaptic strength increases which each pre-synaptic spike and in the former it decreases. In the absence of pre-synaptic activity the synaptic efficacy will quickly return to its original value.

A simplified explanation of the underlying biological mechanism can be given as follows: Each spike which has to be processed by the synapse leads to the release of neurotransmitters at the axon terminal (as briefly described in Section 2.1). If the pre-synaptic neuron fires at a high rate a depletion of these neurotransmitters can occur which weakens the synapse. On the other hand, short-term facilitation is a result of calcium influx into the axon terminal after spike generation, which increases the release probability of neurotransmitters [26]. Which can again

lead to short-term depression.

Since the whole physiological process underlying STP is rather complex, some simplified models have been created, e.g., see Markram, Wang, and Tsodyks [27], Abbott, Varela, Sen, *et al.* [28], and Tsodyks, Pawelzik, and Markram [29].

The model described in Tsodyks, Pawelzik, and Markram [29] uses a normalized variable $0 \leq R(t) \leq 1$ which denotes the fraction of remaining resources after neurotransmitter depletion. This models the effect of short-term depression. The short-term facilitation effect is modeled by a utilization parameter $u(t)$ which represents the fraction of available resources ready for use (release probability of the neurotransmitters). After each pre-synaptic spike at time $t_j^{(f)}$, $u(t)$ increases with $f[1 - u(t)]$ due to the calcium influx into the axon terminal. A fraction of the available resources is then consumed according to $u(t)R(t)$ due to the pre-synaptic spike. In the absence of spikes, $u(t)$ follows an exponential decay to the baseline release probability U with time constant F and $R(t)$ recovers with a time constant D back to one. This process can be modeled by the differential equations

$$\frac{dR(t)}{dt} = \frac{1 - R(t)}{D} - u(t^-)R(t^-)\delta(t - t_j^{(f)}) \tag{2.15}$$

and

$$\frac{du(t)}{dt} = \frac{U - u(t)}{F} + f[1 - u(t^-)]\delta(t - t_j^{(f)}). \tag{2.16}$$

The notation t^- emphasize that these equations should be evaluated in the limit approaching the spike time $t_j^{(f)}$ from below.

One can obtain depressing synapse dynamics, facilitating synapse dynamics, or a combination of both by varying the model parameters $\boldsymbol{\theta} = \begin{bmatrix} D & F & U & f \end{bmatrix}^{\mathrm{T}}$.

Stating Eq. 2.15 and Eq. 2.16 as recurrence relation gives the advantage of faster numerical simulations. This can be done by integrating the above equations between spikes n and $n + 1$ which yields [30]:

$$R_{n+1} = 1 - [1 - R_n(1 - u_n)] \exp\left(-\frac{\Delta t_n}{D}\right) \tag{2.17}$$

$$u_{n+1} = U + [u_n + f(1 - u_n) - U \exp\left(-\frac{\Delta t_n}{F}\right), \tag{2.18}$$

where Δt_n is the inter-spike interval.

As a conclusion on can say that STP can alter neural information transmission by modifying the synaptic efficacy based on the history of pre-synaptic spikes. If the effect of short-term depression is stronger in a certain synapse, then it favors somehow information which is transferred at low firing rates. In contrary, if short-term facilitation is more pronounced, the synapse works

best with information transferred at high rates.

3

The Spikey Neuromorphic System

Spikey is a highly configurable neuromorphic hardware system developed within the FACETS project by members of the Electronic Vision(s) group at the Kirchhoff Institute for Physics in Heidelberg, Germany. The core of this system is a mixed-signal chip with analog implementations of neurons and synapses and digital transmission of action potentials. The analog approach allows highly accelerated and, in terms of dissipated power, efficient emulations of networks compared to simulations on conventional computers. Its configurability enables the realization of almost arbitrary network topologies with a wide verity of neuron and synapse parameters. The associated software provides researchers and non-expert users with an easy-to-use tool for building and emulating neuronal network models.

The following sections give an overview of the hardware system (Section 3.1) and present the incorporated neuron model (Section 3.2) and synapse model (Section 3.3). Furthermore, details on model parameters, their possible configurations, network topologies and their restrictions are given in Section 3.4. Noise is inherently present in analog circuitry and since this can be a major issue under certain circumstances, a closer look at its range and effects is taken in Section 3.5. Finally, the neuronal modeling language and the low-level software interface is briefly described in Section 3.6.

This chapter aims to summarize the most important properties of the *Spikey* system and does not claim to be comprehensive nor does it contain all information. For further information on the neuron and synapse model see Pfeil, Scherzer, Schemmel, *et al.* [31]. Detailed information about the analog implementation of the neuron model, STP and STDP is given in Indiveri, Linares-Barranco, Hamilton, *et al.* [32], Schemmel, Bruderle, Meier, *et al.* [33] and Schemmel, Grubl, Meier, *et al.* [34], respectively. For the documentation of the digital part of the chip see Grübl [35].

3.1 System Overview

The *Spikey* chip is fabricated in a CMOS[3] VLSI[4] manufacturing process. While neuron and synapse circuitries are mostly implemented in analog technique, communication to the host

[3] Complementary metal-oxide-semiconductor

[4] Very-large-scale integration (VLSI) is the process of creating an integrated circuit by combining thousands of transistors into a single chip.

computer and the transmission of action potentials is established by digital circuits. Since the time constants on the chip are approximately 10^4 times smaller than in biology, emulations of networks on *Spikey* are accelerated.

The total number of neurons on the chip is 384 with a maximum number of 256 synapses each. The neurons are distributed evenly over two neuron blocks. Each line of synapses in these blocks, i.e., 192 synapses, is driven by one of the 256 so-called synapse drivers. The 192 synapses in each line share, for technical reasons, the same configuration. As a consequence, all synapses in one line can be either excitatory or inhibitory. Each synapse driver can be configured to receive input from either external spike sources (e.g., generated on the host computer), from on-chip neurons in the same block, or from on-chip neurons in the opposite block. A synaptic weight with a 4 bit resolution can be individually assigned to each of the $256 \cdot 192 = 49152$ synapses in one block. The spike output of every neuron can be fed to either one specific synapse driver in the same block, to the corresponding synapse driver in the adjacent block, to a digital recording unit, or to every possible combination of those three targets. While spike times can be recorded simultaneously from all neurons during network emulations, the recording of membrane potentials is limited to a single but arbitrary neuron. For a schematic diagram of the chip see Fig. 3.1.

The hardware implementations of neurons and synapses are inspired by the conductance-based LIF neuron model using synapses with an alpha-shaped conductance course. A description of the implemented neuron model is given in Section 3.2 and Section 3.3 provides some information on the incorporated synapse model.

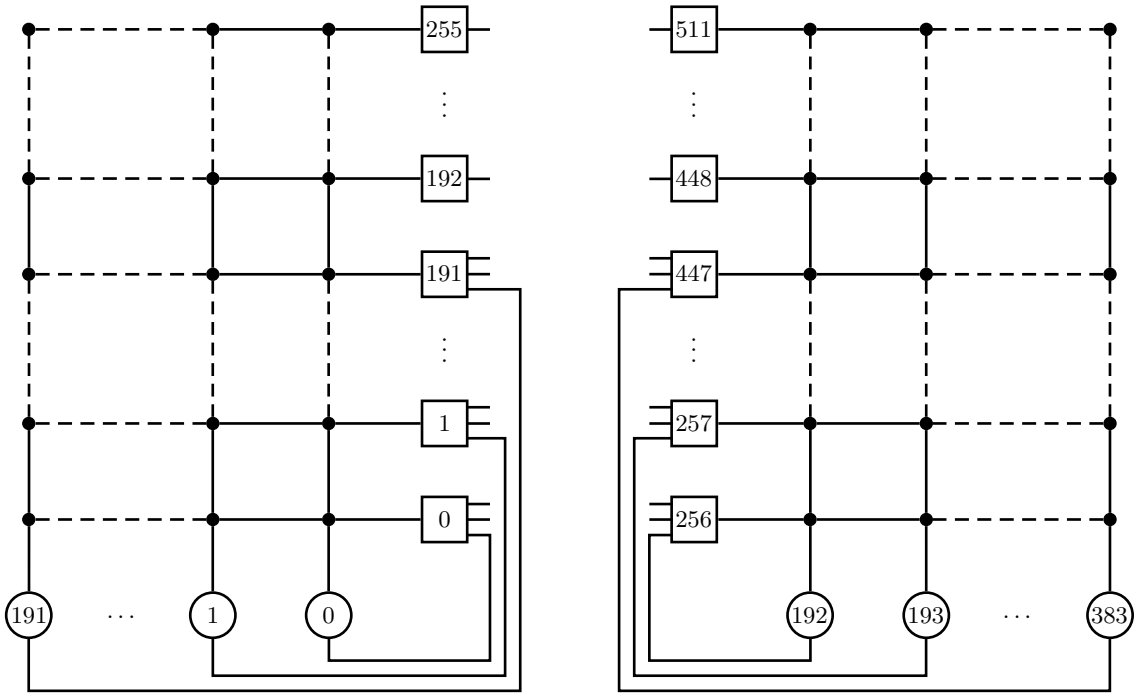

Figure 3.1: Numbering of neurons (*circles*) and synapse drivers (*squares*). The total number of 384 neurons are split into two blocks of 192 neurons with 256 synapses each (*filled circles*). Each line of synapses in these blocks, i.e., 192 synapses, is driven by a synapse driver. The upper 64 synapse drivers are reserved for external inputs only. All synapses in one line can be configured to be either excitatory or inhibitory. The synaptic weight can be set individually for each of the 98304 synapses with a 4 bit resolution.

3.2 Neuron Model

The evolution of the membrane potential $V_{\mathrm{m},i}(t)$ of a *Spikey* chip neuron i is given by the differential equation

$$-C_{\mathrm{m}}\frac{dV_{\mathrm{m},i}(t)}{dt} = g_{\mathrm{L}}(V_{\mathrm{m},i}(t)-E_{\mathrm{L}})+\sum_k g_k^{\mathrm{exc}}(t)\,(V_{\mathrm{m},i}(t)-E_{\mathrm{exc}})+\sum_l g_l^{\mathrm{inh}}(t)\,(V_{\mathrm{m},i}(t)-E_{\mathrm{inh}})\,, \quad (3.1)$$

where the indices k and l run over all excitatory and inhibitory synapses, respectively. When $V_{\mathrm{m},i}(t)$ reaches the threshold potential V_{th} a spike is generated and $V_{\mathrm{m},i}(t)$ is held to the reset potential V_{reset} for a refractory period of t_{refrac}. The total current flow across the membrane is determined by the membrane capacitance C_{m}, the leak reversal potential E_{L}, excitatory E_{exc} and inhibitory E_{inh} reversal potentials, the leak conductance g_{L}, excitatory $g_k^{\mathrm{exc}}(t)$ and inhibitory $g_l^{\mathrm{inh}}(t)$ synaptic conductances. Post-synaptic conductances $g_k(t)$ and $g_l(t)$ are modified by the occurrence of excitatory or inhibitory input events from one of the pre-synaptic neurons at time $t_j^{(f)}$ by

$$g^{\mathrm{exc|inh}}(t) = \tilde{w}_{i,j}\sum_f \epsilon\Big(t-t_j^{(f)}\Big)\,, \quad (3.2)$$

where $t_j^{(f)}$ is the spike time of pre-synaptic neuron j and $\tilde{w}_{i,j}$ is the synaptic efficacy from neuron j to a post-synaptic neuron i. The sum runs over all spike times $t_j^{(f)} < t$ of neuron j. The ϵ-kernel defines the shape of the post-synaptic conductance course and is described in Section 3.3.

If no spikes arrive $V_{\mathrm{m},i}(t)$ will converge towards E_L. The time constant of this exponential convergence is determined by C_m and g_L $\left(\tau_\mathrm{m} = \frac{C_\mathrm{m}}{g_\mathrm{L}}\right)$. For information on the analog implementation of this neuron model see Indiveri, Linares-Barranco, Hamilton, *et al.* [32].

3.3 Synapse Model

The *Spikey* chip implements conductance-based synapses. Due to their voltage-dependency they offer a more realistic impact on their post-synaptic membrane potentials as e.g., purely current-based approaches [36]. The synaptic conductance course implemented in the hardware is very similar to a alpha-shaped function, i.e.,

$$\epsilon(s) = \frac{s}{\tau_{\mathrm{exc|inh}}} \exp\left(\frac{-s}{\tau_{\mathrm{exc|inh}}}\right) H(s), \tag{3.3}$$

with $s = t - t_j^{(f)}$ where $t_j^{(f)}$ is the spike time of the pre-synaptic neuron j. The synaptic time constant is denoted by $\tau_{\mathrm{exc|inh}}$ and $H(\cdot)$ indicates the Heaviside step function.

Besides the alpha-shaped synapse dynamics, a quantal increase followed by an exponential decay is also supported by the hardware's synapse drivers. This, however, requires a proper configuration of the driver. For a detailed description regarding this topic see Brüderle [36].

3.3.1 Short-Term Plasticity (STP)

It has been shown that the synaptic efficiency changes with pre-synaptic activity on the timescale of hundred milliseconds. This mechanism is known as short term depression and short term facilitation. The hardware implementation of STP is described in Schemmel, Bruderle, Meier, *et al.* [33] and follows the ideas developed in Tsodyks and Markram [37] and Markram, Wang, and Tsodyks [27].

In the case of depressing STP the synaptic weight decreases with each pre-synaptic spike and in the case of facilitating STP it increases. The weight is recovered in the absence of pre-synaptic spikes within the recovery period t_rec. Each synapse driver of the *Spikey* chip supports both modes of STP. However, they do not support both modes simultaneously as allowed by the original model.

3.3.2 Spike-Timing Dependent Plasticity (STDP)

The *Spikey* chip offers besides STP another synaptic plasticity mechanism, namely STDP. While the change in synaptic strength induced by STP is not permanent (it recovers in the absence of pre-synaptic spikes), STDP leads to a persistent change in synaptic efficiency. The implemented STDP model is based on Song, Miller, and Abbott [22] and Bi and Poo [21] and is described in Schemmel, Grubl, Meier, *et al.* [34].

In short, the STDP process adjusts the synaptic strength on the relative timing of a particular neuron's pre- and post-synaptic action potentials, i.e., spikes. Causal spike pairs ($t_{\text{pre}} < t_{\text{post}}$) strengthen the synapse and acausal spike pairs ($t_{\text{pre}} > t_{\text{post}}$) weaken it. Pre- and post-synaptic spike times are denoted by t_{pre} and t_{post}, respectively. Since the *Spikey* chip only offers discrete changes in synaptic strengths (i.e., weights) and since their resolution is relatively low, induced weight changes of multiple causal and acausal spike pairs are accumulated until a certain threshold is exceeded. For further information on how the weight updates are performed see Schemmel, Grubl, Meier, *et al.* [34].

3.4 Configurability

One advantage of the *Spikey* chip is its high configurability. Most of the model parameters are adjustable for each individual neuron or synapse. Due to a trade-off made in the size of the current-voltage conversion circuitry, which was necessary in the design of the chip (see Pfeil, Grübl, Jeltsch, *et al.* [38]), the leak reversal potential E_{L}, the inhibitory reversal potential E_{inh}, the threshold voltage V_{th}, and the reset potential V_{reset} are shared between all even/odd-indexed neurons in a neuron block. The capacity of the membrane is realized with a physical capacitor and hence it it fixed an identical for all neurons. However, since the leak conductance g_{L} can be individually adjusted for each neuron and since the membrane time constant τ_{m} is given by $\tau_{\text{m}} = \frac{C_{\text{m}}}{g_{\text{L}}}$, it follows that τ_{m} is also individually adjustable. For a list of neuron and synapse parameters, their default values, and their possible range of values see Table 3.1.

In addition to the possibility of controlling neuron and synapse parameters, the *Spikey* chip offers an almost arbitrary configurability of the network topology. The maximum fan-in is 256 synapses per neuron, which can be composed of up to 192 synapses from on chip neurons and up to 256 synapses from external spike sources. One has to consider that connections made from external inputs reduce the number of possible connections from on chip neurons. Due to the fact that the number of neurons exceeds the number of possible inputs per neuron, an all-to-all connection is not possible. Furthermore, all synapses driven by the same synapse driver are either excitatory or inhibitory. The synaptic weight of each synapse can be configured individually with a 4 bit resolution.

Most of the restrictions on connectivity and synapse configurations are considered by the provided software interface in the process of mapping the neurons defined in the modeling language

to hardware neurons (see Section 3.6). However, it is , for example, possible—without receiving any error message—to make an excitatory and an inhibitory connection from one external spike source to one or more arbitrary on chip neurons. This is, however, not supported by the chip and results in the fact that both synaptic connections are tacitly made excitatory. Furthermore it was found that results differ depending on how one deals with synaptic connections with a weight of zero. The decision of setting these types of connections either as excitatory, inhibitory, or not making a connection at all is left to the user and is not properly handled by the software.

Table 3.1: List of neuron and synapse parameters. Each with the corresponding model parameter names, their default values, and their possible range of values. Parameters denoted with i are individually controllable for each of the 384 neurons (subscript n), 256 synapse line drivers (subscript l), and 98305 synapses (subscript s). Parameters denoted with s are shared between all even/odd neurons or synapse line drivers, respectively. The capacitance of the neuron's membrane is realized with a physical capacitor and is fixed and identical for all neurons. Electronic parameters that have no direct translation to model parameters and parameters related to STP and STDP are not stated. Adapted from Pfeil, Grübl, Jeltsch, *et al.* [38].

Scope	Name	Type	Description	Value	
				Default	Range
Neurons	g_L	$\mathrm{i_n}$	Leak conductance	20 nS	6 nS to 64 nS
	C_m	$\mathrm{g_n}$	Capacity of the membrane	0.2 nF	n/a
	t_refrac	$\mathrm{i_n}$	Duration of the refractory period	1 ms	1 ms to 10 ms
	E_L	$\mathrm{s_n}$	Leak reversal potential	−75 mV	−55 mV to −80 mV
	E_inh	$\mathrm{s_n}$	Inhibitory reversal potential	−80 mV	−55 mV to −80 mV
	E_exc	$\mathrm{g_n}$	Excitatory reversal potential	0 mV	n/a
	V_th	$\mathrm{s_n}$	Firing threshold voltage	−55 mV	−55 mV to −80 mV
	V_reset	$\mathrm{s_n}$	Reset potential of the membrane	−80 mV	−55 mV to −80 mV
Synapse line drivers	τ_inh	$\mathrm{i_l}$	Rise time of the inhibitory synaptic alpha function	5 ms	n/a
	τ_exc	$\mathrm{i_l}$	Rise time of the excitatory synaptic alpha function	5 ms	n/a
Synapses	$\tilde{w}$	$\mathrm{i_s}$	4 bit weight of each individual inhibitory synapse	n/a	0 nS to 60 nS
			4 bit weight of each individual excitatory synapse	n/a	0 nS to 15 nS

3.5 Fixed-pattern and Temporal Noise

Emulations on hardware are, in contrast to software simulations, subject to noise. A significant amount of the overall noise of these systems is introduced by imperfections in the production process. The manufacturing process of CMOS VLSI devices does not allow the production of perfect copies of electronic components (e.g., transistors, resistors, capacitors) and therefore introduces unavoidable variations in the electrical characteristics of these components [39]. These variations are expected to be Gaussian distributed and they stay unchanged once a device is produced. Noise introduced by imperfections in the production process is, due to its static nature,

sometimes referred as fixed-pattern noise and it can be reduced by calibration. Furthermore, other, not neglectable sources of noise are introduced by temporally changing electronic variables due to temperature fluctuation and differences in chip utilization, and electronic noise. These types of noise are hardly predictable and lead to different results in consecutive emulations of the same network.

In the following two sections the effect of static noise on the efficiency of synapse drivers and electronic noise which appears on the membrane potential of the neurons is analyzed and explained in more detail. For a complete list of noise sources and their effects, including variation on the resting potential, firing threshold, digital crosstalk, spontaneous ghost events, etc., see Brüderle [36].

3.5.1 Variations of the Synapse Drivers' Efficacies

The strengths of synapse drivers of the *Spikey* chip vary strongly from synapse driver to synapse driver despite identical settings. This is caused by the variations of electronic components and imperfections in the production process which result in different PSP amplitudes and synaptic time constants [36]. This is to be expected and could, in principle, be compensated—at least to a satisfying degree—by calibration routines. In addition to this, there is also an activity-dependent effect which makes the efficiency of the synapse driver unpredictable. This only influences excitatory drivers and is caused by voltage drops of the excitatory reversal potential E_{exc}. The amount of voltage loss depends on the number of synapses connecting this potential to leaky membranes. The time-dependence of the synapse drivers' efficiency is analyzed and explained in a more detailed manner in Brüderle [36].

An analysis of spatially-dependent variations of PSP amplitudes and time constants for various synapse drivers in combination with different neurons is shown in the following. The analysis is conducted by calculating the PSP-integral based on recordings of the membrane trace. This analysis gives, however, no evidence on whether the variations are caused by varying PSP amplitudes or variations of the time constants. But since the overall synaptic strengths, or rather their variations, are of greater interest throughout this thesis, an analysis of their cause was omitted. The analysis is performed as follows: All considered synapse drivers excite each considered neuron with one PSP of the same weight (the weight is set to the hardware weight factor of 15, i.e., conductances of the excitatory and inhibitory connections are set to 7 nS and 60 nS, respectively). In detail, all combinations of synapse drivers and neurons are simulated and analyzed one by one in order to exclude the activity-dependent effects which were mentioned above. The onset of the PSP is at 50 ms in each simulation with a total simulation time of 160 ms (biological time). The first 480 samples of the recorded membrane trace, i.e., the samples of the membrane potential at rest before the onset of the PSP, are used to estimate the leak reversal potential E_{L} individually for each neuron. The estimates of E_{L} are then used to correct the recordings of the membrane traces such that they have a mean of zero (when excluding the

PSP). Based on the resulting data, the PSP-integrals are estimated by means of the trapezoidal rule with a uniform step size of one sample, i.e., a step size of ≈ 0.1 ms. The PSP-integrals are average over 10 independent hardware runs for all combinations of synapse drivers and neurons in order to reduce the influence of statistical errors.

The absolute values of the PSP-integrals for 15 excitatory and 15 inhibitory drivers in combination with 15 different hardware neurons are shown in Fig. 3.2A and Fig. 3.2B, respectively. The overall mean of the PSP-integrals of excitatory connections is 142.17 mV ms with a standard deviation of 52.75 mV ms and 38.36 mV ms with a standard deviation of 28.95 mV ms of inhibitory connections. The standard deviation of the integrals over the 10 hardware runs, which gives an estimate for the trial-to-trial variability of the synapse drivers' efficiency, is shown in Fig. 3.2C and Fig. 3.2D. The mean standard deviation of excitatory drivers and inhibitory drivers is 16.80 mV ms and 3.59 mV ms, respectively. While the trial-to-trial variability for both configurations of the synapse drivers, i.e., excitatory and inhibitory, is relatively low and similar, the variations between different synapse drivers are in general higher and furthermore the variations of synapse drivers in inhibitory configuration are higher than those of excitatory synapse drivers.

In addition to the PSP-integrals the absolute values of the PSP heights and their variations across different drivers and neurons and across different hardware runs are analyzed. The overall mean of the PSP heights of excitatory connections is 8.95 mV with a standard deviation of 3.9 mV and 2.49 mV with a standard deviation of 1.63 mV of inhibitory connections. The mean standard deviation of the PSP heights over the 10 hardware runs is 0.88 mV for excitatory drivers and 0.26 mV for drivers in inhibitory configuration.

Figures 3.3A and 3.3B show the membrane traces of one neuron receiving Poisson stimuli over synapses driven by different synapse drivers. The neurons receive an excitatory Poisson stimulus from 0.1 s to 0.9 s, an inhibitory Poisson stimulus from 1.1 s to 1.9 s, and from 2.1 s to 2.9 s they receive simultaneously an excitatory and inhibitory Poisson stimulus. The mean firing rate of the stimulus in each interval is 20 Hz. The synaptic conductances of the excitatory and inhibitory connections are set to 7 nS and 60 nS, respectively. For comparison, a NEST simulation with the same configuration is also shown. These figures nicely illustrate the effect of the variability of the strengths of different synapse drivers. While both synapse drivers (excitatory and inhibitory) utilized to generate the results shown in Fig. 3.3A are relatively weak, the results gathered by the use of a different pair of synapse drivers do quite well match the results of the software simulation (see Fig. 3.3B). However, there is still a noticeable discrepancy between the membrane trace recorded from the hardware run to that of the software simulation. This is especially true for synapse drivers which are configured to be inhibitory.

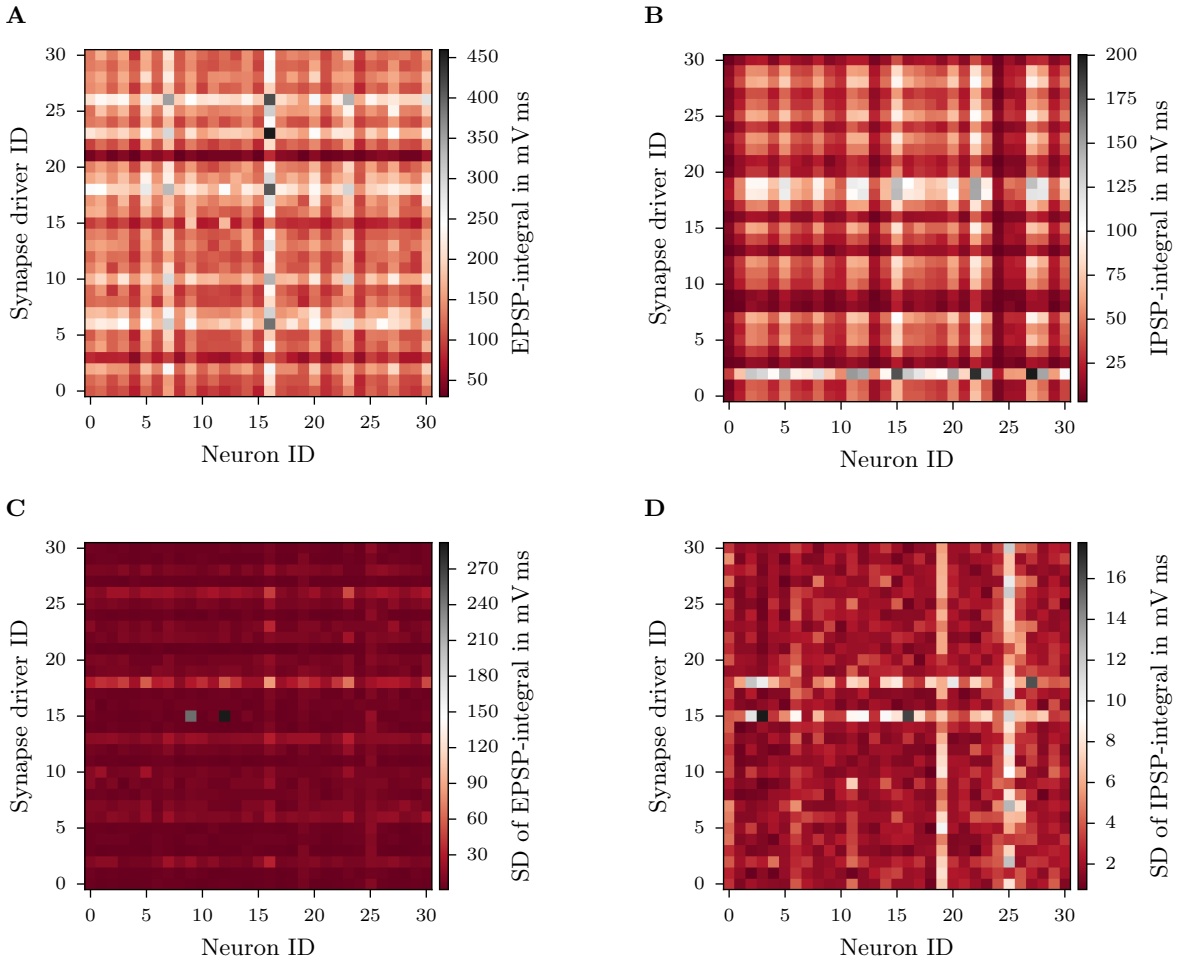

Figure 3.2: Average PSP-integrals and their standard deviation for various driver-neuron combinations. Results are shown for 961 combinations of line drivers and neurons with the neuron ID varying on the x-axis and the driver ID on the y-axis. PSP-integrals are averaged over 10 independent hardware runs. **A** Integrals of excitatory postsynaptic potentials. The mean of the average EPSP-integrals across all driver-neuron combinations is 142.17 mV ms with a standard deviation of 52.75 mV ms. **B** Integrals of inhibitory postsynaptic potentials. The mean of the average IPSP-integrals across all driver-neuron combinations is 38.36 mV ms with a standard deviation of 28.95 mV ms. **C** Standard deviation of the EPSP-integrals over the 10 hardware runs. The mean standard deviation is 16.80 mV ms. **D** Standard deviation of the IPSP-integrals over the 10 hardware runs. The mean standard deviation is 3.59 mV ms.

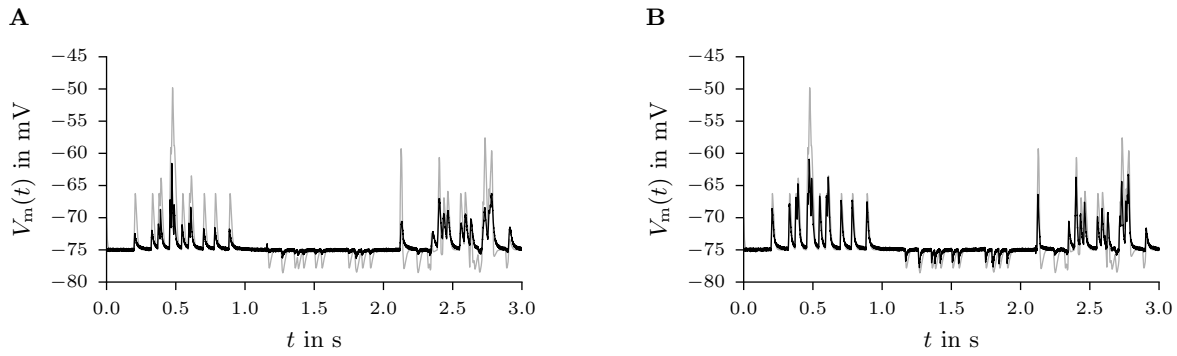

Figure 3.3: Comparison of the membrane potential $V_m(t)$ of a *Spikey* chip neuron to the membrane potential of a neuron simulated in NEST (*gray line*). The neurons receive an excitatory Poisson stimulus from 0.1 s to 0.9 s, an inhibitory Poisson stimulus from 1.1 s to 1.9 s, and from 2.1 s to 2.9 s they receive simultaneously an excitatory and inhibitory Poisson stimulus. The mean firing rate of the stimulus in each interval is 20 Hz. The synaptic conductances of the excitatory and inhibitory connections are set to 7 nS and 60 nS, respectively. Note that a value of 60 nS corresponds to the maximum possible value for synaptic conductances of inhibitory connections on the *Spikey* chip. An offset value is added to the recorded membrane trace such that the resting potential is equal to that of the NEST simulation. **A** Recorded membrane trace of the hardware neuron with ID 1. **B** Recorded membrane trace of the same neuron but the connecting synapses are driven by different line drivers.

3.5.2 Noise on the Membrane Potential

The neuron's membrane potential on the *Spikey* chip is superimposed with some temporal noise $\nu(t)$. This electronic noise is caused by a mixture of physical effects [39]. The most important types of noise are Johnson–Nyquist noise, $1/f$-noise, shot noise, and noise caused by the activity of neighboring circuit elements.

In the following the frequency content of $\nu(t)$ is analyzed and an estimate of its power is given. This is done by calculating the sample power spectral density (PSD) $\hat{S}_{\nu,N}(f)$ of $\nu(t)$ utilizing Welch's method with a Hann window with 50 % overlap. The segment length M is chosen to be 1024 samples. The PSD is estimated with $N = 10^5$ samples (sampled at a rate of $f_{s_1} = 9.6\,\mathrm{kHz}$) of $V_\mathrm{m}(t)$ at rest. The DC average of $V_\mathrm{m}(t)$, i.e., the leak reversal potential E_L, and a possible trend is removed by subtracting the result of a linear least-squares fit of each segment. The noise power $\hat{P}_\nu$ is obtained by averaging the spectral density over the signal bandwidth. In addition to that, the root mean square (RMS) voltage $\hat{V}_{\nu,\mathrm{RMS}}$ of the noise, which is equal to one standard deviation of $\nu(t)$, is computed.

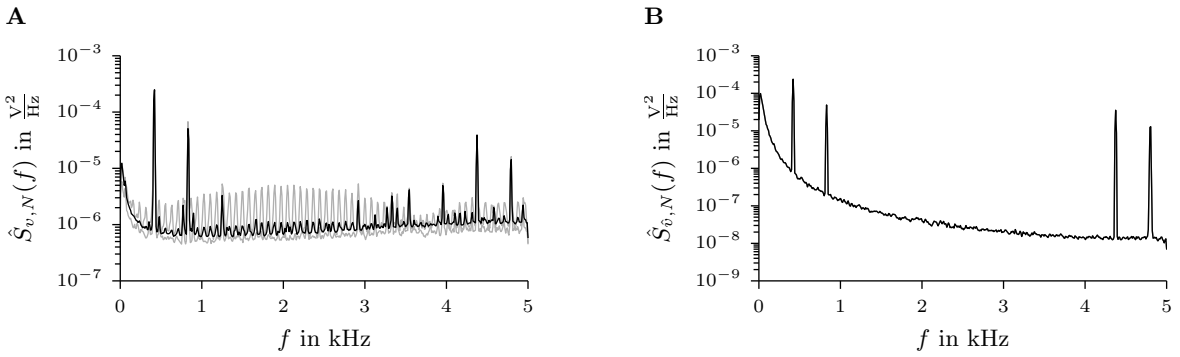

Figure 3.4: Sample PSD of the noise $\nu(t)$ on the membrane potential $V_\mathrm{m}(t)$ of the *Spikey* chip neurons and the sample PSD of its approximation $\hat{\nu}(t)$ which is used in the NEST simulations. The PSD are estimated with $N = 10^5$ samples by using Welch's method with a Hann window with 50 % overlap. The DC average of $V_\mathrm{m}(t)$, i.e., the leak reversal potential E_L, and a possible trend is removed by subtracting the result of a linear least-squares fit of each segment with length $M = 1024$. **A** Average PSD over 10 neurons of the *Spikey* chip (*black line*). The two gray lines show the PSD of $\nu(t)$ for the two neurons with the lowest and highest power in the shown frequency range. Note that the frequencies are stated in the biological time domain. **B** PSD of $\hat{\nu}(t)$ for one neuron simulated in NEST.

The average PSD over 10 neurons of the *Spikey* chip is shown in Fig. 3.4A. One can see that $\nu(t)$ consists of sinusoidal components at different frequencies and white noise. The estimate of the mean noise power of $\nu(t)$ is $\approx 2.2\,\mu\mathrm{V}^2\,\mathrm{Hz}^{-1}$ with a standard deviation of $\approx 0.29\,\mu\mathrm{V}^2\,\mathrm{Hz}^{-1}$. The mean RMS voltage of $\nu(t)$ is $\approx 0.11\,\mathrm{mV}$ with a standard deviation of $\approx 7.6\,\mu\mathrm{V}$. Two things need to be considered in this analysis: firstly, the PSD is calculated based on the measured data obtained from the readout circuit and therefore, it is not clear if the noise is already present at the neuron or if it is imposed by the subsequent measurement circuit, i.e., the connecting wire, analog-to-digital converter (ADC), etc. However, since the RMS voltage is only approximately 0.11 mV, and since there are imprecisions on much larger scales (see Section 3.5.1), the noise

on the membrane voltage is of minor importance. Secondly, the stated frequencies are given in the biological time domain. As a consequence, one can say that the peaks in the spectrum are imposed by noise sources which operate on frequencies 10^4 times larger than the frequencies shown in the PSD.

In NEST, the noise $\nu(t)$ is approximated by injecting a pseudo random white Gaussian noise current $\hat{\mu}(t)$ and sinusoidal currents with different frequencies and amplitudes into the neurons. Therefore, one can write the total induced current $i_{\hat{\nu}}(t)$ as a sum of the individual current sources, that is,

$$i_{\hat{\nu}}(t) = \hat{\mu}(t) + \sum_i A_i \sin(2\pi f_i t). \tag{3.4}$$

Due to the low-pass character of the neuron's membrane, the noise voltage $\hat{\nu}(t)$ on the membrane potential $V_\mathrm{m}(t)$ can be written as

$$\hat{\nu}(t) = h(t) * i_{\hat{\nu}}(t), \tag{3.5}$$

where $h(t)$ indicates the impulse response of the neurons membrane.

For simplicity, only sinusoidal components with $\hat{S}_{\nu,N}(f) > 5.0\,\mu\mathrm{V}^2\,\mathrm{Hz}^{-1}$ (where $\hat{S}_{\nu,N}(f)$ is the mean noise power over the 10 considered *Spikey* neurons) are considered in the approximation of the noise $\nu(t)$. Hence, the index i in Eq. 3.4 runs over all

$$(A, f)_i \in \{(44\,\mathrm{pA}, 420\,\mathrm{Hz}), (39\,\mathrm{pA}, 830\,\mathrm{Hz}), (126\,\mathrm{pA}, 4375\,\mathrm{Hz}), (90\,\mathrm{pA}, 4800\,\mathrm{Hz})\}.$$

The noise current $\hat{\mu}(t)$ is independently drawn for each neuron from a Gaussian distribution with zero mean and a standard deviation of 33 pA. The resulting sample PSD $\hat{S}_{\hat{\nu},N}(f)$ of $\hat{\nu}$ is estimated with $N = 10^5$ samples (sampled at a rate of $f_{\mathrm{s}_2} = 10\,\mathrm{kHz}$) from the resting membrane potential of one neuron simulated in NEST and is shown in Fig. 3.4B. Due to the inherent low-pass filtering the spectral distribution of the power is different than that of $\nu(t)$. However, the amplitudes of the noise currents $\hat{\mu}(t)$ and A_i are chosen such that $\hat{P}_{\hat{\nu}} = \hat{P}_\nu$ and $\hat{V}_{\hat{\nu},\mathrm{RMS}} = \hat{V}_{\nu,\mathrm{RMS}}$.

3.6 Software Interface

The model parameters mentioned in Section 3.4 are almost all represented by voltages and currents on the hardware. In order to emulate neural networks on the *Spikey* chip, an abstractly described neural network must be mapped to the appropriate hardware resources while considering the hardware specific constraints, and all biological model parameters have to be translated to the corresponding analog voltage and current parameters. Moreover, hardware parameters which have no translation to model parameters and parts of the chip circuitry which do not belong to the actual network, e.g., measurement circuits for the membrane potential, need to

be set, configured, and operated.

In order to hide these technical details from the user, a software package in multi-layer architecture was developed by the Electronic Vision(s) group. The software operating the hardware can be divided into the Python package PyNN[5], the hardware abstraction layer (HAL), and the low-level code [40]. In the following a short summary of this topic based on Bill [40] is given. For a more detailed explanation see the aforementioned source or Müller [41].

3.6.1 PyNN

The PyNN-project (Davison, Brüderle, Kremkow, *et al.* [42]) aims for the development of a simulator-independent language for building neuronal network models. PyNN is developed within the FACETS project and currently supports NEST (Gewaltig and Diesmann [43]), PCSIM (Pecevski, Natschläger, and Schuch [44]), NEURON (Hines and Carnevale [45]), Brian (Goodman and Brette [46]), and the *Spikey*[6] hardware.

The PyNN application programming interface (API) includes functions for creating neurons and making connections between them (e.g., create() and connect()), classes for generating stimuli (e.g., SpikeSourceArray), and functions to acquire simulation results (e.g., record()). Beside this procedural API, which gives a view of a model that is centered more on individual neurons and connections, the PyNN high-level API implements classes for the management of populations of neurons and projections between them (e.g., Population and Projection).

The functions of the pyNN.hardware.spikey module communicate with the lower software layer, i.e., the hardware abstraction layer.

3.6.2 Hardware Abstraction Layer

The hardware abstraction layer PyHAL (Brüderle, Grübl, Meier, *et al.* [47]) is implemented in the Python programming language and is used to store and organize the commands and network configurations received from the PyNN-layer. This layer's functionality is organized in two submodules, namely config and buildingblocks. The buildingblocks module contains classes for managing neurons, networks and their topology. The assignment of abstract neurons to hardware neurons, the translation of biological parameters to hardware parameters, and the communication to the low-level software is done by the config module.

3.6.3 Low-Level Code

The low-level, hardware specific code of the *Spikey* software package is implemented in C++. It communicates directly with the hardware and it provides functions to configure the hardware,

[5] PyNN (pronounced "pine") is a simulator-independent language for building neuronal network models, see `http://neuralensemble.org/PyNN/`

[6] The *Spikey* module is not included in the release of the PyNN package, since it cannot be used without the hardware.

to send external input spike trains to the chip, and to read back the spike events from on chip neurons.

4

In-the-loop Training of Neuromorphic Hardware

In this chapter a method for training a feed-forward spiking neural network which is emulated on the *Spikey* neuromorphic hardware is presented. The utilized algorithm, i.e., backpropagation, can be roughly divided into a two phase cycle, propagation and weight update. The input data is converted into discrete events, i.e., spikes, and is then presented to the network where it propagates through the subsequent layers until it reaches the output layer. The spike events of all neurons during this forward propagation are recorded and interpreted as output values of corresponding artificial neurons which use a binary activation function. An error value which represents the discrepancy between the desired network output and the actual network output is computed—using a loss function—for each artificial output neuron and is then propagated backwards, starting from the output layer, until each artificial neuron has an associated error value which represents its contribution to the overall error. The gradient of the loss function with respect to the weights in the network is then computed by using the obtained error values. Finally, an optimization algorithm aims to minimize the loss function by calculating an update value for each weight which is proportional to the obtained gradient. Note that only the forward propagation is carried out by the spiking neural network while the computation of the error, the backward propagation of errors, and the calculation of the new weights is done in an abstract model of the network, which comprises the artificial neurons, implemented in TensorFlow [48]. A detailed description of the learning algorithm and the utilized network model is given in Section 4.1.

In order to test the capability of the algorithm to train a spiking neural network—it is beyond question that it works for ordinary artificial neural networks—it is applied on two classification problems. A binary classification problem defined by the XOR logical function is presented in Section 4.2. In this task it is evaluated whether a spiking neural network trained by the presented algorithm is able to successfully perform the classification of non-linearly separable data. In Section 4.3 a network is trained to classify a subset of the MNIST hand written digit data set. Both tasks are carried out by a feed-forward spiking neural network emulated on *Spikey* and, for comparison, on a software model of the network simulated in NEST.

4.1 The Learning Algorithm

This section gives a detailed description of the training steps. It includes the initialization of weight and bias values, their quantization, and their mapping to synaptic conductances, the forward propagation step which is carried out by the spiking neural network, the backward propagation step including an explanation of the abstract software model including the artificial neuron's activation function and its approximate derivative, and finally the weight update step. An illustration of the training procedure is shown in Fig. 4.1.

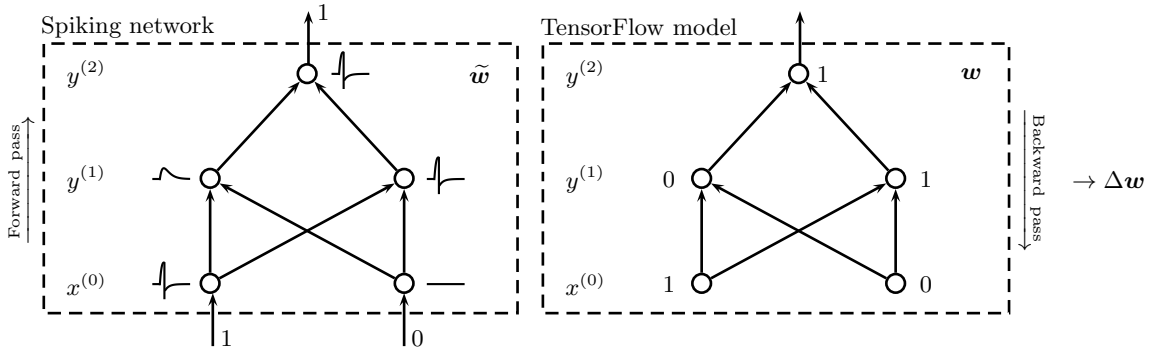

Figure 4.1: Schemata of the spiking network (*left*) and the TensorFlow model (*right*). The input layer, hidden layer, and the output layer are labeled with $x^{(0)}$, $y^{(1)}$, and $y^{(2)}$, respectively. The input spikes are presented to the spiking network which implements the forward pass. The spike events of the hidden neurons and the spike events of the output neurons are then extracted and interpreted as output of the corresponding artificial neuron in the TensorFlow model. The backpropagation step is then performed in the TensorFlow model based on the imposed hidden and output layers outputs. Finally, $\Delta \mathbf{w}$ is used to update the weights in both networks.

Weight Initialization and Quantization

In each run a different set of initial weigh values $w_{i,j}$ are used. The weights are independently drawn from a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with a certain mean μ and standard deviation σ where values whose magnitude is more than two standard deviations from the mean are dropped and re-picked. The values μ and σ are chosen according to the task in question (see Section 4.2 and Section 4.3). Bias values b_j are all initialized to the same value.

The weight and bias values are then quantized at $n = 4\,\text{bit}$ according to a uniform mid-tread quantizer with a step size of $\Delta = 1/(2^n - 1)$ according to the quantization function $Q(w_{i,j})$, that is

$$Q(w_{i,j}) = \Delta \left\lfloor \frac{w_{i,j}}{\Delta} + \frac{1}{2} \right\rfloor, \tag{4.1}$$

where $\lfloor \cdot \rfloor$ is the floor function. The corresponding staircase transfer function is shown in Fig. 4.2A. The quantized values are then mapped to synaptic conductances $\tilde{w}_{i,j}$ according to

$$\tilde{w}_{i,j} = \begin{cases} g_{\text{inh}}^{\min} \frac{1}{\Delta} Q(-w_{i,j}) & \text{if } w_{i,j} < 0, \\ \\ g_{\text{exc}}^{\min} \frac{1}{\Delta} Q(w_{i,j}) & \text{if } w_{i,j} \geq 0, \end{cases} \tag{4.2}$$

where $g_{\text{exc}}^{\min} = 1\,\text{nS}$ and $g_{\text{inh}}^{\min} = 4\,\text{nS}$ are the minimum excitatory and inhibitory conductances, respectively. The mapping of the quantized weight values to synaptic conductances $\tilde{w}_{i,j}$ is depicted in Fig. 4.2A. Note that in the NEST simulations, conductances $\{\tilde{w}_{i,j} \mid Q(w_{i,j}) \in [-15\Delta, 0)\}$ are multiplied with -1 to meet the requirement of negative synaptic strengths of inhibitory connections in NEST. This is in contrast to the *Spikey* chip where these values have to be greater or equal to zero. The obtained synaptic conductances are then used as initial values for the strength of synaptic connections in the spiking neural network.

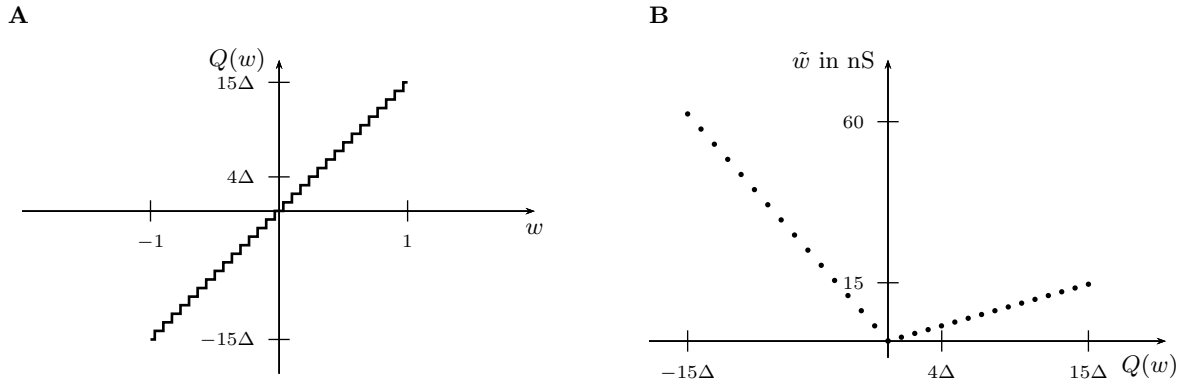

Figure 4.2: Weight quantization and the mapping of quantized weights to synaptic conductances. **A** Quantization function $Q(w)$ according to Eq. 4.1. The high precision values $w \in [-1, 1]$, which are obtained from the TensorFlow model after each weight update (or from the weight initialization step), are quantized to 31 distinct values. **B** Mapping of the quantized weights to synaptic conductances $\tilde{w}$ according to Eq. 4.2. Note that in the NEST simulations, conductances $\{\tilde{w} \mid Q(w) \in [-15\Delta, 0)\}$ are multiplied with -1 to meet the requirement of negative synaptic strengths of inhibitory connections in NEST.

Forward Propagation

A detailed scheme of the spiking neural network which carries out the forward propagation is shown exemplary in Fig. 4.3. Since neurons of the *Spikey* chip can only make outgoing connections which are either all excitatory or inhibitory, it is necessary to use two populations of neurons, i.e., one excitatory and one inhibitory population, in the input layer and in each hidden layer. A pair of an excitatory neuron and the corresponding inhibitory neuron is hereafter referred to as a "unit". This network topology gives rise to two problems: one the one hand, it makes learning more complex since twice as much weights in the input layer and in the hidden layers have to be learned, on the other hand, it makes the network more prone to variations in

the strength of the synapse drivers and in neuron parameters (see Section 3.5 for more details on these variations). The latter could in principle be compensated by the learning algorithm, provided that the weight resolution is small enough. The constraint on neurons of the *Spikey*

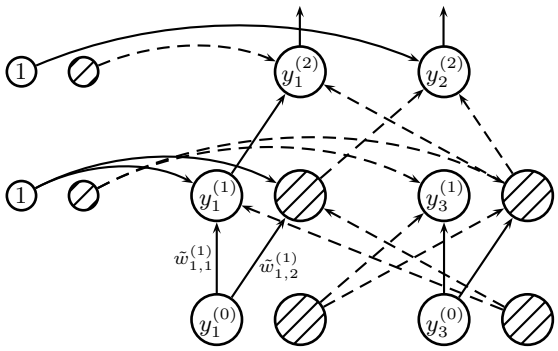

Figure 4.3: Detailed scheme of the used network topology on *Spikey* demonstrated with a network with two input units, two hidden units, and two output neurons. Each unit of the input and the hidden layer consists of an excitatory neuron (*large circles*) and an inhibitory neuron (*large shaded circles*). The bias in the hidden layer(s) and the output layer is realized with one excitatory neuron (*small circles*) and one inhibitory neuron (*small shaded circles*) per layer. The weights are learned separately for each connection.

chip that outgoing connections have to be of the same type does not apply to neurons simulated in NEST. For simplicity, the networks simulated in NEST are therefore comprised of only one neuron per unit.

The input vector $\boldsymbol{x} \in \{0,1\}^{n\times 1}$ is converted into spike events of the input neurons. The excitatory neuron $y_i^{(0)}$ and the inhibitory neuron $y_{i+1}^{(0)}$ of the input layer emit one spike at time $t_0^{(f)}$ if $x_{\lceil\frac{i+1}{2}\rceil} = 1$ where $i \in \{2k+1 \mid k \in \mathbb{N}_0, k < n\}$. Note that excitatory neurons are odd-indexed and inhibitory neurons are even-indexed. Setting the spike times of the bias neurons of the output layer is not straightforward since the exact spike times of the neurons in the hidden layer depend on the synaptic weight. For simplicity, it was chosen that the bias neurons in the output layer spike at $t_0^{(f)} + d$, where the delay d is empirically determined.

These spike events are transmitted via synaptic connections to a post-synaptic neuron of the subsequent layer $l \in \{1, 2, \ldots, L\}$ where they modify excitatory $g^{\text{exc}}(t)$ and inhibitory $g^{\text{inh}}(t)$ synaptic conductances of this neuron. The time course of this conductance change is given by

$$g^{\text{exc}}(t) = g_{\text{exc}}^{\min}\frac{1}{\Delta}Q\left(w^{(l)}_{\lceil\frac{i+1}{2}\rceil,j}\right)\sum_f \epsilon\left(t - t_i^{(f)}\right) \quad \text{if} \quad w^{(l)}_{\lceil\frac{i+1}{2}\rceil,j} > 0 \tag{4.3}$$

and

$$g^{\text{inh}}(t) = g_{\text{inh}}^{\min}\frac{1}{\Delta}Q\left(-w^{(l)}_{\lceil\frac{i+1}{2}\rceil,j}\right)\sum_f \epsilon\left(t - t_i^{(f)}\right) \quad \text{if} \quad w^{(l)}_{\lceil\frac{i+1}{2}\rceil,j} < 0, \tag{4.4}$$

respectively. The sum runs over all spike times of excitatory $y_i^{(l-1)}$ and inhibitory $y_{i+1}^{(l-1)}$ pre-

synaptic neurons and $w^{(l)}_{\lceil \frac{i+1}{2} \rceil, j}$ is the synaptic efficiency from a pre-synaptic neuron to a post-synaptic neuron $y^{(l)}_j$ in the subsequent layer. The sum can be omitted if the pre-synaptic neuron is an input neuron, i.e., $y^{(0)}_i$ and $y^{(0)}_{i+1}$, since each input neuron emits at most one spike for each input vector. The ϵ-kernel defines the shape of this post-synaptic conductance course which is described in Section 3.3.

Note that the indices of the pre-synaptic neurons are restricted to $i \in \{2k+1 \mid k \in \mathbb{N}_0, k < \frac{m}{2}\}$ where m is the number of neurons (including excitatory and inhibitory neurons) in that layer. This is necessary since either excitatory or inhibitory connections are made between two nodes. The membrane potential $V_{\mathrm{m},j}(t)$ of a post-synaptic neuron $y^{(l)}_j$ depends on $g^{\mathrm{exc}}(t)$ and $g^{\mathrm{inh}}(t)$. Its evolution is is given by the differential equation

$$-C_{\mathrm{m}} \frac{dV_{\mathrm{m},j}(t)}{dt} = g_{\mathrm{L}}(V_{\mathrm{m},j}(t) - E_{\mathrm{L}}) + \sum_{s \in \mathcal{S}_{\mathrm{exc}}} g^{\mathrm{exc}}_s(t)\,(V_{\mathrm{m},j}(t) - E_{\mathrm{exc}}) + \sum_{s \in \mathcal{S}_{\mathrm{inh}}} g^{\mathrm{inh}}_s(t)\,(V_{\mathrm{m},j}(t) - E_{\mathrm{inh}})\,, \tag{4.5}$$

where the sum runs over all excitatory $\mathcal{S}_{\mathrm{exc}}$ and inhibitory $\mathcal{S}_{\mathrm{inh}}$ synapses, respectively. For a detailed description of this neuron model see Section 3.2.

If $V_{\mathrm{m},j}(t)$ reaches the threshold potential V_{th} a spike is generated and $V_{\mathrm{m},j}(t)$ is held to the reset potential V_{reset} for a refractory period of t_{refrac}. After t_{refrac} the neuron might emit one or more further spikes depending on the amplitude and time course of its synaptic conductance. If neuron $y^{(l)}_j$ is in the output layer the forward propagation step is completed.

Backward Propagation

The backward propagation step is performed in an abstract software model of the spiking neural network. The output $o^{(l)}_j$ of an abstract neuron j of a unit in layer l is the weighted sum of outputs $o^{(l-1)}_i$ of neuron i in the previous layer $l-1$.

$$o^{(l)}_j = \varphi\left(\sum_{i=1}^{n} w^{(l)}_{i,j} o^{(l-1)}_i\right), \tag{4.6}$$

If neuron j is in the first layer after the input layer, then $o^{(l-1)}_i$ is simply x_i. Note that $w^{(l)}_{i,j}$ are the high precision weight value between neuron i and neuron j. The artificial neurons use the nonlinear activation function

$$\varphi(x) = H(x - \theta), \tag{4.7}$$

where $H(x-\theta)$ is the shifted unit step function, i.e.,

$$H(x-\theta) = \begin{cases} 0 & \text{if } x < \theta, \\ \\ 1 & \text{else.} \end{cases} \tag{4.8}$$

The threshold θ is set to 8Δ where Δ is the quantization step size (see Eq. 4.1). This corresponds to a synaptic conductance $\tilde{w}$ of 8 nS. The threshold potential V_{th} of the *Spikey* chip neurons is set such that the neuron spikes if it receives a pre-synaptic spike with synaptic efficacy of 8 nS.

Since the derivative of $H(x-\theta)$ is given by

$$\frac{d}{dx}H(x-\theta) = \delta(x-\theta), \tag{4.9}$$

where $\delta(\cdot)$ is is the Dirac delta, which is not suitable for backpropagation, the derivative of the activation function $\varphi(x)$ is instead defined as

$$\frac{d}{dx}\varphi(x) \coloneqq \max(0, 1-|x-\theta|). \tag{4.10}$$

The activation function $\varphi(x)$ and its derivative is shown in Fig. 4.4A and Fig. 4.4B, respectively.

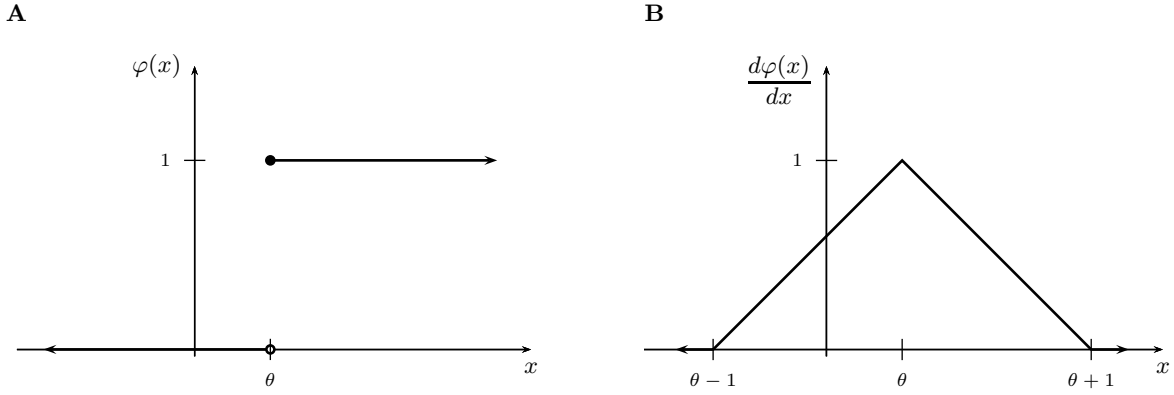

Figure 4.4: Activation function $\varphi(x)$ of the artificial neurons used in the TensorFlow model. Since the derivative of $\varphi(x)$ is not suitable for backpropagation, its derivative is instead defined as $\frac{d\varphi(x)}{dx} \coloneqq \max(0, 1-|x-\theta|)$. A threshold value $\theta > 0$ is necessary in order to prevent, under consideration of an appropriate mapping between θ and the threshold potential V_{th}, the neurons in the spiking network to emit spikes in the absence of an activation, i.e., when their membrane potential $V_{\text{m}}(t)$ is at the resting potential. **A** Activation function $\varphi(x)$ according to Eq. 4.7. **B** Approximation of the derivative of $\varphi(x)$ as defined in Eq. 4.10.

For each output neuron the error is defined as the squared error function, that is

$$E = \frac{1}{2}\left(t_i - \hat{o}_i^{(L)}\right)^2, \tag{4.11}$$

where E is the squared error, t is the target output, and $\hat{o}_i^{(L)}$ is assumed to be the output of an

artificial neuron of the output layer and is set to one if the corresponding neuron in the spiking neural network emitted at least one spike during the forward propagation. Otherwise it is set to zero. The gradient of the error with respect to a weight $w_{i,j}^{(l)}$ is

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \frac{\partial E}{\partial \hat{o}_j^{(l)}} \frac{\partial \hat{o}_j^{(l)}}{\partial \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)}} \frac{\partial \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)}}{\partial w_{i,j}^{(l)}}, \tag{4.12}$$

where

$$\frac{\partial \sum_i w_{i,j}^{(l-1)} \hat{o}_i^{(l-1)}}{\partial w_{i,j}^{(l)}} = \hat{o}_i^{(l-1)}. \tag{4.13}$$

If the neuron is in the first layer after the input layer, i.e., if $l = 1$, then $\hat{o}_i^{(l-1)}$ is just x_i. The derivative of the output $\hat{o}_j^{(l)}$ of neuron j with respect to its activation is per definition

$$\frac{\partial \hat{o}_j^{(l)}}{\partial \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)}} = \max\Big(0, 1 - \Big| \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)} - \theta \Big| \Big). \tag{4.14}$$

The partial derivative of the error with respect to the output of neuron j is

$$\frac{\partial E}{\partial \hat{o}_j^{(l)}} = \frac{\partial E}{\hat{o}_i^{(L)}} = \hat{o}_i^{(L)} - t_i, \tag{4.15}$$

if neuron j is in the output layer. If neuron j is in an inner layer one has

$$\frac{\partial E}{\partial \hat{o}_j^{(l)}} = \sum_k \left(\frac{\partial E}{\partial \hat{o}_k^{(l+1)}} \frac{\partial \hat{o}_k^{(l+1)}}{\partial \sum_j w_{j,k}^{(l)} \hat{o}_k^{(l+1)}} w_{j,k}^{(l)} \right), \tag{4.16}$$

where the sum runs over all neurons k which receive input from neuron j. Hence, the derivative with respect to $\hat{o}_j^{(l)}$ can be calculated if all derivatives with respect to the outputs $\hat{o}_k^{(l+1)}$ of the next layer, i.e., the one closer to the output, are known.

Finally, the gradient of the error E with respect to the weights is

$$\frac{\partial E}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} \hat{o}_i^{(l-1)}, \tag{4.17}$$

with

$$\delta_j^{(l)} = \begin{cases} (\hat{o}_i^{(L)} - t_i) \max\Big(0, 1 - \Big| \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)} - \theta \Big| \Big) & \text{if } j \text{ is an output neuron,} \\ \Big(\sum_k \delta_k^{(l+1)} w_{j,k}^{(l)} \Big) \max\Big(0, 1 - \Big| \sum_i w_{i,j}^{(l)} \hat{o}_i^{(l-1)} - \theta \Big| \Big) & \text{if } j \text{ is an inner neuron.} \end{cases} \tag{4.18}$$

The change in weight is then computed according to the Adam optimizer [49], i.e.,

$$\Delta w_{i,j}^{(l)} = -\lambda_n \frac{m_n}{\sqrt{v_n + \epsilon}}, \tag{4.19}$$

where λ is the learning rate, m is the 1st moment estimate, v is the 2nd moment estimate, and $\epsilon = 1 \times 10^{-8}$. In each time step n, λ, m, and v are updated according to

$$\lambda_n = \lambda \frac{\sqrt{1 - \beta_2^n}}{1 - \beta_1^n} \tag{4.20}$$

$$m_n = \beta_1 m_{n-1} + (1 - \beta_1) \frac{\partial E}{\partial w_{i,j}^{(l)}} \tag{4.21}$$

$$v_n = \beta_2 v_{n-1} + (1 - \beta_2) \left(\frac{\partial E}{\partial w_{i,j}^{(l)}} \right)^2 \tag{4.22}$$

with initial values $m_0 = 0$, $v_0 = 0$. The exponential decay rate for the 1st moment estimate is $\beta_1 = 0.9$, and the exponential decay rate for the 2nd moment estimate is $\beta_1 = 0.999$. The learning rate λ is chosen according to the task in question. The new weights are then clipped to the interval $[-1, 1]$ and quantized according to Eq.4.1. The propagation and weight update cycle is repeated until the error is sufficiently small.

4.2 XOR Classification Task

The XOR function is a typical example of a not linearly separable Boolean function. The function is of the form $f\colon B^2 \to B$, where $B = \{0, 1\}$ is the Boolean domain consisting of exactly two elements whose interpretations include *false* and *true*. The truth table of the XOR function is shown in Table 4.1. The function computes the logical exclusive or, which yields true if and only if the two inputs have different truth values. One can also define the XOR function

Table 4.1: Truth table of the XOR function. The value of the output variable y is only true when the inputs x_1 and x_2 differ, i.e., if one is true and the other is false.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

in the form $f\colon B^2 \to B^2$. In this case, the two output variables y_1 and y_2 encode the logical operation of its inputs in a one-hot manner. Consequently, the output is considered as false if y_1 is true and y_2 is false and is considered true if y_1 is false and y_2 is true (see the truth table in Table 4.2).

Table 4.2: Truth table of the XOR function with one-hot encoding of the output. The logical operation outputs true only when the inputs x_1 and x_2 differ, i.e., if one is true and the other is false.

x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

The neural networks utilized to produce the truth tables shown in Table 4.1 and Table 4.2 are illustrated in Fig.4.5A and Fig.4.5B, respectively. The network shown in Fig.4.5A consists of two input units, two hidden units, one bias unit associated to the hidden and the output layer, and one output neuron. The network depicted in Fig.4.5A has a similar topology but the output layer consists of two neurons in order to represent the two output variables shown in Table 4.2.

In each training step of the network the full batch is used for training, i.e., the input data set is

$$\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}^{\mathrm{T}} \tag{4.23}$$

and the targets are $\boldsymbol{t} = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}^{\mathrm{T}}$ and $\boldsymbol{t} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}^{\mathrm{T}}$, respectively.

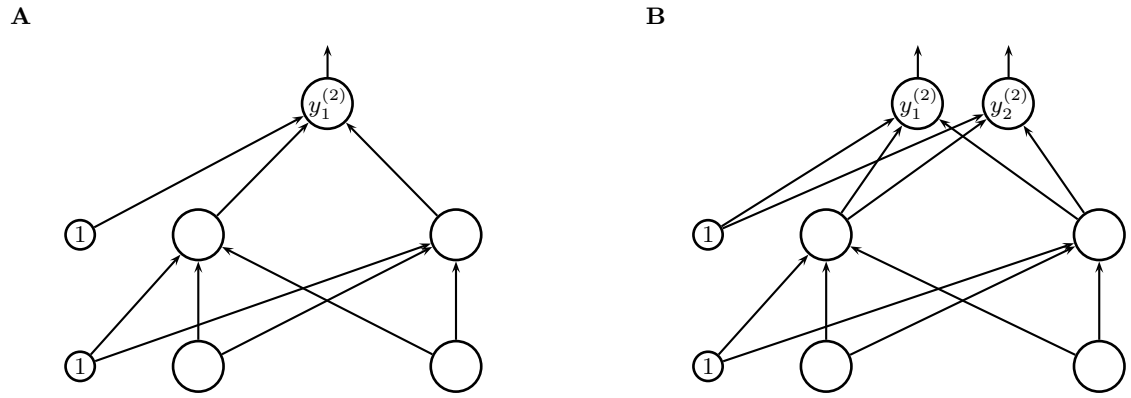

Figure 4.5: Topologies of the feed-forward neural networks which are used in the XOR classification task. The networks consist of one input layer, one hidden layer, and one output layer. A bias unit (*small circles*) is associated to the hidden and output layer. **A** Minimal network which is required to model the XOR problem. The network consists of two input units, two hidden units, and one output neuron. **B** Similar network as in **A** but the label layer consists of two neurons. This enables one-hot coding of the target output.

4.2.1 Results of the XOR Classification Task—NEST

In the following, the NEST simulation results of the XOR task are shown. Training is performed for 400 epochs with a learning rate λ of 0.01. The individual training examples are presented to the network every 100 ms starting at $t = 50$ ms. The required simulation time for one epoch is therefore 400 ms. The initial weights are drawn from a Gaussian distribution with a mean of 0.3 and a standard deviation of 0.1. The bias values are uniformly initialized to 0.1.

The average evolution of the training error for both of the considered network topologies over 20 independent runs in which the network output agrees with the target output within 400 epochs of training (each with a different initial set of weight and bias values) is shown in Fig. 4.6A and Fig. 4.6B, respectively. On average, the network with one neuron in the output layer correctly computes the XOR function after about 200 epochs of training while the network with two neurons in the output layer, i.e., the network which encodes the class labels in a one-hot fashion, needs for the same task about 300 epochs. Based on these results, it was chosen that the output layer of the networks which are emulated on *Spikey* and trained to classify the XOR function include only one neuron.

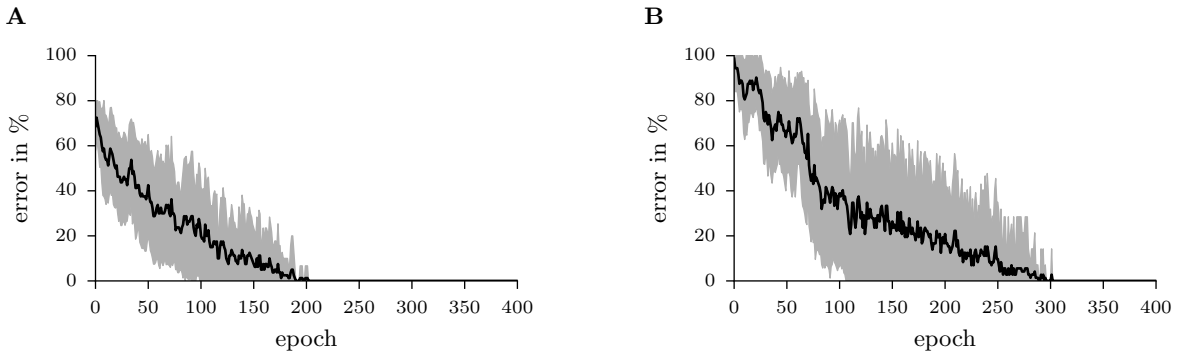

Figure 4.6: Evolution of the error during the training epochs. In each training step the full batch is used for training and the error is defined as the fractions of examples (in one batch) for which the output of the network does not agree with the target classification. **A** Classification error as a function of the training epoch. Shown is the mean error (*black line*) ±1 standard deviation (*shaded area*) over 20 independent runs in which the network output agrees with the target output within 400 epochs of training. **B** Same as in **A** but the output layer of the network consists of two neurons, i.e., the target output is encoded in one-hot manner.

XOR Task with Binary Encoding of the Class Labels

Figure 4.7 shows the evolution of the weight and bias values during training in a specific run. Shown are the quantized values. Weight and bias values of the hidden layer are shown in Fig. 4.7A and Fig. 4.7B shows the weight and bias values of the output layer. In this run, a training accuracy of 100% was reached after about 120 epochs. This can also be seen in the convergence of the weights.

Figure 4.8 shows the spike events of the input layer's neurons $y_i^{(0)}$, the hidden layer's neurons $y_j^{(1)}$, and the output neuron $y_1^{(2)}$ in the same run during training. In the left panel the first five epochs of training is shown and the right panel shown the last five epochs of training.

The time course of the membrane potential of the first hidden neuron, the second hidden neuron, and the output neuron in the test run is shown in Fig. 4.9, Fig. 4.10, and Fig. 4.11, respectively. In addition, the excitatory and inhibitory synaptic conductance course is shown in the bottom panel of these figures. Figure 4.11 nicely shows the effect of temporal summation of post-synaptic conductances due to the two pre-synaptic spikes of neuron $y_1^{(1)}$ between $t = 150\,\text{ms}$ and $t = 200\,\text{ms}$ (see Fig. 4.9). This, however, could be a problem during training since it is not considered by the artificial model.

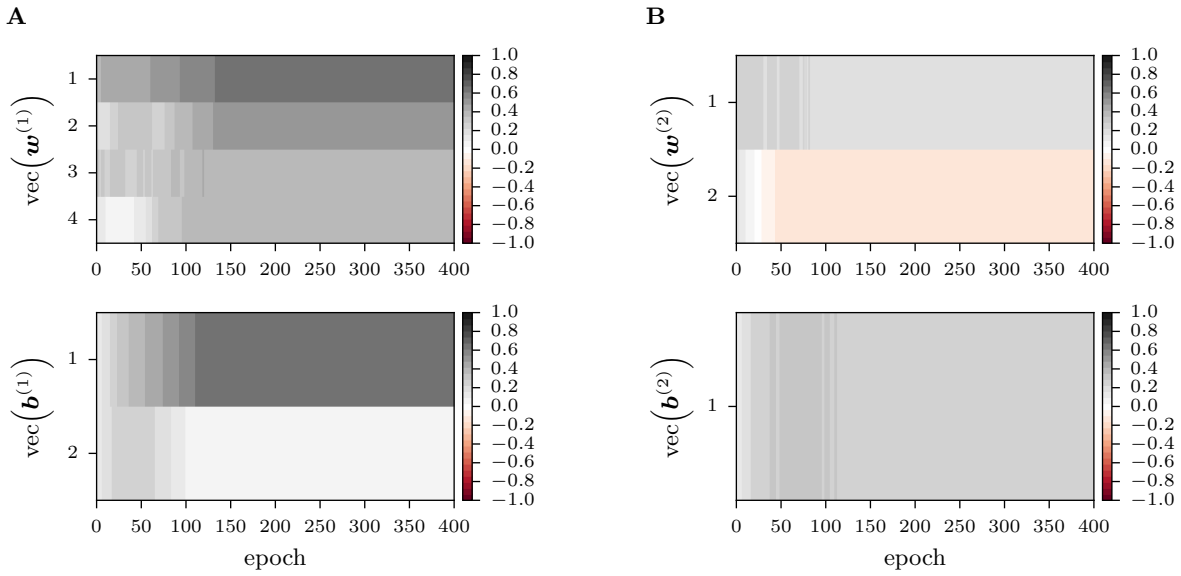

Figure 4.7: Evolution of the weight and bias values as a function of the training epoch for one specific run. **A** Quantized weight (*top panel*) and bias values (*bottom panel*) of the hidden layer. **B** Quantized weight (*top panel*) and bias values (*top panel*) of the output layer.

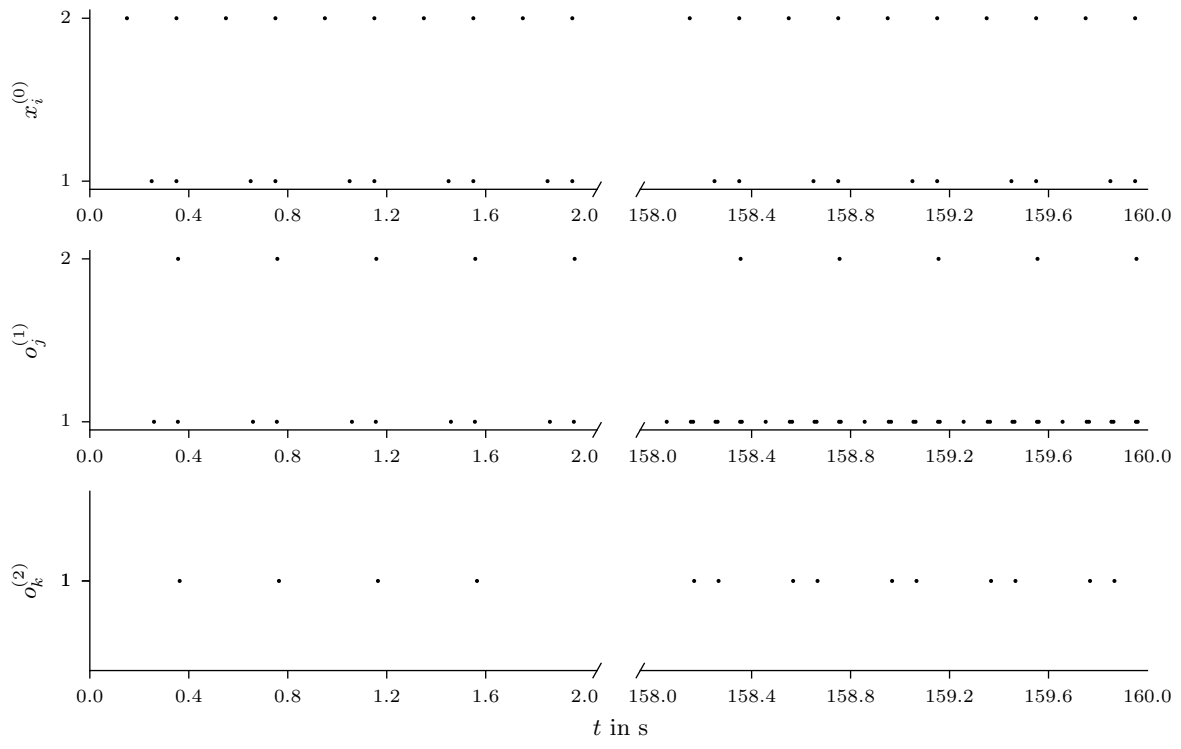

Figure 4.8: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the hidden layer's neurons $y_j^{(1)}$, and the output neuron $y_1^{(2)}$ during training. The left panels show the neural activity in the first 5 training epochs and the right panels show the neural activity in the last 5 training epochs. One epoch corresponds to a NEST simulation time of 400 ms.

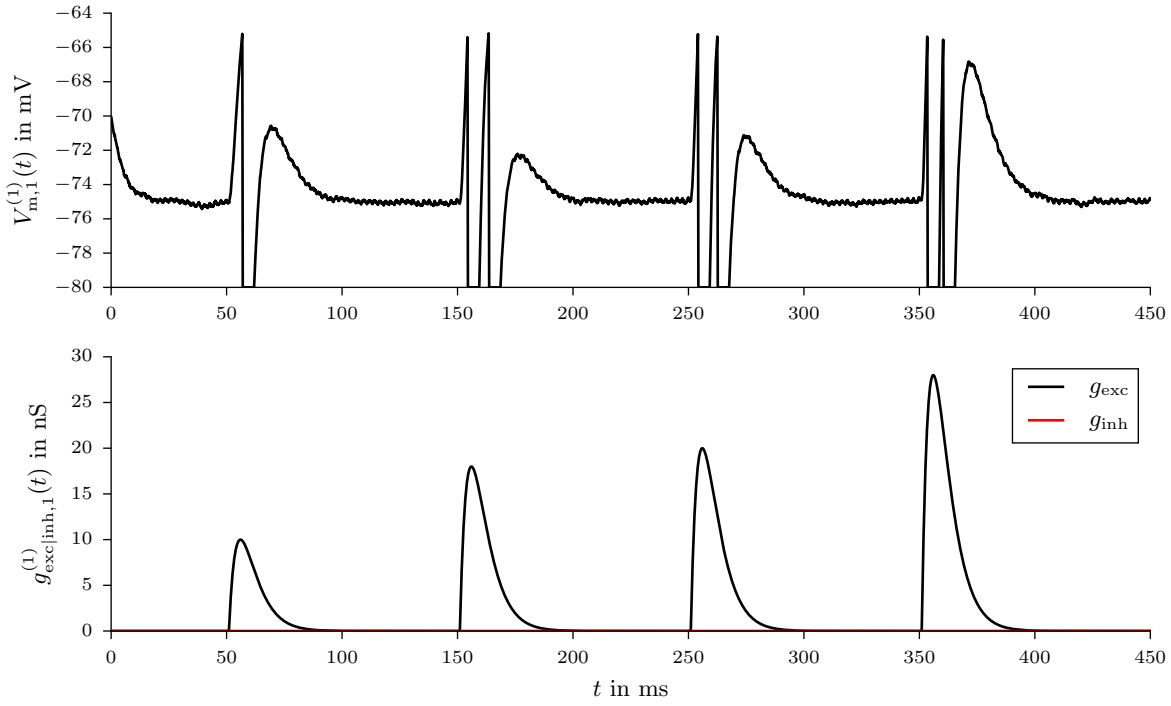

Figure 4.9: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_1^{(1)}$ of the hidden layer during the test run.

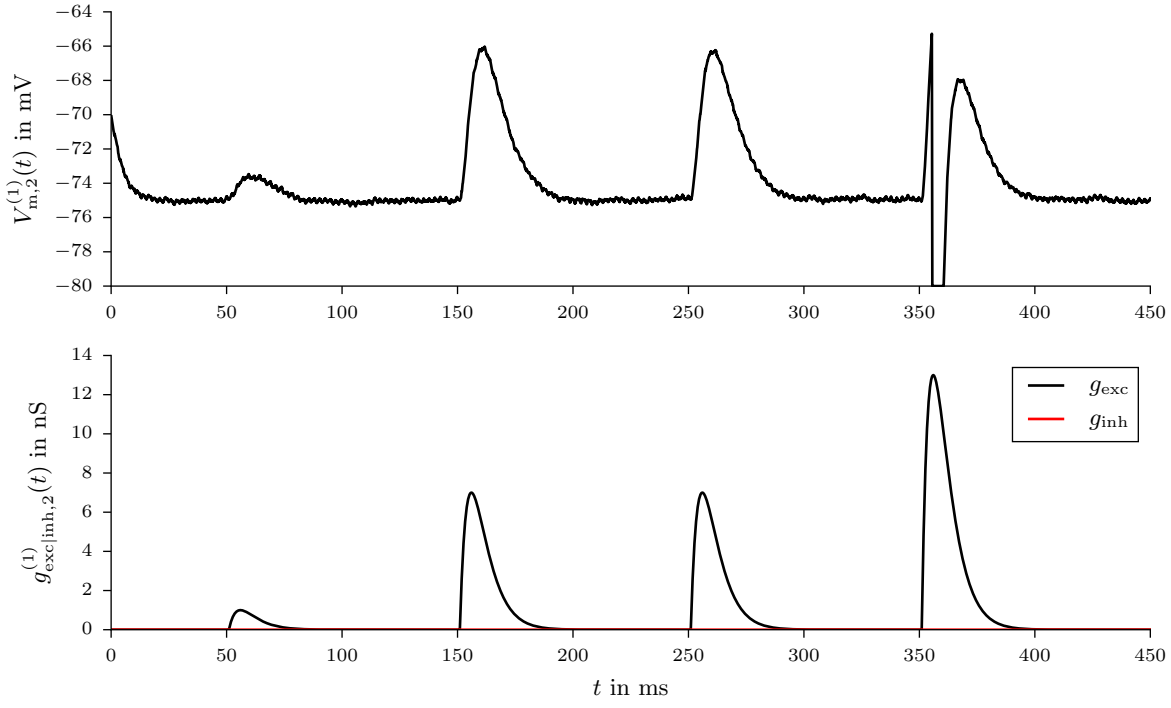

Figure 4.10: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_2^{(1)}$ of the hidden layer during the test run.

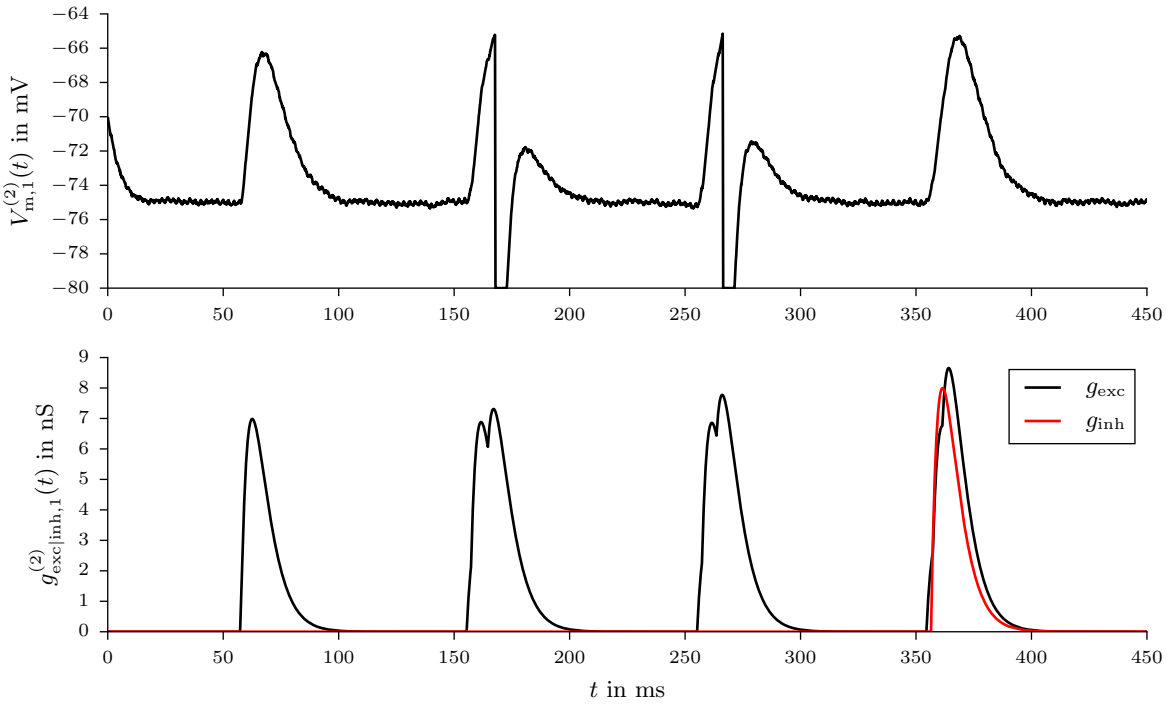

Figure 4.11: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_1^{(2)}$ of the output layer during the test run.

XOR Task with One-Hot Encoding of the Class Labels

Figure 4.12 shows the evolution of the weight and bias values during training in a specific run. Shown are the quantized values. Weight and bias values of the hidden layer are shown in Fig. 4.12A and Fig. 4.12B shows the weight and bias values of the output layer. In this run, a training accuracy of 100% was reached after about 300 epochs. This can also be seen in the convergence of the weights.

Figure 4.13 shows the spike events of the input layer's neurons $y_i^{(0)}$, the hidden layer's neurons $y_j^{(1)}$, and the output neurons $y_1^{(2)}$ and $y_2^{(2)}$ in the same run during training. In the left panel the first five epochs of training is shown and the right panel shown the last five epochs of training.

The time course of the membrane potential of the first hidden neuron, the second hidden neuron, and the output neurons in the test run is shown in Fig. 4.14, Fig. 4.15, Fig. 4.16, and Fig. 4.17, respectively. In addition, the excitatory and inhibitory synaptic conductance course is shown in the bottom panel of these figures.

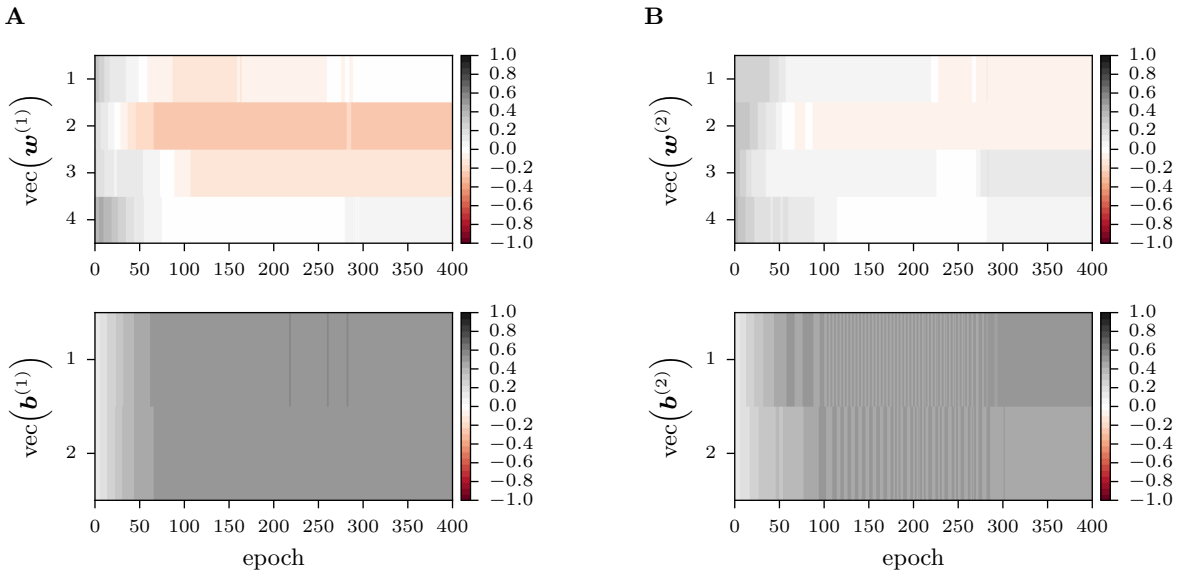

Figure 4.12: Evolution of the weight and bias values as a function of the training epoch for one specific run. **A** Quantized weight (*top panel*) and bias values (*bottom panel*) of the hidden layer. **B** Quantized weight (*top panel*) and bias values (*top panel*) of the output layer.

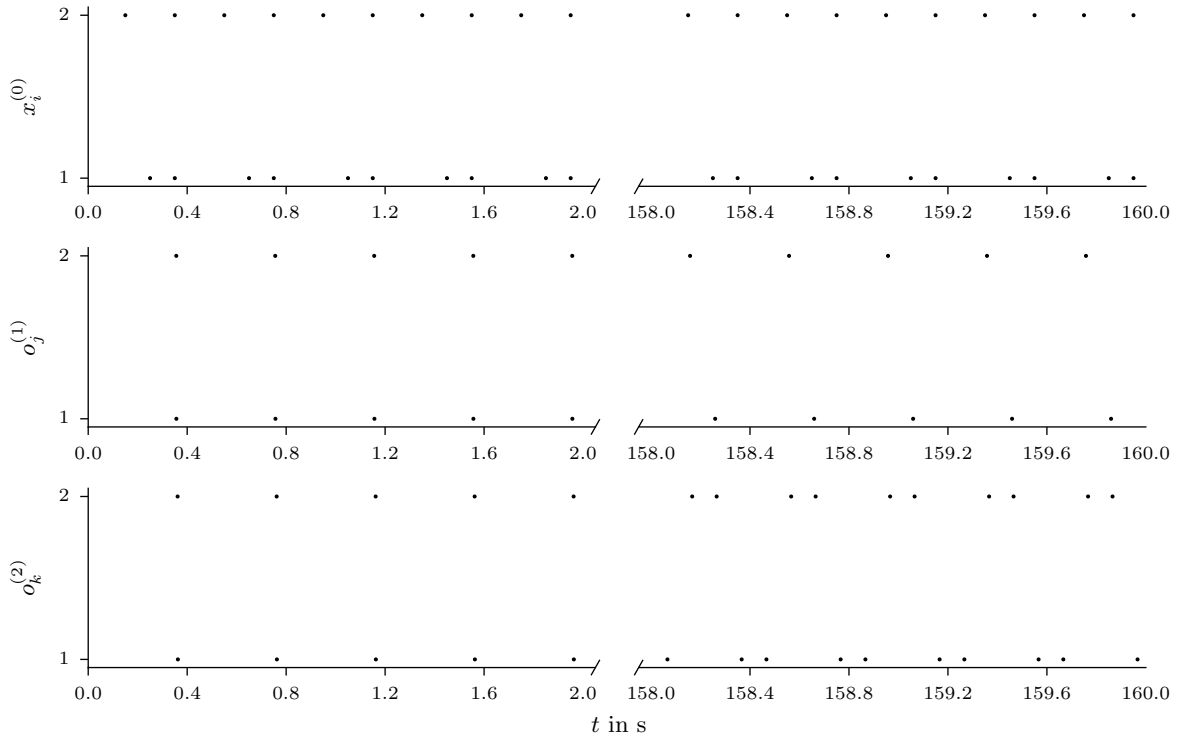

Figure 4.13: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the hidden layer's neurons $y_j^{(1)}$, and the output neurons $y_1^{(2)}$ and $y_2^{(2)}$ during training. The left panels show the neural activity in the first 5 training epochs and the right panels show the neural activity in the last 5 training epochs. One epoch corresponds to a NEST simulation time of 400 ms.

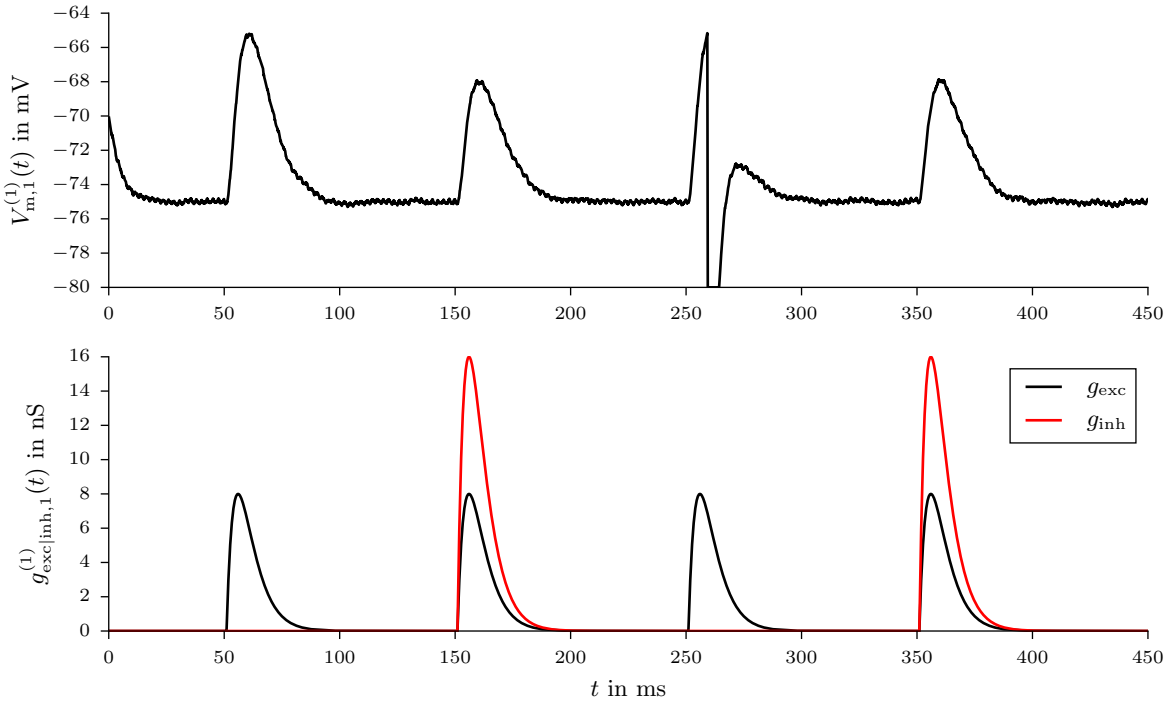

Figure 4.14: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_1^{(1)}$ of the hidden layer during the test run.

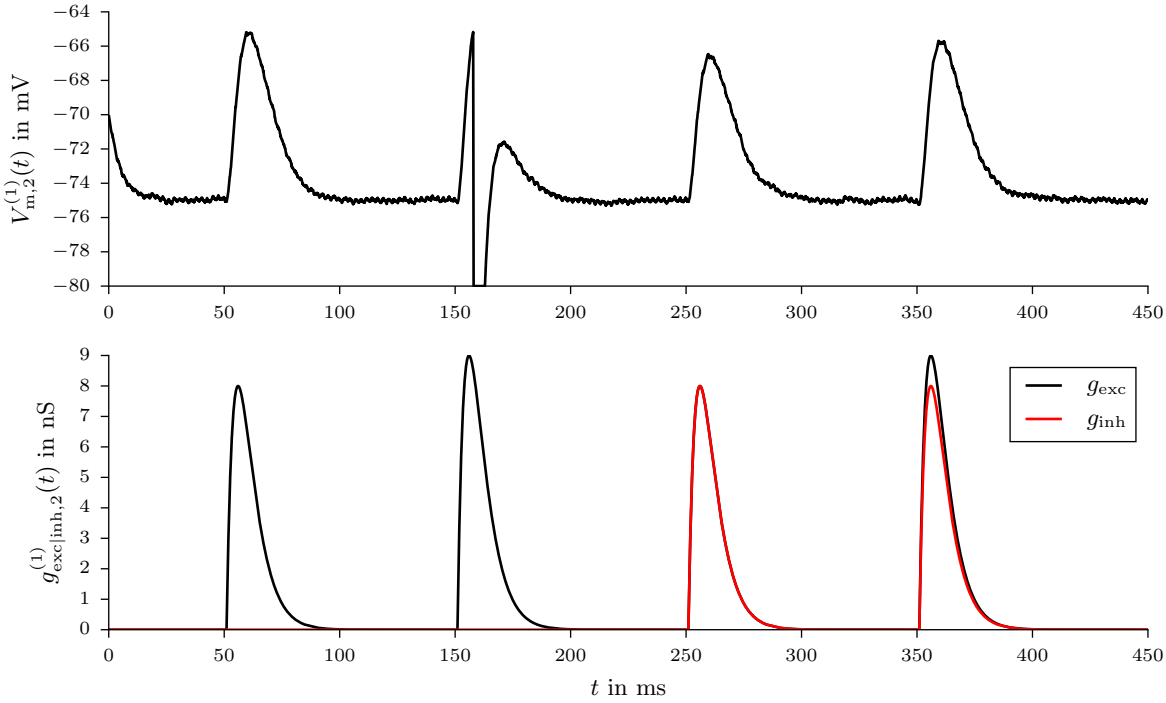

Figure 4.15: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_2^{(1)}$ of the hidden layer during the test run.

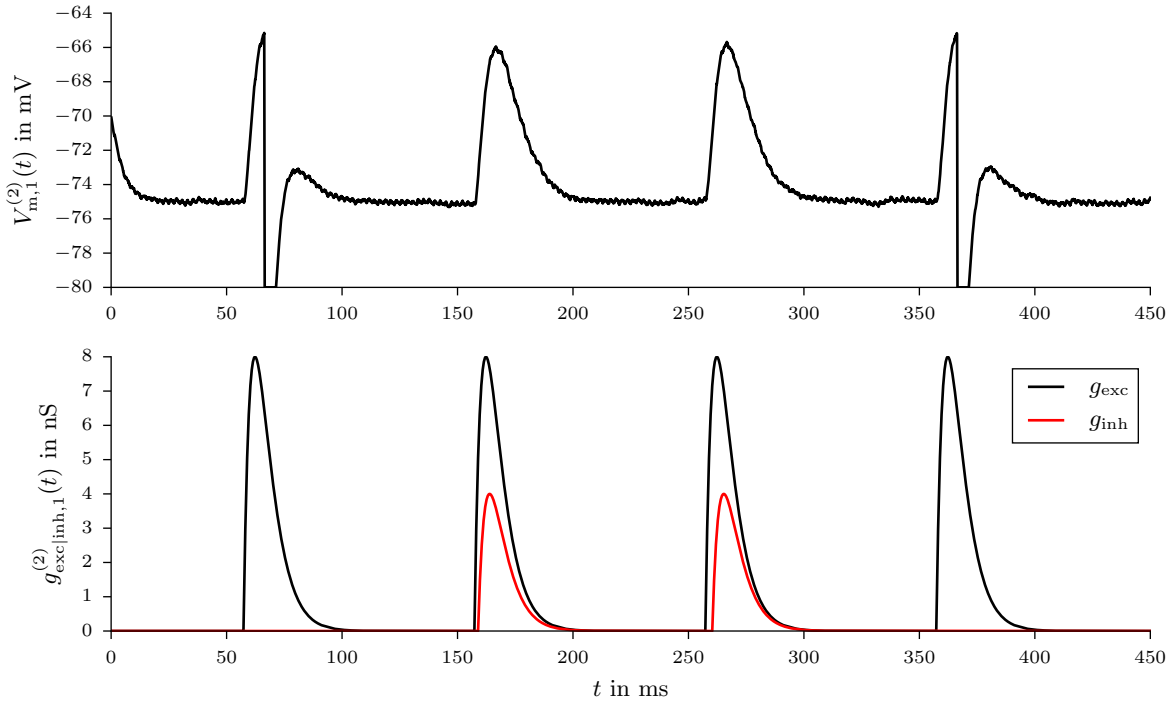

Figure 4.16: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_1^{(2)}$ of the output layer during the test run.

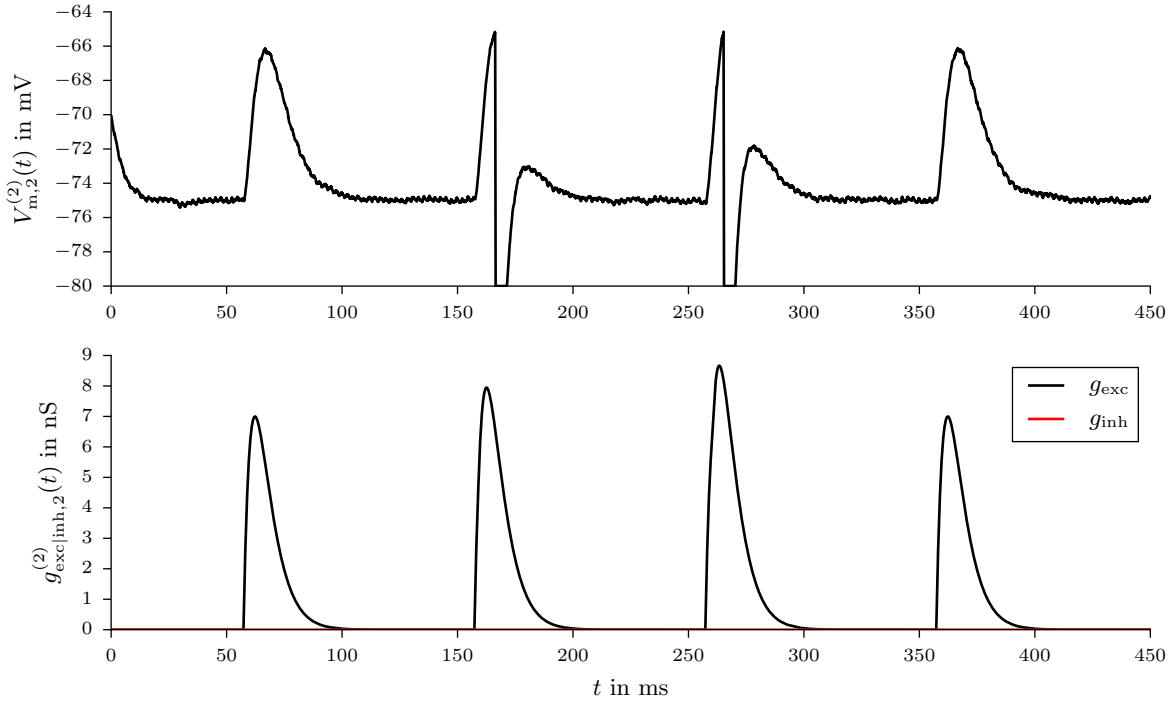

Figure 4.17: Membrane potential $V_{\mathrm{m}}(t)$ (*top panel*) and synaptic conductance course $g_{\mathrm{exc|inh}}(t)$ (*bottom panel*) of neuron $y_2^{(2)}$ of the output layer during the test run.

4.2.2 Results of the XOR Classification Task—Spikey

In the following, the *Spikey* emulation results of the XOR task are shown. Training is performed for 800 epochs with a learning rate λ of 0.01. The individual training examples are presented to the network every 100 ms starting at $t = 50$ ms. The required simulation time for one epoch is therefore 400 ms (biological time). The initial weights are drawn from a Gaussian distribution with a mean of 0.3 and a standard deviation of 0.1. The bias values are uniformly initialized to 0.1.

The average evolution of the training error over 20 independent hardware runs in which the network output agrees with the target output within 800 epochs of training (each with a different initial set of weight and bias values) is shown in Fig. 4.18. In each of the test runs, the network was able to correctly classify the XOR function (which was most likely just luck). However, no stable solution could be found as can be seen in Fig. 4.18. Even if training is performed for 1600 epochs, the classification error never stayed at zero. The cause for this fluctuation in the error lies probably in the trial-to-trial variability (see Section 3.5) which the training algorithm (in combination with the limited resolution of the synaptic weights) could not compensate.

Figure 4.19 shows the evolution of the weight and bias values during training in a specific run. Shown are the quantized values. Weight and bias values of connections from input neurons to excitatory neurons in the hidden layer are shown in Fig. 4.19A and Fig. 4.19B shows the weight and bias values of connections from input neurons to inhibitory neurons in the hidden layer. Fig. 4.20 shows the evolution of the weight and bias values over training.

Figure 4.21 shows the spike events of the excitatory input layer's neurons $y_i^{(0)}$, the excitatory hidden layer's neurons $y_j^{(1)}$, and the output neuron $y_1^{(2)}$ in the test run.

The time course of the membrane potential of the output neuron in the test run is shown in Fig. 4.22.

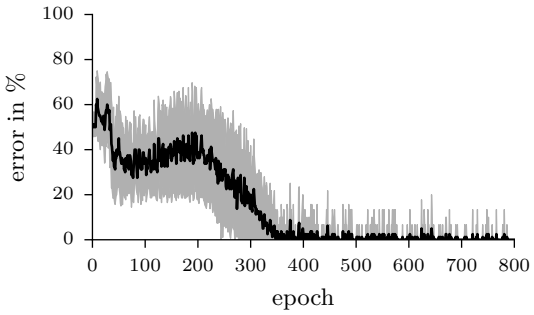

Figure 4.18: Evolution of the error during the training epochs. In each training step the full batch is used for training and the error is defined as the fractions of examples (in one batch) for which the output of the network does not agree with the target classification. Shown is the mean error (*black line*) ±1 standard deviation (*shaded area*) over 20 independent runs in which the network output agrees with the target output within 800 epochs of training.

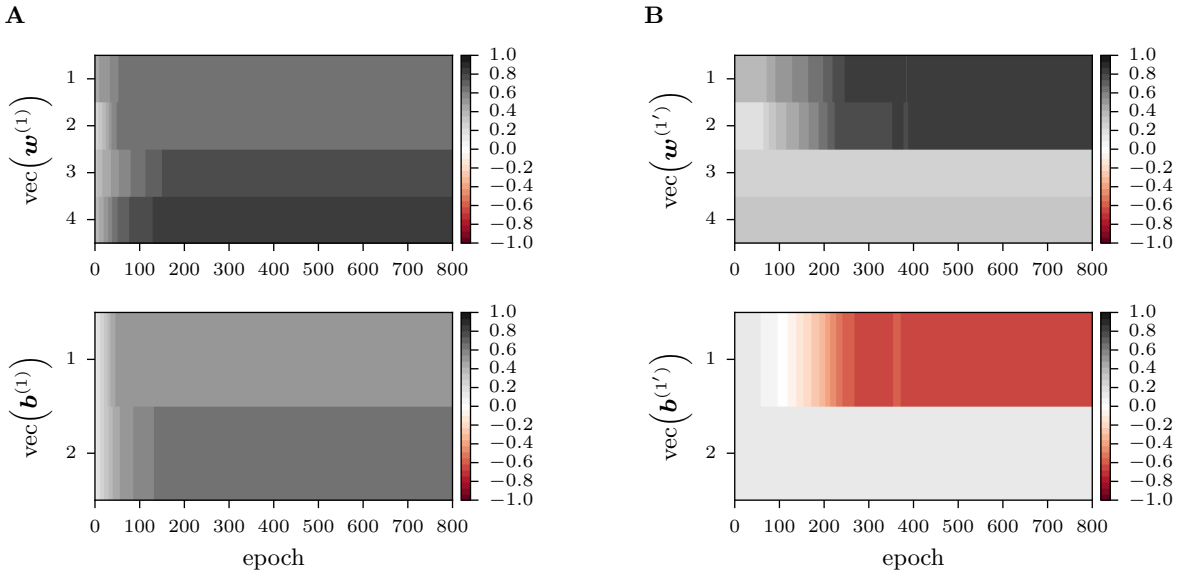

Figure 4.19: Evolution of the weight and bias values of the hidden layer as a function of the training epoch for one specific run. **A** Quantized weight values of connections from input neurons to excitatory hidden neurons (*top panel*) and quantized bias values between the bias neuron of the hidden layer and excitatory neurons in the hidden layer. **B** Quantized weight values of connections from input neurons to inhibitory hidden neurons (*top panel*) and quantized bias values between the bias neuron of the hidden layer and inhibitory neurons in the hidden layer.

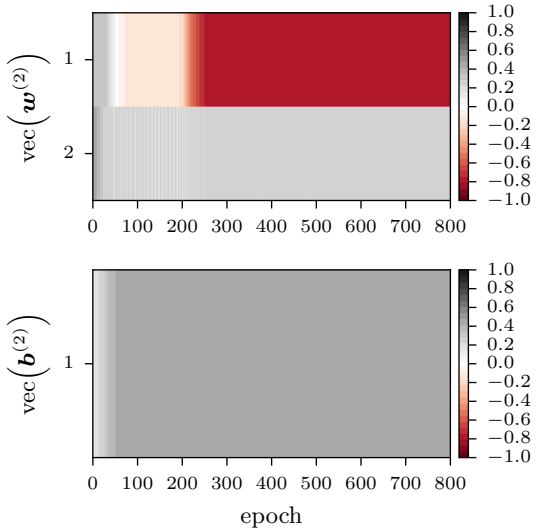

Figure 4.20: Quantized weight (*top panel*) and bias values (*bottom panel*) of the output layer as a function of the training epoch for one specific run.

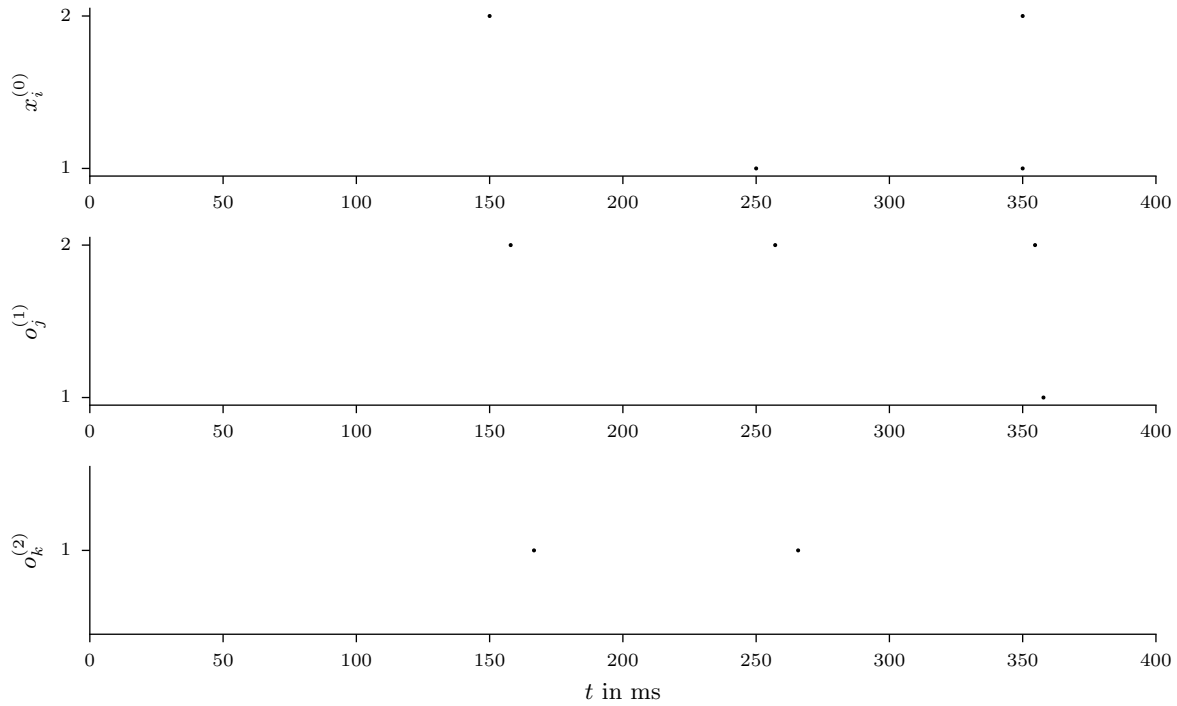

Figure 4.21: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the hidden layer's neurons $y_j^{(1)}$, and the output neuron $y_1^{(2)}$ in the test run. The activity of the inhibitory neurons is not show.

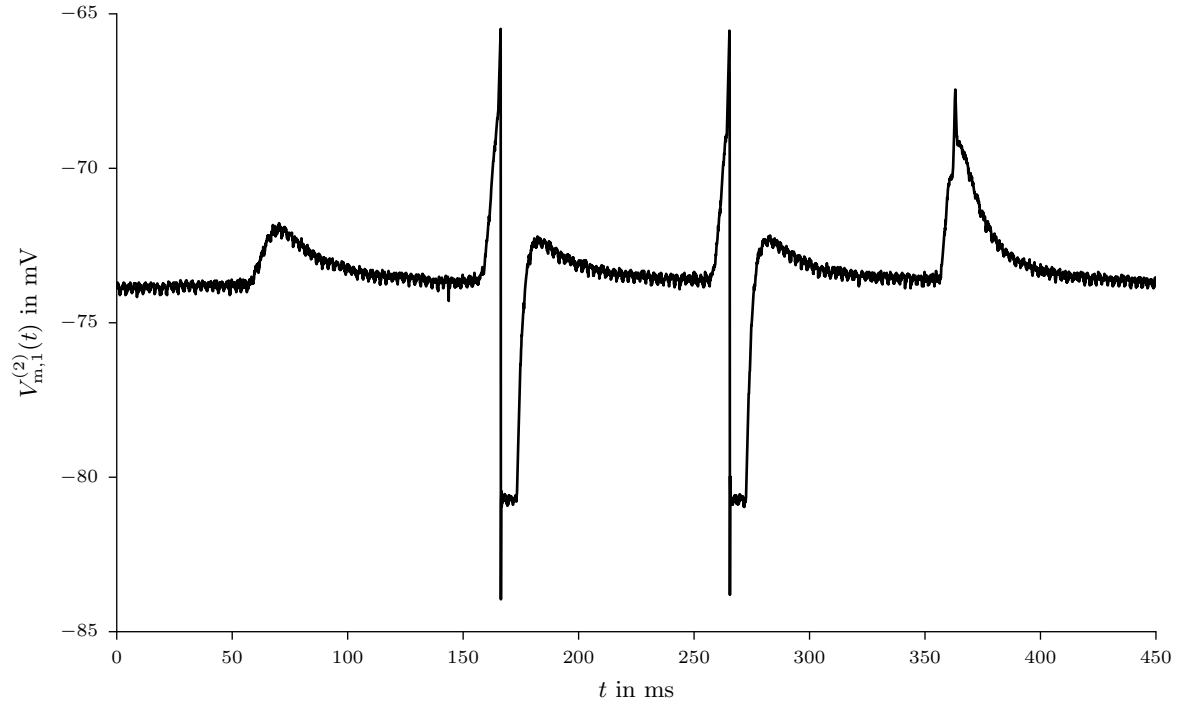

Figure 4.22: Membrane potential $V_{\mathrm{m}}(t)$ of neuron $y_1^{(2)}$ of the output layer during the test run.

4.3 MNIST Hand Written Digits Classification Task

The classification task presented her is base on the MNIST data set [50]. The data set consists of a training set of 60000 images and a test set of 10000 images of the handwritten digits 0 to 9. The resolution of these gray-level images is 28×28. The images used in this task are generated from the original images by extracting the 20×20 pixel box which contains the "actual" image. The resolution is then reduced to 8×8 pixels by bi-cubic interpolation. Finally, a threshold is applied in order to convert the gray-level images into binary images. The threshold is calculated individually for each image by utilizing Otsu's method [51]. In short, this method computes a threshold value such that the intra-class variance (foreground pixels and background pixels) is minimal. The resulting image $\boldsymbol{I}_i$ is then used as network input $\boldsymbol{x}_i = \mathrm{vec}(I_i)$. Examples of original MNIST images of a "0", "1", "4", "6", and "7" are shown in the top row of Fig. 4.24 and the bottom row of this figure shows their binarized version.

The topology of the feed-forward network which is utilized in this task is shown in Fig. 4.23. It consists of an input layer where the number of units is equal to the number of pixels of the input image, two hidden layers, and an output layer. The number of output neurons is equal to the number of classes the network is trained to classify.

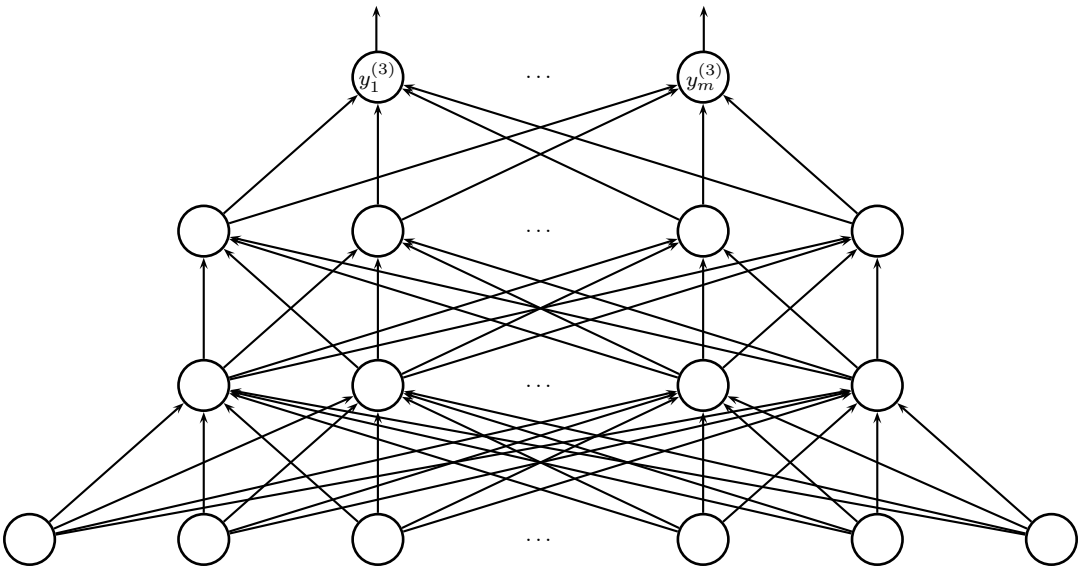

Figure 4.23: Topology of the feed-forward neural network with one input layer, two hidden layers and one label layer. The number of input units is equal to the number of pixels of the input image. The bias units associated to each of the hidden layers and to the output layer are not shown. The number of output neurons is equal to the number of classes the network is trained to identify.

The test accuracy in the NEST simulations on 5 classes is 96.3%. On *Spikey* only 3 classes are used. The accuracy on the test set is 91.5%. Training is done with mini-batches of certain size (for details see Section 4.3.1 and Section 4.3.2). A digit is considered as correctly classified if the corresponding output neuron emits at least one spike after the presentation of the image

while all other output neurons remain silent.

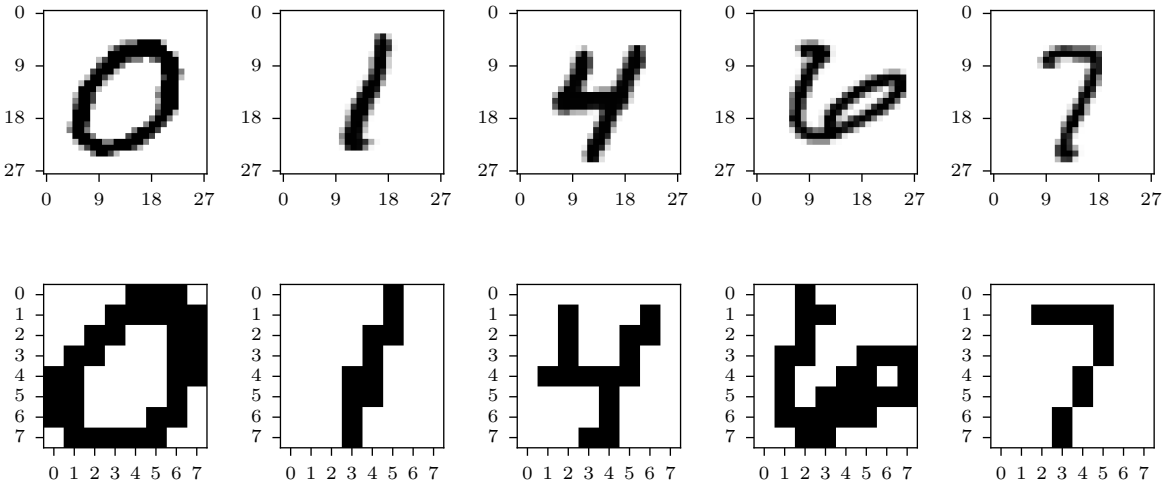

Figure 4.24: Examples of original MNIST images of a "0", "1", "4", "6", and "7" (*top row*) and the images which are used as network input (*bottom row*). The images used as input data are generated from the original images by extracting the inner 20×20 pixel box, reducing the resolution to 8×8 pixels by bi-cubic interpolation, and finally applying a threshold in order to convert the gray-level images to binary images. The threshold value is calculated individually for each image by utilizing Otsu's method.

4.3.1 Results of the MNIST Hand Written Digits Classification Task—NEST

In the following, the NEST simulation results of the MNIST task are shown. Training is one on a modified subset (as described above) of the MNIST data set. All images from 5 classes ("0", "1", "4", "6", and "7") are used. This results in a training set of 28062 images and a test set of 5083 images. Training is performed for 1370 steps with mini-batches of size 1024. The individual training examples are presented to the network every 100 ms starting at t = 50 ms. The initial weights are drawn from a Gaussian distribution with a mean of 0.0 and a standard deviation of 0.1. The bias values are uniformly initialized to 0.1.

The used network is has two hidden layers with 25 units each. The final test accuracy is 96.3% with a standard deviation of 0.56%.

Figure 4.25 shows the evolution of the classification error as a function of the training steps. Shown is the mean training error over 10 independent runs. Examples of correctly and wrongly classified digits are shown in Fig. 4.26. The top rows show their original version.

The spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the neurons $y_j^{(1)}$ of the first hidden layer, the neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's neurons $y_l^{(3)}$ for 10 digits in a specific test run is shown in Fig. 4.27. Figure 4.28 shows the activity of the neurons for 100 digits in the same test run. Spike events of an output neuron corresponding to a wrong class are colored in red. As can be seen from Fig. 4.27 the output neurons emit in the most cases only one spike for each class.

A histogram of the quantized weight and bias values after training is shown for the first hidden layer, the second hidden layer, and the output layer in Fig. 4.29, Fig. 4.30, and Fig. 4.31, respectively.

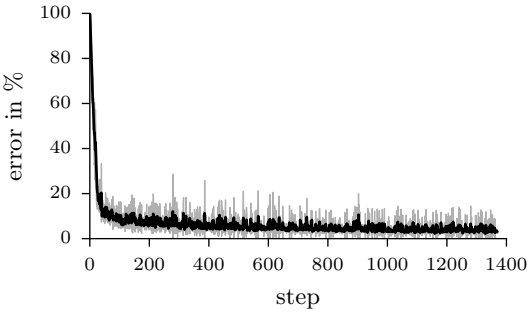

Figure 4.25: Classification error per batch as a function of the training step. Training is done for 1370 steps with mini-batches of size 1024. The error is defined as the fraction of examples (in one batch) for which the output of the network does not agree with the target classification. Shown is the mean error (*black line*) ±1 standard deviation (*shaded area*) over 10 independent runs.

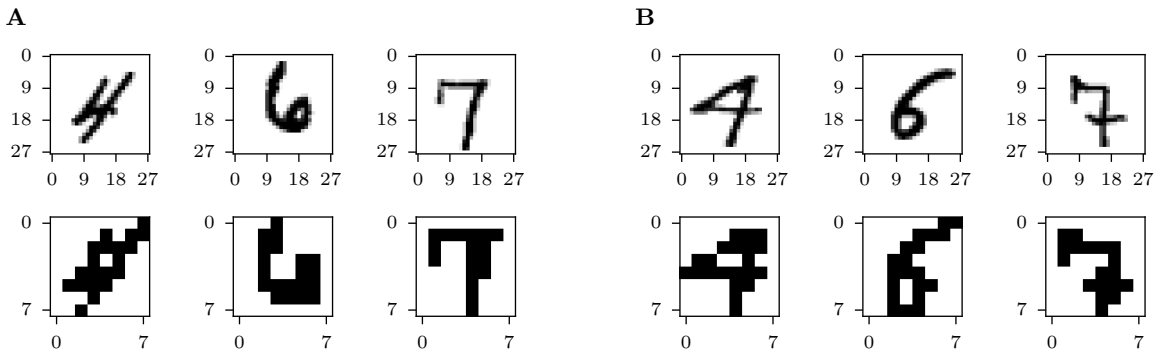

Figure 4.26: Examples of correctly and wrongly classified digits in the test run. The top row shows the original MNIST images and the bottom row shows their binarized version which are used as network input. **A** Three examples of correctly classified digits. **B** Three examples of wrongly classified digits.

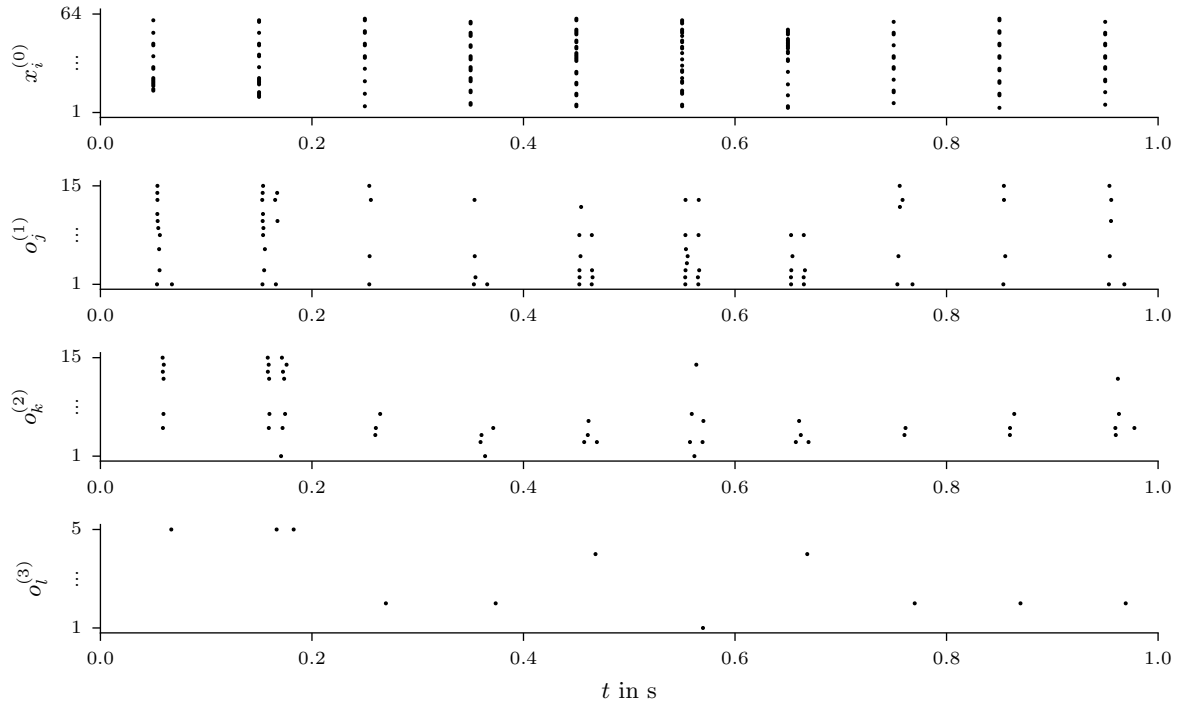

Figure 4.27: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the neurons $y_j^{(1)}$ of the first hidden layer, the neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's neurons $y_l^{(3)}$ for 10 digits in the test run. A digit is considered as correctly classified if the corresponding output neuron emits at least one spike after the presentation of the image while all other output neurons remain silent.

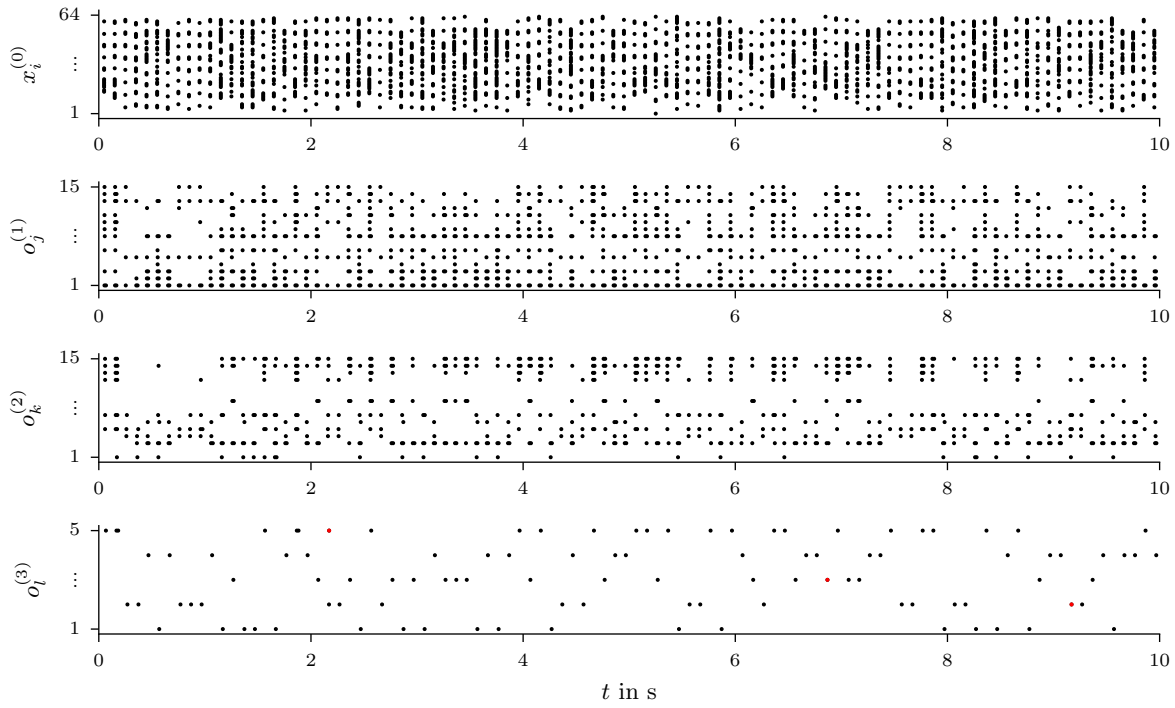

Figure 4.28: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the neurons $y_j^{(1)}$ of the first hidden layer, the neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's neurons $y_l^{(3)}$ for 100 digits in the test run. A digit is considered as correctly classified if the corresponding output neuron emits at least one spike after the presentation of the image while all other output neurons remain silent. Spike events of an output neuron corresponding to a wrong class are colored in red.

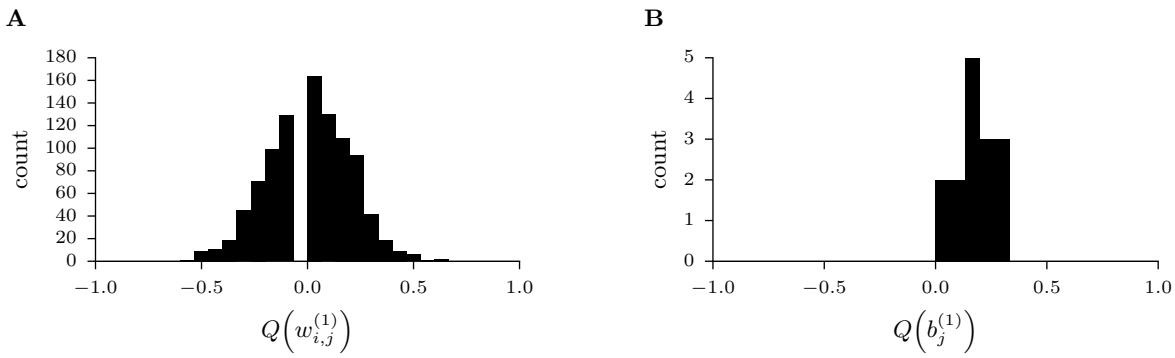

Figure 4.29: Histogram of the quantized weight and bias values of the first hidden layer after training. **A** Histogram of the quantized weighs. **B** Histogram of the quantized bias values.

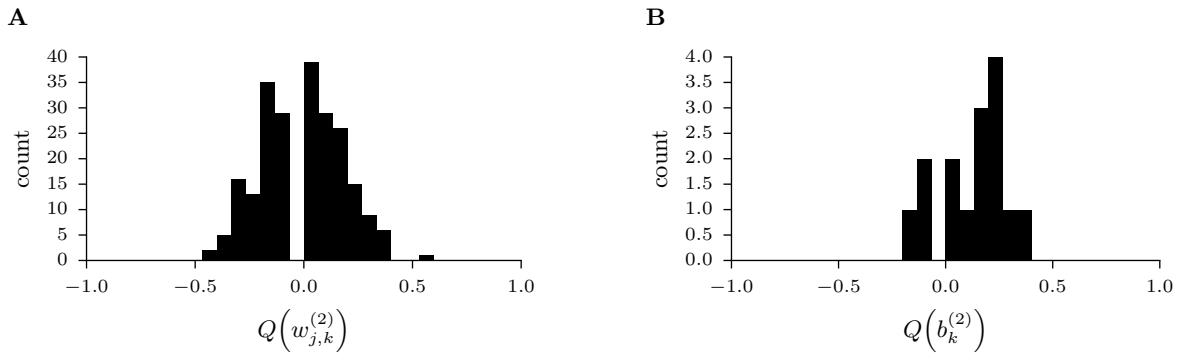

Figure 4.30: Histogram of the quantized weight and bias values of the second hidden layer after training. **A** Histogram of the quantized weighs. **B** Histogram of the quantized bias values.

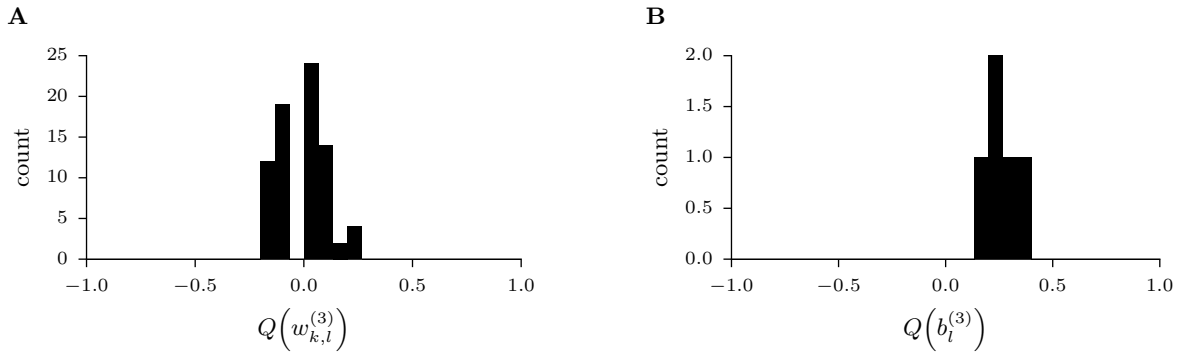

Figure 4.31: Histogram of the quantized weight and bias values of the output layer after training. **A** Histogram of the quantized weighs. **B** Histogram of the quantized bias values.

4.3.2 Results of the MNIST Hand Written Digits Classification Task—Spikey

In the following, the *Spikey* emulation results of the MNIST task are shown. Training is done on a modified subset (as described above) of the MNIST data set. All images from 3 classes ("0", "1", "4") are used. This results in a training set of 16930 images and a test set of 3079 images. Training is performed for 131 steps with mini-batches of size 1024. The individual training examples are presented to the network every 100 ms starting at t = 50 ms. The initial weights are drawn from a Gaussian distribution with a mean of 0.0 and a standard deviation of 0.1. The bias values are uniformly initialized to 0.1.

The used network is has two hidden layers with 25 units each. The final test accuracy is 91.5% with a standard deviation of 3.24%.

Figure 4.32 shows the evolution of the classification error as a function of the training steps. Shown is the mean training error over 10 independent runs. Examples of correctly and wrongly classified digits are shown in Fig. 4.33. The top rows show their original version.

The spike raster plot of the neural activity of the input layer's excitatory neurons $x_i^{(0)}$, the excitatory neurons $y_j^{(1)}$ of the first hidden layer, the excitatory neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's excitatory neurons $y_l^{(3)}$ for 10 digits in a specific test run is shown in Fig. 4.34. Figure 4.35 shows the activity of the neurons for 100 digits in the same test run. Spike events of an output neuron corresponding to a wrong class are colored in red. As can be seen from Fig. 4.34 the output neurons emit in the most cases only one spike for each class.

A histogram of the quantized weight and bias values after training is shown in Fig. 4.36 to Fig. 4.38.

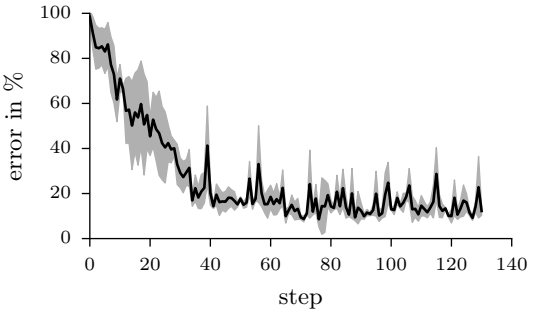

Figure 4.32: Classification error per batch as a function of the training step. Training is done for 131 steps with mini-batches of size 1024. The error is defined as the fraction of examples (in one batch) for which the output of the network does not agree with the target classification. Shown is the mean error (*black line*) ±1 standard deviation (*shaded area*) over 10 independent runs.

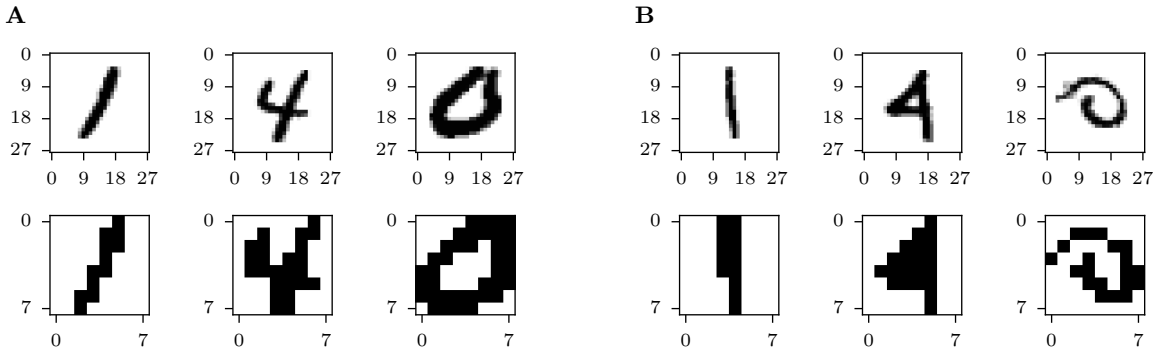

Figure 4.33: Examples of correctly and wrongly classified digits in the test run. The top row shows the original MNIST images and the bottom row shows their binarized version which are used as network input. **A** Three examples of correctly classified digits. **B** Three examples of wrongly classified digits.

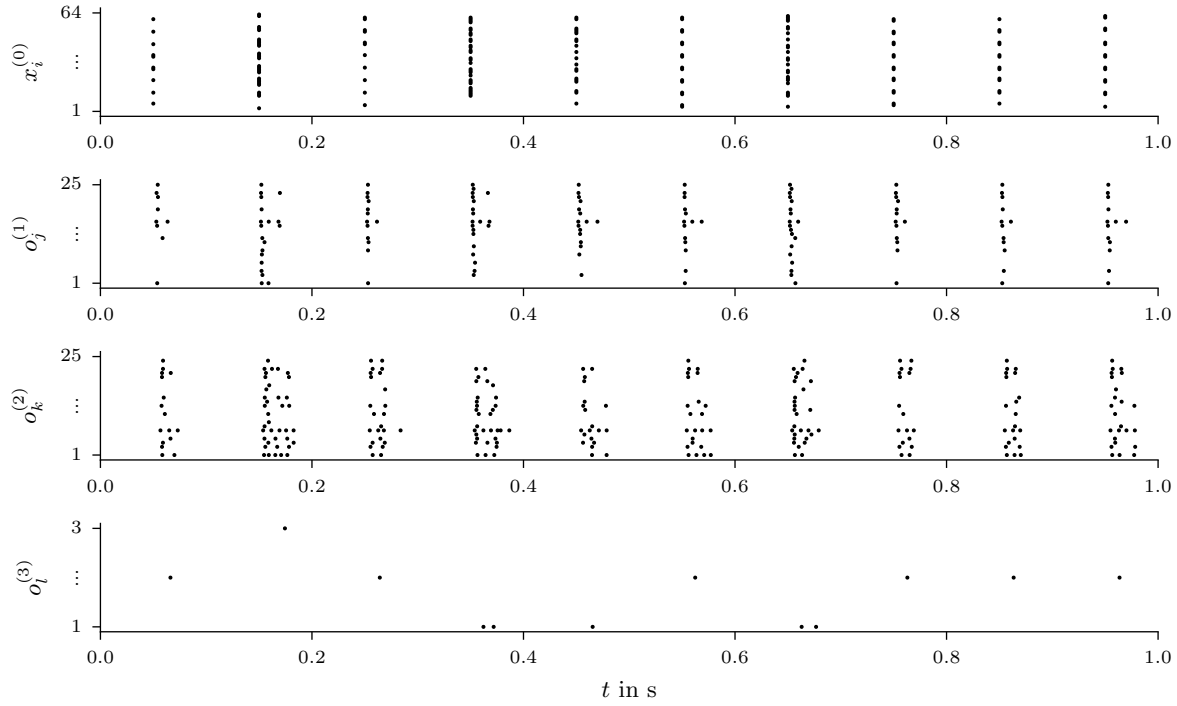

Figure 4.34: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the neurons $y_j^{(1)}$ of the first hidden layer, the neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's neurons $y_l^{(3)}$ for 10 digits in the test run. A digit is considered as correctly classified if the corresponding output neuron emits at least one spike after the presentation of the image while all other output neurons remain silent.

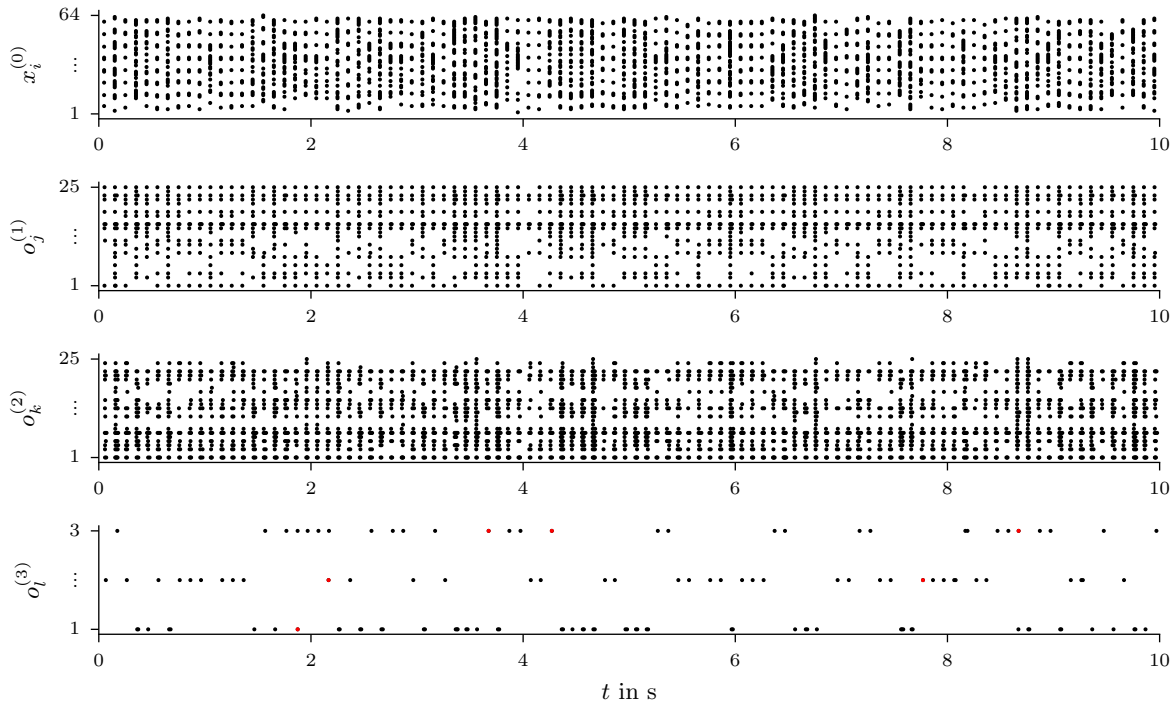

Figure 4.35: Spike raster plot of the neural activity of the input layer's neurons $x_i^{(0)}$, the neurons $y_j^{(1)}$ of the first hidden layer, the neurons $y_k^{(2)}$ of the second hidden layer, and the output layer's neurons $y_l^{(3)}$ for 100 digits in the test run. A digit is considered as correctly classified if the corresponding output neuron emits at least one spike after the presentation of the image while all other output neurons remain silent. Spike events of an output neuron corresponding to a wrong class are colored in red.

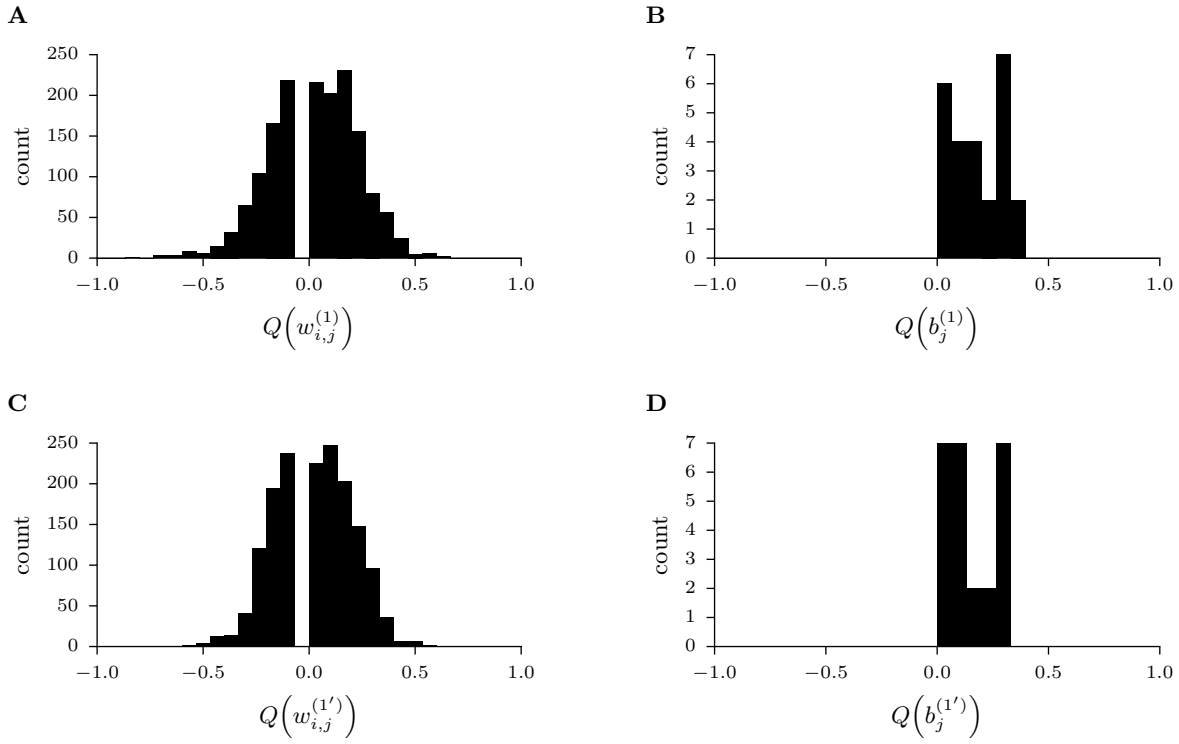

Figure 4.36: Histogram of the quantized weight and bias values of the first hidden layer after training. **A** Quantized weight values of connections between input neurons and excitatory neurons of the first hidden layer. **B** Quantized bias values between the bias neuron of the first hidden layer to excitatory neurons of the first hidden layer. **C** Quantized weight values of connections between input neurons and inhibitory neurons of the first hidden layer. **D** Quantized bias values between the bias neuron of the first hidden layer to inhibitory neurons of the first hidden layer.

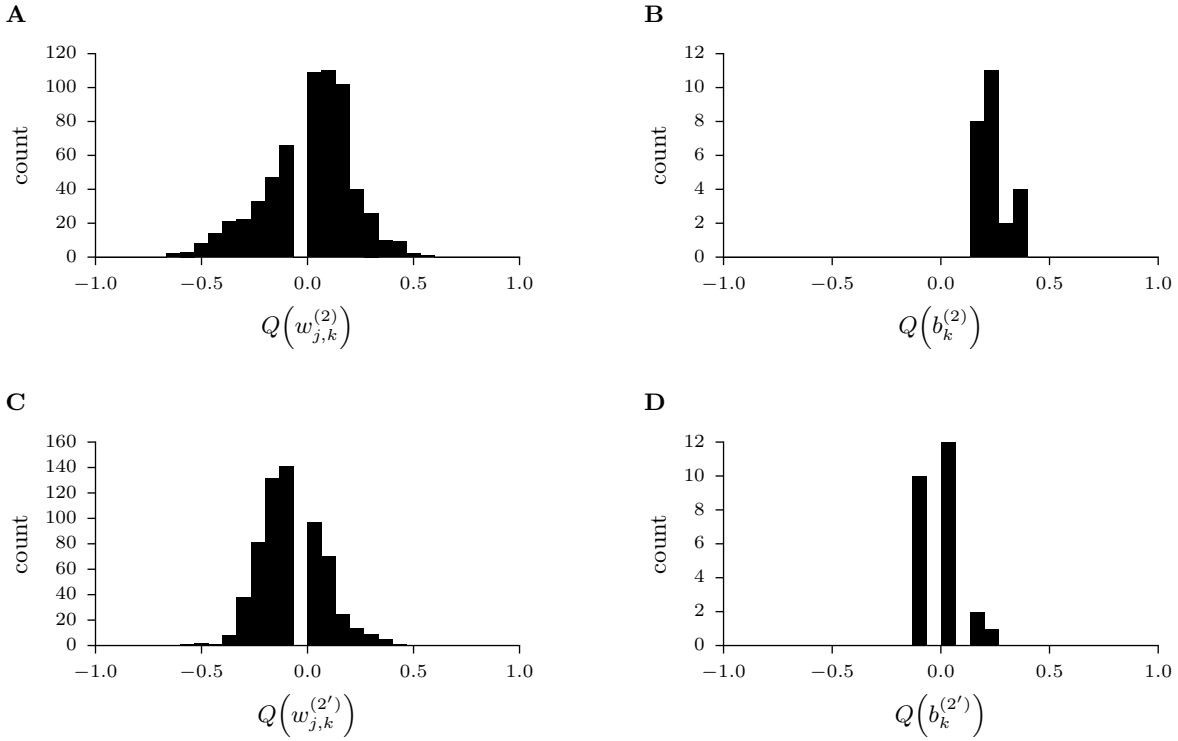

Figure 4.37: Histogram of the quantized weight and bias values of the second hidden layer after training. **A** Quantized weight values of connections between neurons of the first hidden layer and excitatory neurons of the second hidden layer. **B** Quantized bias values between the bias neuron of the second hidden layer to excitatory neurons of the second hidden layer. **C** Quantized weight values of connections between neurons of the first hidden layer and inhibitory neurons of the second hidden layer. **D** Quantized bias values between the bias neuron of the second hidden layer to inhibitory neurons of the second hidden layer.

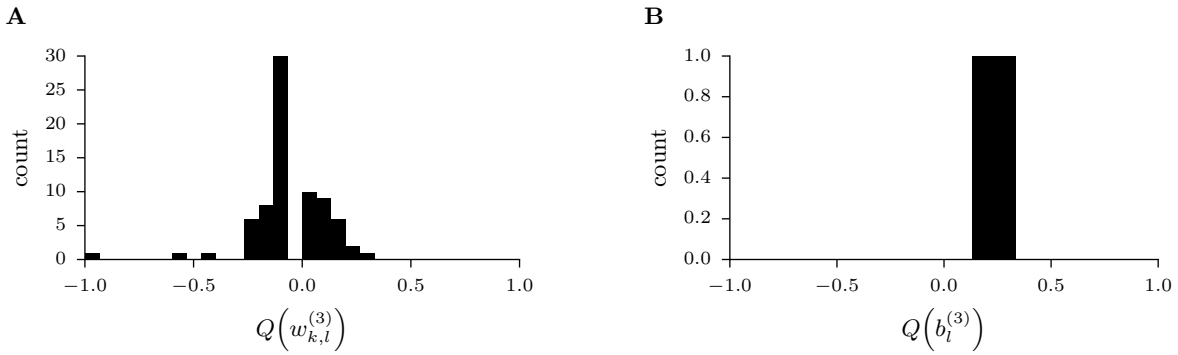

Figure 4.38: Histogram of the quantized weight and bias values of the output layer after training. **A** Histogram of the quantized weighs. **B** Histogram of the quantized bias values.

5 Discussion

It has been shown that a feed-forward spiking neural network can be successfully trained by backpropagation in conjunction with a rather coarse software model of the network. The backpropagation algorithm which uses gradient based optimization requires differentiable activation functions. Despite the nondifferentiability of the artificial neuron's activation function, that is the unit step function, the backpropagation algorithm could be successfully applied. This was possible by using an approximate derivative of the activation function.

The spiking neural networks in question were emulated on an accelerated analog neuromorphic hardware chip—The *Spikey* neuromorphic chip. While action potentials on this chip are transmitted by using digital communication, neurons and synapses are implemented in hardware. This analog approach allows the design of highly accelerated chips.

The network can only fully benefit from the hardware acceleration factor when classification is performed. It was found that in the training phase the limiting factor is the initialization of the chip. However, since the input data was not rate-coded but its binary values where directly translated into single spikes of the input neurons, the time between the presentation of the individual examples can be chosen quiet low. The only limiting factor is that one has to allow the membrane potential to decay before the presentation of the next input. The chosen delay of 100 ms would allow, by considering the hardware acceleration factor of about 10^4, the classification of 100000 MNIST digits within 1 s.

Analog hardware is inherently subject to noise, distortions in neuron and synapse dynamics need to be taken into account. These quite high variations could not successfully be compensated by the utilized learning algorithm. This might be due to the limited resolution in synaptic weight. Also the trial-to-trial variability is quiet a big issue as seen in the XOR task.

Since the activation function used in the artificial neurons allows only binary output values the mapping of temporal effects is not possible. If and how this is compensated by the learning algorithm would require further analysis. To address the issue of the rather high variability in neuron and synapse parameter one could use a rate-coded approach. This would make the model more robust for "dead" neurons. Another possibility could be to perform population averaging. However, since the size of the networks considered in this work is already at the capacity limit of the *Spikey* chip this is not practicable. When performing the averaging over multiple runs one could not take full advantage of the hardware acceleration which is also not preferable.

The obtained results in the MNIST task are quiet far away from state-of-the-art results. How-

ever, at least in the NEST simulations it was shown that the abstract model of the spiking neural network in combination with the utilized learning algorithm is functioning. The performance could possible be further increased. One option would be to simply increase the network size. Fine tuning of certain learning parameters could also be taken into account.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] W. Maass, "Fast sigmoidal networks via spiking neurons", *Neural Computation*, vol. 9, no. 2, pp. 279–304, 1997.

[3] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface", *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[4] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing", *Proceedings of the National Academy of Sciences*, p. 201 604 850, 2016.

[5] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations", in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[6] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[7] W. Maass, "Networks of spiking neurons: The third generation of neural network models", *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[8] W. Maass, G. Schnitger, and E. D. Sontag, "On the computational power of sigmoid versus boolean threshold circuits", in *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, IEEE, 1991, pp. 767–776.

[9] B. DasGupta and G. Schnitger, "The power of approximating: A comparison of activation functions", in *NIPS*, 1992, pp. 615–622.

[10] D. Ferster and N. Spruston, "Cracking the neuronal code", *Science*, vol. 270, no. 5237, p. 756, 1995.

[11] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing", *Neural networks*, vol. 14, no. 6, pp. 715–725, 2001.

[12] E. Kandel, J. Schwartz, and T. Jessell, *Essentials of Neural Science and Behavior*. McGraw-Hill, 1995, ISBN: 9780838522455. [Online]. Available: `https://books.google.at/books?id=vAY0v7F4JYEC`.

[13] W. Maass and H. Markram, "Synapses as dynamic memory buffers", *Neural Networks*, vol. 15, no. 2, pp. 155–161, 2002.

[14] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[15] R. B. Stein, "Some models of neuronal variability", *Biophysical journal*, vol. 7, no. 1, pp. 37–68, 1967.

[16] M. Shelley, D. McLaughlin, R. Shapley, and J. Wielaard, "States of high conductance in a large-scale model of the visual cortex", *Journal of computational neuroscience*, vol. 13, no. 2, pp. 93–109, 2002.

[17] A. Kumar, S. Schrader, A. Aertsen, and S. Rotter, "The high-conductance state of cortical networks", *Neural computation*, vol. 20, no. 1, pp. 1–43, 2008.

[18] W. Foster, L. Ungar, and J. Schwaber, "Significance of conductances in hodgkin-huxley models", *Journal of Neurophysiology*, vol. 70, no. 6, pp. 2502–2518, 1993.

[19] D. O. Hebb, *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[20] S. Lowel and W. Singer, "Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity", *Science*, vol. 255, no. 5041, p. 209, 1992.

[21] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type", *Journal of neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[22] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity", *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[23] W. M. Kistler and J. L. Van Hemmen, "Modeling synaptic plasticity in conjunction with the timing of pre-and postsynaptic action potentials", *Neural Computation*, vol. 12, no. 2, pp. 385–405, 2000.

[24] C. F. Stevens and Y. Wang, "Facilitation and depression at single central synapses", *Neuron*, vol. 14, no. 4, pp. 795–802, 1995.

[25] H. Markram and M. Tsodyks, "Redistribution of synaptic efficacy between neocortical pyramidal neurons", *Nature*, vol. 382, no. 6594, p. 807, 1996.

[26] R. S. Zucker and W. G. Regehr, "Short-term synaptic plasticity", *Annual review of physiology*, vol. 64, no. 1, pp. 355–405, 2002.

[27] H. Markram, Y. Wang, and M. Tsodyks, "Differential signaling via the same axon of neocortical pyramidal neurons", *Proceedings of the National Academy of Sciences*, vol. 95, no. 9, pp. 5323–5328, 1998.

[28] L. F. Abbott, J. Varela, K. Sen, and S. Nelson, "Synaptic depression and cortical gain control", *Science*, vol. 275, no. 5297, pp. 221–224, 1997.

[29] M. Tsodyks, K. Pawelzik, and H. Markram, "Neural networks with dynamic synapses", *Neural computation*, vol. 10, no. 4, pp. 821–835, 1998.

[30] R. P. Costa, P. J. Sjöström, and M. C. Van Rossum, "Probabilistic inference of short-term synaptic plasticity in neocortical microcircuits", 2013.

[31] T. Pfeil, A.-C. Scherzer, J. Schemmel, and K. Meier, "Neuromorphic learning towards nano second precision", in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, 2013, pp. 1–5.

[32] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, *et al.*, "Neuromorphic silicon neuron circuits", *Frontiers in neuroscience*, vol. 5, p. 73, 2011.

[33] J. Schemmel, D. Bruderle, K. Meier, and B. Ostendorf, "Modeling synaptic plasticity within networks of highly accelerated I&F neurons", in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, IEEE, 2007, pp. 3367–3370.

[34] J. Schemmel, A. Grubl, K. Meier, and E. Mueller, "Implementing synaptic plasticity in a VLSI spiking neural network model", in *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, IEEE, 2006, pp. 1–6.

[35] A. Grübl, "VLSI implementation of a spiking neural network", PhD thesis, 2007.

[36] D. Brüderle, "Neuroscientific modeling with a mixed-signal VLSI hardware system", PhD thesis, 2009.

[37] M. V. Tsodyks and H. Markram, "The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability", *Proceedings of the National Academy of Sciences*, vol. 94, no. 2, pp. 719–723, 1997.

[38] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier, "Six networks on a universal neuromorphic computing substrate", *Frontiers in Neuroscience*, vol. 7, p. 11, 2013.

[39] W. J. Dally and J. W. Poulton, *Digital systems engineering. 1998.*

[40] J. Bill, "Self-stabilizing network architectures on a neuromorphic hardware system", Master's thesis, University of Heidelberg, HD-KIP-08-44, 2008.

[41] E. Müller, "Operation of an imperfect neuromorphic hardware device", PhD thesis, University of Heidelberg, HD-KIP-08-43, 2008.

[42] A. Davison, D. Brüderle, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "PyNN: A common interface for neuronal network simulators", 2009.

[43] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)", *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[44] D. Pecevski, T. Natschläger, and K. Schuch, "Pcsim: A parallel simulation environment for neural circuits fully integrated with Python", 2009.

[45] M. L. Hines and N. T. Carnevale, "The NEURON simulation environment", *Neural computation*, vol. 9, no. 6, pp. 1179–1209, 1997.

[46] D. F. Goodman and R. Brette, "The brian simulator", *Frontiers in neuroscience*, vol. 3, p. 26, 2009.

[47] D. Brüderle, A. Grübl, K. Meier, E. Mueller, and J. Schemmel, "A software framework for tuning the dynamics of neuromorphic silicon towards biology", in *International Work-Conference on Artificial Neural Networks*, Springer, 2007, pp. 479–486.

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

[49] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", *ArXiv preprint arXiv:1412.6980*, 2014.

[50] Y. LeCun, C. Cortes, and C. J. Burges, *The mnist database of handwritten digits*, 1998.

[51] N. Otsu, "A threshold selection method from gray-level histograms", *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

Acknowledgments

I want to thank myself for being so wonderful... no, that's not appropriate. At first I like to thank my supervisor Prof. Robert Legenstein for making this thesis possible and for his great support. I also like to thank the whole staff of the Institute for Theoretical Computer Science for the warm reception. A special thanks goes also to Dr. Johannes Bill and Dr. Eric Müller for giving me access to the *Spikey* system and for doing their best in giving support for it. My family and friends also deserve a big "thank you" for distracting me from work in a good way.

Last but not least, I would like to thank the coffee machine from the institute for providing me with tasty coffee (not the bad coffee machine, which requires—in my opinion—too much user intervention).