Department of Physics and Astronomy
Heidelberg University

Master Thesis
in Physics
submitted by
Alexander Kugele
born in Waiblingen

February 2018

Solving the Constraint Satisfaction Problem Sudoku on Neuromorphic Hardware

This Master thesis has been carried out by Alexander Kugele
at the

Kirchhoff Institute for Physics
Heidelberg University

under the supervision of

Prof. Dr. Karlheinz Meier

Solving the Constraint Satisfaction Problem Sudoku on Neuromorphic Hardware

Artificial Neural Networks are nowadays widely used, e.g. to solve problems in the fields of image recognition and decision-making. Spiking Neural Networks, which mimic the functional principle of networks in the brain more closely, are often not yet competitive. However, application-specific circuits like the HICANN chips of the BrainScaleS system are able to emulate such Spiking Neural Network dynamics in a fast and efficient way. In this thesis a method to solve the Constraint Satisfaction Problem sudoku is implemented on this neuromorphic hardware system. The performance is analyzed and compared to simulation. The network itself is a stochastic solver with fixed weights for all synapses. A training algorithm is devised and implemented, increasing the performance of the network for arbitrary sudokus. This technique can be generalized to other CSPs.
To conduct experiments using multiple HICANN chips, the existing calibration framework has been extended to efficiently calibrate full wafer systems.
The training together with the increased number of available calibrated HICANNs denote an important step to large-scale neural network emulations on the BrainScaleS system.

Lösen des Bedingungserfüllungsproblems Sudoku auf neuromorpher Hardware

Künstliche neuronale Netze werden heutzutage oft verwendet, zum Beispiel in den Fachrichtungen Bilderkennung und Entscheidungsfindung. Gepulste neuronale Netze, die das Funktionsprinzip des Gehirns nachahmen, sind in vielen Bereichen noch nicht konkurrenzfähig zu künstlichen neuronalen Netzen. Anwendungsspezifische Schaltungen wie die HICANN Chips des BrainScaleS System emulieren gepulste neuronale Netze schnell und effizient. In dieser Arbeit wird eine Methode realisiert, um das Bedingungserfüllungsproblem Sudoku auf diesem neuromorphen Hardwaresystem zu lösen. Die Leistung des Systems wird analysiert und mit Software-Simulationen verglichen. Das Netzwerk selbst ist ein stochastischer Löser mit festen Gewichten für alle Synapsen. Ein Trainingsalgorithmus wird entwickelt und umgesetzt, der die Leistung des Netzwerks für beliebige Sudokus erhöht. Diese Methode kann auf andere Bedingungserfüllungsprobleme verallgemeinert werden.
Um Experimente auf mehreren HICANN Chips durchzuführen, wurde das vorhandene Kalibrations-Framework erweitert, um ganze Wafersysteme effizient zu kalibrieren.
Das Training zusammen mit der gesteigerten Anzahl an kalibrierten HICANNs stellen einen wichtigen Schritt in Richtung großskaliger neuronaler Netzwerkemulationen auf dem BrainScaleS System dar.

Contents

1. Introduction

Decision-making based on experience and object recognition are just two of many tasks even state-of-the-art algorithms struggle to solve. The human brain excels at these tasks, however the mechanisms leading from networks of spiking neurons to the formation of thoughts and memories are still to a great extent unknown. Nowadays, many different approaches are used in the thriving field of machine learning to tackle these tasks, whith artificial neural networks being the most promising ones (*Du*, 2010; *Hagan and Menhaj*, 1994; *Hornik et al.*, 1989). These networks mimic the neural networks of the brain only in the fact that information can be sent between neurons. The connections, called synapses, weigh their input, before sending it to other neurons. These weights determine the networks ability to accomplish a specific task. It is often necessary to train the network with enormous amounts of data, which is time and energy consuming. Thus recently, multiple big semiconductor companies launched custom application-specific integrated circuits (ASICs) to accelerate and streamline inference, e.g. Intel Movidius (*Intel*, 2018) and learning, e.g. Google's Tensor Processing Unit (*Jouppi et al.*, 2017). Still, these systems lack the general-purpose functionality of the brain due to the fact that they for now can only be trained to solve at most a narrow set of tasks. Furthermore, they only loosely mimic the way neural network in the brain works. It is doubtful that conclusions about the working principles of the brain can be drawn from designing these artificial neural network chips. But insights about brain processes are required to develop treatments for brain disorders like Alzheimers or Parkinson's diseases, and have an impact on the treatment of psychological disorders. Hence, a better understanding of the human brain is beneficial for many branches of research in industry and university.

The Human Brain Project, a Flagship of the European Union (*FET*, 2018), brings together researchers from areas like neurology, biology, psychology and electronics to collaboratively extend the knowledge about the human brain. One sub-department is the SP9 Neuromorphic Computing Platform. It develops two large-scale neuromorphic machines: SpiNNaker (*Furber et al.*, 2014) and the BrainScaleS system (*Schemmel et al.*, 2010). The latter is designed and built in a collaboration between the Electronic Vision(s) group Heidelberg and the TU Dresden. It features about four million neurons and 880 million synapses on 7680 identical and interconnected

custom ASICs, called HICANNs. Each neuron is implemented as an analog circuit that models the mathematical Adaptive exponential integrate-and-fire model, which in turn enables to model aspects of biological neural networks. This physical model system with its speed-up factor of about 10,000 compared to real biological time, it allows to study temporal changes in neural networks that are very difficult to observe in a laboratory. However, in contrast to the mathematical model, the hardware neurons are inherently different due to the finite manufacturing accuracy (*Koke*, 2017). To reduce this variability, calibration routines have to be developed and applied that ensure high quality simulations. Conducting small toy experiments in simulations with a neural simulator (*Goodman and Brette*, 2008; *Kunkel et al.*, 2017) and on the BrainScaleS system helps to identify and reduce differences between the two and is a first step for large-scale experiments.

This thesis aims to investigate a method to solve constraint satisfaction problems with neural networks on the BrainScaleS system. In the course of this task, the existing calibration is improved and extended to the whole wafer-system. The content of this thesis is divided into three chapters. In the first chapter, an overview of the BrainScaleS system is given. The second chapter covers means to improve the calibration of the refractory period on the HICANN chips in the BrainScaleS system and to scale the existing single-HICANN calibration up to a whole wafer. This work builds on the calibration routines developed in *Schmidt* (2014) and the findings in *Koke* (2017) and *Kleider* (2017). The third chapter describes the implementation of a network of LIF neurons which can solve the number-placement puzzle sudoku as an exemplary case of a constraint satisfaction problem. The theoretical description can be found in *Habenschuss et al.* (2013), an implementation on another VLSI system is in *Binas et al.* (2015). A special training of the network is implemented to use the weights of the synapses in the network to compensate for intrinsic variations of the hardware neurons. This novel type of in-the-loop training of neural networks was first used in *Schmitt et al.* (2017) to train a Spiking Neural Network to recognize MNIST digits. The approach in this thesis builds on this training technique but instead of training the network with regular input as in a classical machine learning approach, a particular training algorithm is implemented to reduce the neuron-to-neuron variations.

2. The Neuromorphic System

This chapter describes the BrainScaleS system, a physical model system to emulate spiking neural networks. First, a single neuron is described in theory and on chip. Then, the HICANN chip is presented, featuring 512 neurons and a synapse array to set up neural networks. To emulate even larger networks, wafers are designed and as a whole integrated in a system. This procedure, its advantages and limitations are presented in sections 2.2.2 and 2.2.3.

2.1. Neuron Model

The following sections present the equations of the neuron model and their implementation on hardware.

2.1.1. Theory

The neurons of the BrainScaleS system are designed to model the differential equations of the Adaptive exponential integrate-and-fire model (*R et al.*, 2008) shown in eqs. (2.1) and (2.2) together with the adaptation rule after a spike in eq. (2.3) and the refractory mechanism in eq. (2.4).

$$C_{\mathrm{m}}\frac{dV}{dt} = -g_L(V - E_{\mathrm{rev}}) + g_L\Delta T\exp(\frac{V - V_{\mathrm{thresh}}}{\Delta_T}) - w + I \tag{2.1}$$

$$\tau_w\frac{dw}{dt} = a(V - E_{\mathrm{rev}}) - w \tag{2.2}$$

$$\text{if} \quad V(t^f) \geq V_{\mathrm{thresh}} \quad \text{then} \quad w \to w + b \tag{2.3}$$

$$\text{if} \quad V(t^f) \geq V_{\mathrm{thresh}} \quad \text{then} \quad V \to V_{\mathrm{reset}} \quad \text{until} \quad t \to t^f + \tau_{\mathrm{ref}} \tag{2.4}$$

The variables describe the membrane potential V, the adaption variable w, the input current I, the membrane capacitance C_{m}, the leak conductance g_L, the reset voltage V_{reset}, the leak reversal potential E_{rev}, the threshold voltage V_{thresh}, the slope factor ΔT, the adaption coupling parameter a, the adaption change b, time constant τ_w and the refractory period τ_{ref}. At t^f the neuron fires. The current I is increased for every spike the neuron receives, depending on an activation function. In the case of the HICANN chip, neurons receive an exponential activation as shown in eqs. (2.5)

and (2.6).

$$I_i(t) = w_{ij}(V(t) - E_{\text{syn}})\alpha(t - t_j) \tag{2.5}$$

$$\alpha(t - t_j) = -\exp\left(-\frac{t}{\tau_{\text{syn}}}\right)\Theta(t - t_j) \tag{2.6}$$

The parameter w_{ij} is the weight of the synapse between neuron i and neuron j, α is the activation function, $\Theta(t)$ is the Heaviside function, t_j is the time the spike arrives at neuron i and E_{syn} is called the reversal potential.

A subset of these equations, where the adaption variable and the slope factor are set to 0, is called the Leaky Integrate-and-Fire model (*Cessac and Vieville*, 2007). Writing the inhibitory and excitatory stimulus as separate parts, the final equation reads for a LIF neuron reads

$$\begin{aligned} C_{\text{m}}\frac{dV_i}{dt} = &- g_L(V - E_{\text{rev}}) \\ &+ (V_i - E_{\text{syn,exc}})\sum_j w_{ij}\sum_{t_j^f<t}\alpha_{\text{exc}}(t - t_j^f) \\ &- (V_i - E_{\text{syn,inh}})\sum_k w_{ik}\sum_{t_k^f<t}\alpha_{\text{inh}}(t - t_k^f) \end{aligned} \tag{2.7}$$

with the aforementioned refractory mechanism.

Figure 2.1 shows the behavior of a LIF neuron simulated with the neural network simulator NEST when changing a single parameter. The parameter names have been adapted to the naming in NEST, thus $E_{\text{syn,inh}} \mathrel{\hat{=}} E_{\text{rev,i}}$ and the same for the excitatory part as well as $E_{\text{rev}} \mathrel{\hat{=}} V_{\text{rest}}$.
In the following, the neuron parameters are explained panel by panel. Changes of the parameters in (A-F) result in a different shape of the post-synaptic potential (PSP). The reset potential V_{reset} (G) is the voltage to which the neuron is reset after it spiked. A lower value also results in a longer time period until the neuron is at rest again. Increasing the refractory time τ_{ref} (H) increases the period in which the neuron is clamped to V_{reset} and also increases the time until the neuron is at rest. Furthermore, it determines the maximum spiking frequency of the neuron as $\nu_{\text{max}} = \frac{1}{\tau_{\text{ref}}}$. Lowering the threshold V_{thresh} (I) induces spikes at lower input rates and thus decreases the dynamic range of the neuron trace.

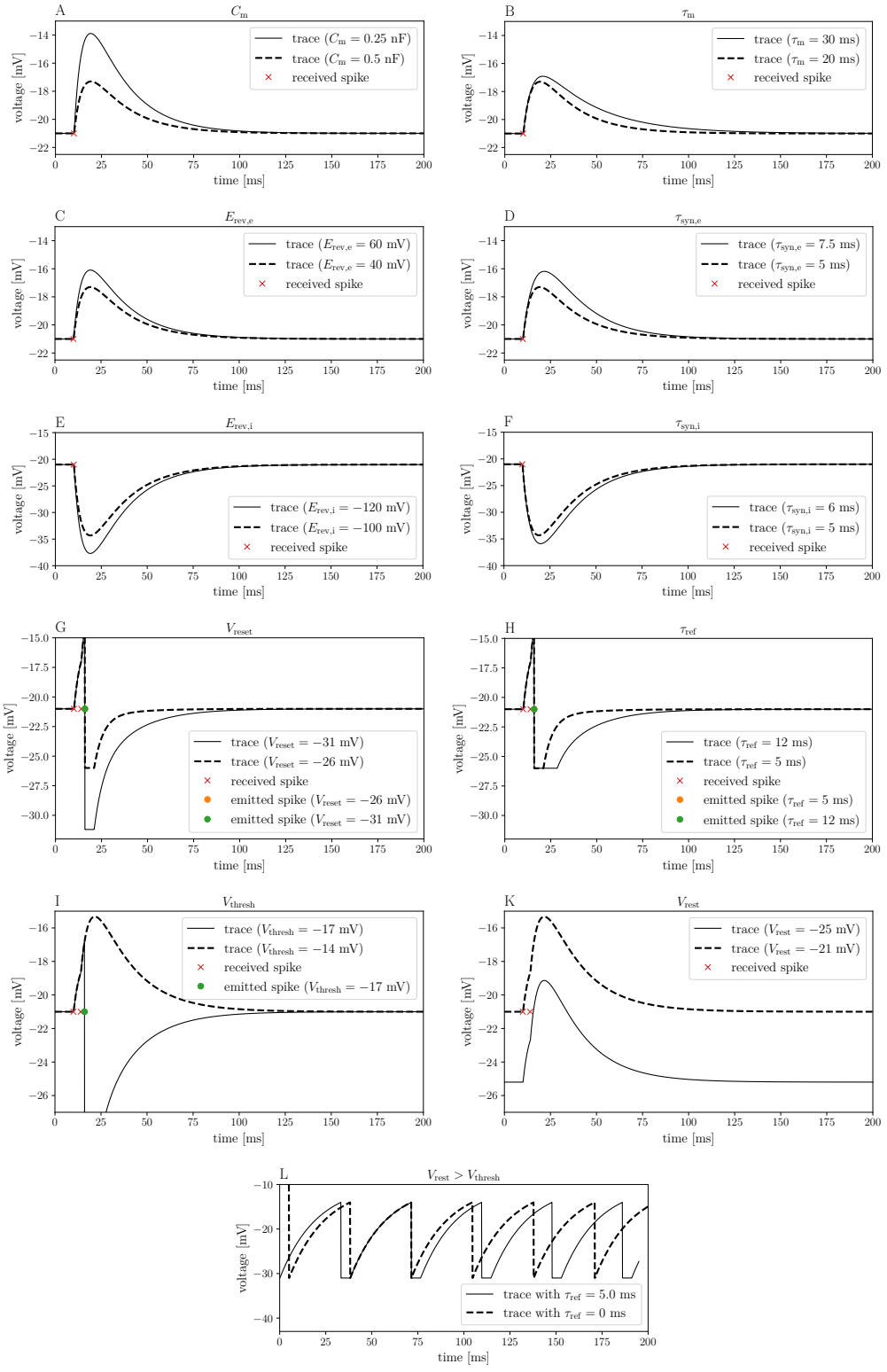

Figure 2.1.: **Relationship between parameter changes and trace behavior.** The plots are generated with NEST. Information about individual panels is given in section 2.1.1.

Changing V_{rest} (K) introduces an absolute shift. The last frame (L) shows the behavior when $V_{\text{rest}} > V_{\text{thresh}}$. The refractory period can be identified as the base line. The rise behavior does not change for different τ_{ref}.

The next section describes the neuron implementation in hardware and presents the BrainScaleS system.

2.1.2. Hardware

The basic idea to implement a neuron on hardware is to design an electric circuit that models the differential equation of the AdEx model in eq. (2.1). The advantage is that the network runs directly on the hardware instead of using the hardware only for computations of the differential equations as it is done in numerical simulations. In the case of the HICANN chip, a completely analog circuit emulates the AdEx model in eq. (2.1). A digital part routes the spikes between neurons. The components are only explained briefly here, for a detailed description see *Kleider* (2017); *Koke* (2017); *Millner* (2012). Essentially, each model parameter is represented in the neuron circuit by either a current or a voltage. A lookup table to translate parameter names to hardware names and their corresponding control parameter names is given in *Schmidt* (2014, pp. 10,13). For example, τ_{ref} is called τ_{refrac} and is controlled by the bias current I_{pl}. Each control parameter is set via a digital 10 bit floating gate value which results in a current from 0 - 2.5 µA or a voltage from 0 - 1.2 V. To be precise, the floating gates are analog, but the programming circuit configures them with 10 bit resolution. These hardware ranges have to be mapped to the digital 10 bit range, which can be either done by an ideal calibration calculated and simulated in *Schwartz* (2013), or via a calibration measurement. On the other hand, the time constants and voltages in hardware have to be translated to the biological domains of the neuron model. This translation has been defined in *Schmidt* (2014, pp. 11–12) to be

$$V_{\text{HW}}[\text{V}] = 2V_{\text{bio}}[\text{V}] + 1000 \tag{2.8}$$

$$\tau_{\text{HW}} = \frac{\tau_{\text{bio}}}{10000}. \tag{2.9}$$

For example, the threshold voltage is translated from $V_{\text{thresh,FG}} \rightarrow V_{\text{thresh,HW}} \rightarrow V_{\text{thresh,bio}}$ and the refractory time is translated as $I_{\text{pl,FG}} \rightarrow \tau_{\text{ref,HW}} \rightarrow \tau_{\text{ref,bio}}$.

As the hardware neurons are inherently different because of transistor mismatch due to a limited manufacturing accuracy, calibrating the neuron circuits is the preferred way to go, although the system can also be used uncalibrated, i.e. with the ideal transformations, if variations between neurons is not a problem for the network setup. Chapter 3 describes the refractory time calibration in more detail

and analyzes its current and possible accuracy. Calibration methods for the other parameters are described in *Kleider* (2017); *Schmidt* (2014).

In the next section the HICANN chip is described, which comprises the hardware neurons and synapses. Then the wafer-scale integration of this chip is explained and the current setup of 20 wafers, called the BrainScaleS system, is depicted.

2.2. Hardware Layers

The hardware layers are presented from the smallest unit, a single HICANN chip, to the whole wafer-scale system with 20 wafers.

2.2.1. Single Chip

To model neural networks, many freely configurable neurons are needed together with the possibility to define arbitrary network structures. The first aspect can be realized in hardware by designing the circuit of a single neuron and putting it on the chip multiple times. For a detailed description of the hardware design, see *Schemmel et al.* (2008, 2010). The articles describe the High Input Count Neuronal Network (HICANN) chip, which can be seen on the left of fig. 2.2. It features 512 AdEx neurons (see section 2.1, prominently placed in two rows in the middle of the chip. The floating gates in between the neurons store all parameter values, which can be a neuron parameter or some other parameter of the chip.

The neurons are wedged in by two synapse arrays, which allow to draw connections between neurons. A thorough explanation how the spike routing works can be found in *Jeltsch* (2014). The routing is summarized as follows: One spike is a 6 bit address event. Spikes from multiple neurons are first merged in a tree structure, called the merger tree. At the bottom of the tree, spike events are relayed into the Layer1 network, the network responsible to route spikes from one neuron to another. Each spike in the network is routed from the aforementioned bottom merger to a horizontal bus, then to a sparse crossbar switch which in turn is connected to a vertical bus. This bus then is connected to another crossbar switch which routes the spike to one of the 224 synapse drivers. These are connected to specific synapses which eventually are connected to a synaptic input of a specific neuron. Thus, in total 224 (drivers) $\times$ 512 (neurons) $=$ 114 688 programmable dynamic synapses can be defined on the chip. Setting up a specific network is a non-trivial effort and specific routing algorithms have been implemented in *Jeltsch* (2014). Additionally, the network structure can not be changed during an experiment. However, synapses can be deactivated, thus if the biggest network of the experiment to conduct fits on

the chip, also any subset can be used.

Configuration and experiment control of a chip is done via a dedicated field programmable gate array (FPGA), which in turn receives the experiment configuration by a host computer. The FPGA allows three different modes of operation, described in *Müller* (2014). In this thesis only the iterative/interactive mode is used, which consists of a configuration step that defines the network on-chip followed by multiple experiments. During the experiment runtime changes of some network parameters are possible. External spikes are buffered in the FPGA to ensure precise timing and fast execution speed. As can be seen in the next section, one FPGA is able to control eight HICANN chips at once. This is an important step to scale up the system, as 512 neurons are not enough to conduct large scale experiments. How the system is scaled up is presented in the next section.

2.2.2. Wafer

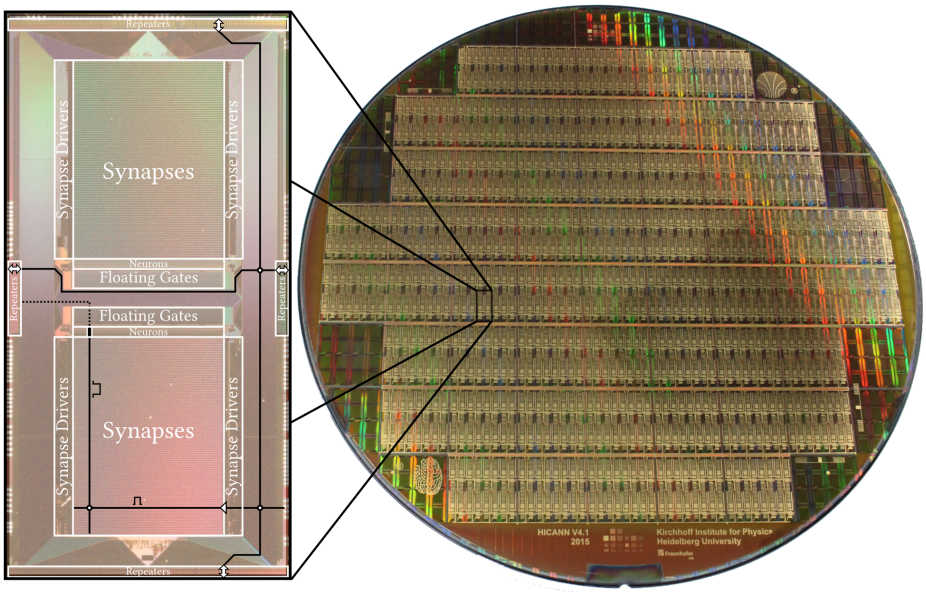

Figure 2.2.: **The HICANN chip and a postprocessed wafer.** The neurons are placed in the center. Next to them are the floating gate cells which store analog parameters of the chip. The synapse arrays are placed next to the neuron rows and can be configured to set up a network between the neurons. The wafer is one of the most recent version 4.1 wafers. It contains 384 identical HICANNs. The golden coat is a postprocessed layer that interconnects the chips. Pictures adapted from *Koke* (2017).

A single-chip system with 512 neurons is far from a large-scale neural network. or

larger networks a simple method is to interconnect multiple chips. The technique used in the BrainScaleS system is rather uncommon: it is called wafer-scale integration. The chips are not cut out of a silicon wafer as in the usual production process. Instead, the wafer is kept as a whole, enabling dense and cheap connections between the chips. The repeaters shown in fig. 2.2 recover the signal of individual spikes and send spikes to neighboring chips. The right side of this figure presents a whole wafer of the newest version 4.1, already coated in postprocessing with a golden coat to connect the chips. In principle, neurons can be connected over the whole wafer using the repeaters on the edges of each HICANN. External spikes are sent to the HICANNs via the FPGAs. Eight HICANN chips are managed by one FPGA.

Figure 2.3 shows a schematic of a full wafer with 384 HICANNs. As the HICANN chip length is about twice its width and 8 HICANNs are controlled by one FPGA, the wafer is subdivided in 2×4 quadrants called reticles. The numbers hovering over each quadrant are the reticle/FPGA coordinates. In total 48 FPGAs per wafer are used. These reticles are further divided in groups of eight which in turn share two analog outputs. These are used to measure analog signals on a chip, most commonly neuron traces. In the figure, they are depicted in different colors with names from P1/P4 to P3/P6 for top and bottom. The names stem from the Analog Readout Board (AnaB), shown in fig. 2.4 on the left together with the other components needed to operate the wafer. Besides the already mentioned FPGAs and AnaBs there is the AuxPwr, which supplies the power and the Main, vertical and horizontal PCBs. In conclusion, a whole wafer consists of 384 nearest-neighbor connected HICANNs, totaling almost 200,000 neurons and 4.4 million synapses. Still, this is less than the number of neurons of a fruit fly (*Nass and Przedborski*, 2011, p. 325). Therefore, multiple wafer systems can be combined, which is explained in the next section.

2.2.3. Wafer-Scale System

To scale the number of neurons and synapses further, multiple wafer systems can be combined (*Brüderle*, 2009). The BrainScaleS System in Heidelberg contains 20 wafers, resulting in approximately 4 million neurons and 0.8 billion synapses. It can be remotely accessed via the Human Brain Project Neuromorphic Computing Platform (*Davison*, 2018) by any interested member of the HBP upon request. The whole system can be configured using the abstract description API pyNN (*Davison et al.*, 2009) described in section 2.3.3. The HBP guidebook at *Davison et al.* (2017) explains how to get access and demonstrates basic experiment setups. The next section summarizes the system specifications needed to successfully conduct experiments on the hardware.

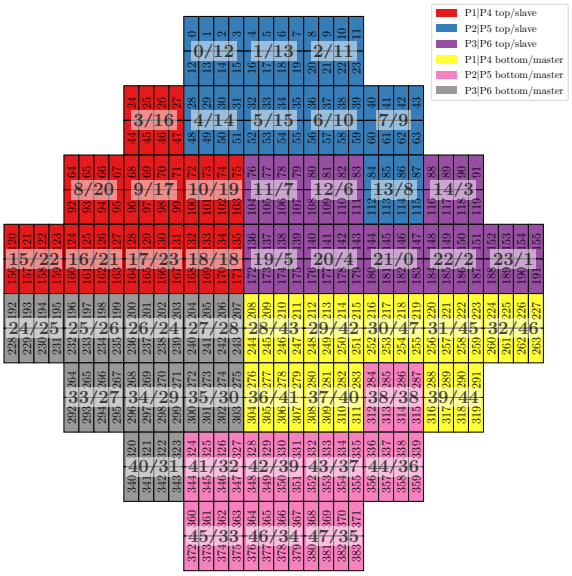

Figure 2.3: **Wafer schematic with HICANN, reticle and FPGA Coordinates.** Areas of the same color share two analog outputs which can be connected to a neuron to readout its trace during an experiment. At most two traces can be measured in one area of the same color. This limits the number of simultaneous measurements to 12. A block of 2×4 HICANNs is called a reticle and each reticle is controlled by one FPGA. The numbers hovering over the HICANNs are the reticle/FPGA coordinate, respectively.

2.2.4. System Specifications

In the following, the features and limitations that a system like the BrainScaleS System brings along are presented.

Transistor Mismatch

Not every transistor is exactly the same due to the finite production accuracy. Even if the variation from transistor to transistor is very little, in Very Large Scale Integration System (VLSI) like the BrainScaleS System, this can lead to undesired behavior and thus has to be characterized. *Schmidt* (2014) investigates these effects on hardware, based on Monte-Carlo simulations by *Kiene* (2014). The variations between circuits can be mostly compensated for by calibrating each neuron.

Floating Gates

The neuron parameters as well as other parameters of the system are stored in Floating Gate arrays (*Martins et al.*, 2009, 2010; *Serrano and Hasler*, 2004). Values are stored by putting a charge on a gate which in turn represents a digital 10-bit value. This procedure produces stable parameter values in time, as shown in *Klähn* (2017, pp. 48–45). Additionally, the cells are very small and have a low power consumption. However, reprogramming the floating gates comes with two caveats:

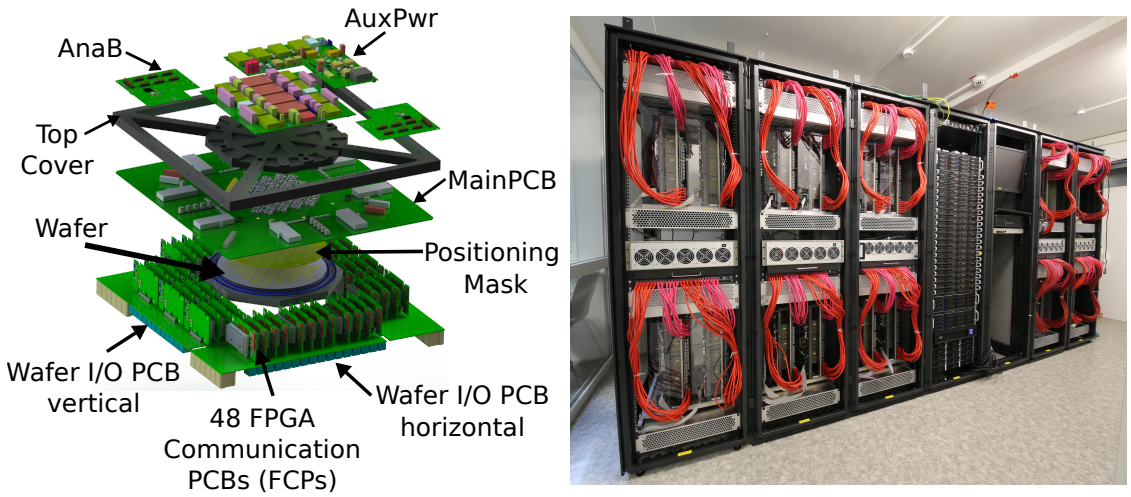

Figure 2.4.: **Wafer module and the BrainScaleS system.** *Left:* Exploded-view drawing of one rack place in the BrainScaleS system. The FPGA communication PCBs are each connected to eight HICANNs each. The insertion frame and top cover stabilize the module. AuxPwr supplies the power, the analog readout boards (AnaB) enable to read out analog signals like neuron traces. *Right:* The BrainScaleS system with 20 wafer modules. Each rack place contains four modules, behind the red cable harnesses. The racks in the middle carry power supplies and a compute cluster. Pictures taken from *Davison et al.* (2017).

First, the time for a full reprogramming cycle is of the order of one second. Furthermore, only rows can be configures independently, thus changing a single value always requires to reprogram the whole row. Therefore, if possible, the network should be configured only at the experiment start. The weights are stored in digital Static Random-Access Memory (SRAM) cells and, hence, can be changed quickly during experiments. Second, the floating gate cells are rather imprecise, resulting in trial-to-trial variations between two reprograms even if the parameters are unchanged. These variations can not be compensated for by calibration. Therefore, experiments on the system should either be tolerant against parameter variations between neurons or a pre-measurement has to be executed each time to determine the actual parameter values for the experiment.

Synapse Loss

The aforementioned value of 114,688 synapses per chip is actually the maximum number of synapses. However, if complex network structures are needed, the actual value before synapse loss occurs can be lower due to a lack of hardware resources. Furthermore, each neuron has only two synaptic inputs, one inhibitory and one excitatory. These can be connected to one half of the synapse drivers, resulting in

a maximum of 224 inhibitory/excitatory connections for a single neuron. To solve this problem, the synaptic inputs of multiple neighboring neurons can be combined, while at the same time lowering the total neuron count. At most 64 neurons can be used together, resulting in $224 \times 64 = 14\,336$ inputs and 8 neurons per chip. The minimum neuron size N can be calculated if the number of incoming synapses n_s is known as $N = \lceil n_s/224 \rceil$, with the ceiling operator $\lceil \cdot \rceil$.

Bandwidth

As with every connection, the bandwidth to send spikes from neuron to neuron is limited. The units are MegaEvents per second, one event being equal to one spike. However, this is measured in wall clock time, therefore the number of biological spikes per second is smaller by a factor of 10^4, the speedup of the HICANN chip. First, the Host has to send external spikes to the FPGA via network. With 30 MEv/s, the bandwidth from Host to FPGA and back is hardly ever a problem. Additionally, the FPGA is able to buffer about 312 MEv. Then, spikes are sent from the FPGA to one or more HICANNs. The bandwidth limit measured by *Koke* (2017) is lower than the theoretical one of 25 MEv/s, at about 17.8 MEv/s. This can be limiting, if example one does not want to use the background generator to have complete control over the spike timings. Then, at a biologically plausible rate of about 10 Hz per neuron, it would only be possible to use $17.8\,\text{MEv/s} \div 10^4 \div 10\,\text{Hz} = 178$ neurons of the chip. It should be noted, that these bandwidth limits are not a lower limit, so in practice the bandwidth can be even smaller. The bandwidth of 62.5 MEv/s from HICANN to HICANN does not cause these limitations. Therefore, one can feed spikes to neurons of one HICANN by using several FPGAs and/or neighboring HICANNs.

Blacklisted components

Fabricating millions of circuits on a wafer, it is inevitable that some components will not work at all, like single neurons, repeaters or synapse drivers. Therefore, during system calibration, coordinates of neurons that show unusual behavior are saved to a file and are not used in experiments. Results of blacklisted neurons are presented in section 3.2.

2.3. Software Packages and Modules

The software packages and tools that were used during the course of this thesis are presented in this section.

2.3.1. cake

cake is the calibration software developed in the group, main contributors are C. Koke, S. Schmitt, M. Kleider and D. Schmidt (c.f. *Kleider*, 2017; *Koke*, 2017; *Schmidt*, 2014). It is written in Python and features scripts to calibrate single HICANNs, evaluate the calibration and plot measurement results. Calibration routines are defined for each neuron parameter. Hardware configuration and parameter steps are in a separate file and are loaded at the start of a calibration or evaluation run. For a more detailed description of the individual neuron parameter calibration routines, see the aforementioned theses.

2.3.2. SLURM

The SLURM workload manager controls the local cluster nodes. This is in general useful to divide the cluster resources (CPUs, RAM, etc.) equally among the users. Furthermore, SLURM gives out licenses, which are used to manage the access to the hardware units such as the wafer setups and other prototype and demonstration chips. If multiple users request the same licenses, the jobs are queued and executed successively. This guarantees that multiple experiments do not interfere. For a detailed description of the implementation on the cluster of the BrainScaleS system, *Mauch* (see 2016).

2.3.3. PyNN and NEST

PyNN is a high-level abstract neural network description API for Python. Network models are defined simulator-independently and then executed on a simulation backend. This backend can be a software simulator like NEST, or an emulation device like the BrainScaleS system. This allows to write code that can be executed effortless in simulation and emulation.

2.3.4. pyHMF, marocco and the BrainScaleS software stack

pyHMF is the backend for the BrainScaleS system, short for python Hybrid Multiscale Facility. It is responsible for translating the biological network description to the hardware. As part of this, marocco performs the mapping to decide which neurons in software are represented by which hardware neurons. Moreover, the synapses in software are assigned to hardware synapses. This is a complex procedure as the algorithm has to include hardware constraints and blacklisted components. For a thorough description, see *Jeltsch* (2014).

2.3.5. TensorFlow

TensorFlow (*Abadi et al.*, 2016) is an open-source software library for machine learning. With its abstract description API and highly efficient backend it is especially useful to set up artificial neural networks (ANNs). These networks only merely resemble neural networks in the fact that one builds up a graph of neurons (vertices) and edges with weights (synapses). But instead of processing spikes through the network and evolving the network in time, data that is input in the network is simply passed to the next layer with some activation function. Evolution of the weights depends on a mathematical algorithm which minimizes the value of some predefined error function. A simple example of such a network is explained in section 4.4. TensorFlow version 1.0.0 was used throughout this thesis.

3. Wafer-Scale Calibration

A perfect emulation of a neuron with analog hardware is not possible because of transistor level variations. Calibration routines are important to reduce these manufacturing inaccuracies. A framework for single-HICANN calibration of the neuron parameters already exists (see section 2.3.1). In this section, the calibration of the refractory period is scrutinized. It is challenging to calibrate due to the fact that very small times have to be measured. It is expected that floating gates limit the calibration precision (c.f. section 2.2.4). Furthermore, the single-HICANN calibration is scaled up to calibrate whole wafers and results for three of the newest wafers in the BrainScaleS System are shown. An analysis of the other neuron parameters besides τ_{ref} has already been done with this software in *Kleider* (2017). All times and voltages in the following sections will be in the hardware domain, c.f. section 2.1.2 for the translation to the biological domain.

3.1. Calibration of the refractory period τ_{ref}

3.1.1. Method

Each neuron has a distinct time interval after it spiked in which it can not be excited again, cf. fig. 2.1 panel H and L. This time interval (minus the rise time) is called the refractory period and can be controlled for each neuron by the pulse current I_{pl}. To measure this period, the neuron is set to a constant-spiking state by setting the spike threshold V_{t} below the resting potential E_{l}. The measurement procedure is as follows: First, all neurons are set to the smallest possible interspike interval t_{min} by setting I_{pl} to the maximum value of about 2500 nA. A trace is recorded, the spike times measured and the average time difference between spikes t_{min} is calculated, for each neuron separately. Second, measurements at different values for I_{pl} are recorded in the same manner, resulting in time differences $t(I_{\text{pl}}) = t_i$. Then, for every neuron n the refractory time at this value of I_{pl} can be calculated as

$$\tau_{\text{ref}}(i, n) = t_i(n) - t_{\text{min}}(n). \tag{3.1}$$

An exemplary measurement of one neuron for $t(I_{\text{pl}} = 10\,\text{DAC})$ and t_{min} is shown in fig. 3.1. Subtracting the rise time from the measured spike difference t_i gives

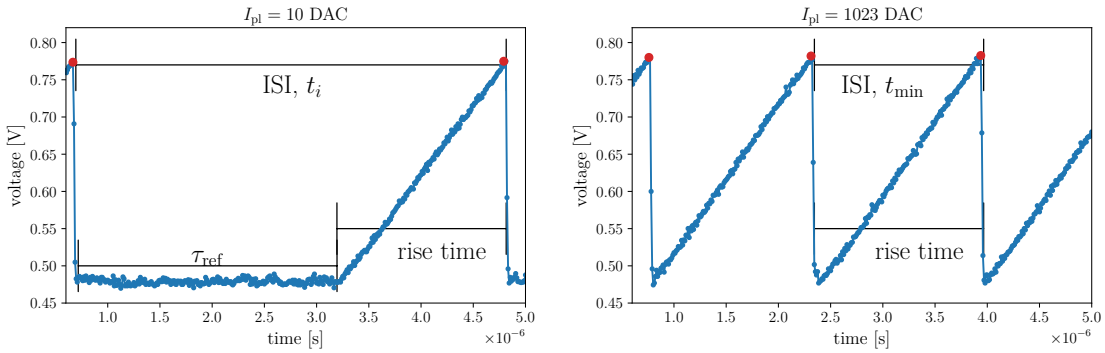

Figure 3.1.: **Constant-spiking state of a neuron on hardware.** *Left:* The refractory period can be identified as the length of the baseline of the voltage trace. Together with the rise time, it adds up to the interspike interval. *Right:* At the highest current setting, no refractory period can be identified, which means its less than the time resolution of the analog readout sample rate. Thus, the rise time coincides in this case with the interspike interval.

the refractory period τ_{ref}. On the right, the same neuron's trace is shown for the maximum value of I_{pl}. The rise time marker is not changed in length between the panels, confirming that the rise time does not change in this example when changing the refractory time. This is also the expected behavior from simulation if no other parameters are changed, c.f. fig. 2.1. However, if the membrane time constant τ_{m} or the reset potential V_{reset} changes, also the rise time will change. This can inevitably occur when setting new floating gate values between measurements. This introduces trial-to-trial variations, c.f. section 2.2.4.

Multiple values for t_i are recorded and the respective value for τ_{ref} are calculated to get a calibration curve. The ideal curve function was determined with transistor-level simulations in *Schwartz* (2013, p. 69) to be

$$I_{\text{pl}} = \frac{1023}{\frac{62.5}{\mu\text{s}}\tau_{\text{ref}}[\mu\text{s}] + 1}[\text{DAC}]. \tag{3.2}$$

To fit this function, at least one constant parameter has to be defined as a free fit parameter. After testing different free parameter sets on one HICANN, it was chosen to use the fit function

$$I_{\text{pl}} = \frac{1}{a\tau_{\text{ref}} + b} \tag{3.3}$$

with two free parameters a and b. The starting points for the fit can be extracted from the ideal calibration in eq. (3.2) to be $b_0 = 9.78 \times 10^{-4}\,\text{DAC}^{-1}$ and $a_0 = 6.11 \times 10^{4}\,\text{s}^{-1}\,\text{DAC}^{-1}$.

An example of the fit and the corresponding residuals can be seen in fig. 3.2. The fit curve follows the trend of the measurement points. The relative error for b is 15.3, which is justified by the fact that the current $I_{\text{pl}}(\tau_{\text{ref}} = 0)$ can not be determined accurately from the data as the curve has a singularity at $\tau_{\text{inf}} = -\frac{b}{a}$ and the steepness of the curve diverges to ±infinity.

The residuals in the right panel of fig. 3.2 show up to 27% deviation to the fit function. This effect is explained by the trial-to-trial variations between runs: Writing neuron parameters in the floating gate cells can only be done with a finite accuracy, c.f. section 2.2.4. This effect can for a calibration measurement be averaged out when enough repetitions are recorded. Still, it limits the precision of the calibration. The standard deviation of the distribution of the times is investigated in section 3.1.3 to estimate the impact on the accuracy of τ_{ref}.

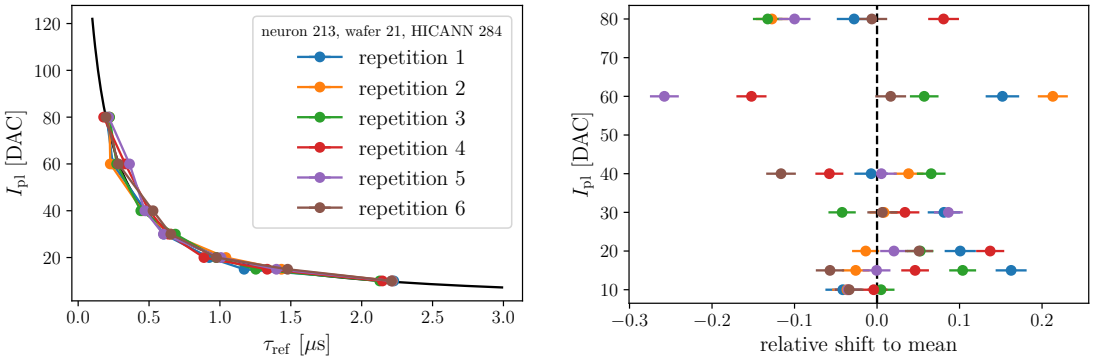

Figure 3.2.: **Calibration curve of a neuron on hardware.** Six runs are recorded to reduce trial-to-trial error. The curve in eq. (3.3) is fitted to all data points. The fit parameters are $a = (45\,100 \pm 2100)\,\text{s}^{-1}\,\text{DAC}^{-1}$ and $b = (0.004 \pm 0.063)\,\text{DAC}^{-1}$. The error bars are the standard error as in eq. (3.4) for approximately $N = 100$ ISIs. These do not account for the trial-to-trial variation, which could be measured in a second run by taking the standard deviation of multiple measurements at the same value for I_{pl}. This is investigated in section 3.1.3.

3.1.2. Results

This section discusses the results of the calibration of the refractory period τ_{ref}. The measurements presented are done on three wafers, 599 HICANNs, in total 306 688 neurons. The method to calibrate HICANNs in parallel is presented in section 3.2.

		without calibration			with calibration		
I_{pl} [DAC]	target [µs]	mean [µs]	σ [µs]	$\tau_{\mathrm{ref},0}$ [%]	mean [µs]	σ [µs]	$\tau_{\mathrm{ref},0}$ [%]
10	1.621	1.8	1.5	1.3	1.11	0.43	1.9
17	0.9468	1.01	0.75	1.4	0.61	0.31	2.2
25	0.6387	0.66	0.48	1.7	0.40	0.39	3.3
33	0.48	0.49	0.35	2.2	0.29	0.16	5.0
41	0.3832	0.38	0.27	2.9	0.23	0.33	7.0
48	0.325	0.32	0.23	3.7	0.19	0.23	9.0
56	0.2763	0.26	0.19	4.8	0.17	0.14	11.4
64	0.2397	0.23	0.17	5.7	0.14	0.14	14.1
72	0.2113	0.20	0.15	7.1	0.13	0.33	16.3
80	0.1886	0.18	0.14	8.5	0.117	0.099	18.9
100	0.1477	0.14	0.11	11.9	0.092	0.093	24.4
275	0.04352	0.049	0.065	36.0	0.047	0.074	46.9
450	0.02037	0.036	0.058	45.8	0.040	0.070	52.7
625	0.01019	0.031	0.055	51.0	0.037	0.067	55.9
800	0.00446	0.030	0.055	52.6	0.036	0.066	57.0

Table 3.1.: **Results for the calibration of the refractory period.** The standard deviation σ is in all but five cases lower with calibration. The column I_{pl} is only relevant without calibration, as with calibration, each neuron is set to its respective I_{pl} value. The column $\tau_{\mathrm{ref},0}$ denotes the percentage of neurons with $\tau_{\mathrm{ref}} = 0\,\mathrm{s}$. The distance between mean value and target as well as the number of neurons with a measured refractory period of 0 is higher with calibration. This indicates that the fit introduces a bias to the refractory period. At about 0.15 µs the refractory period is without calibration as precise as with calibration.

Table 3.1 lists average values for the refractory period with and without calibration. The target value without calibration is calculated with eq. (3.2) and the DAC value in the table. With calibration, the DAC values are calculated individually for each neuron from their calibration curve. To determine the fit curve, each neuron was measured for these eight values of I_{pl}: $\{10, 15, 20, 30, 40, 60, 80\}$ DAC and four repetitions.

The standard deviation is in all but five cases lower after calibration. The smaller the target value for τ_{ref}, the closer the mean and standard deviation with and without calibration are. This is expected for two reasons: First, the measurement precision declines with smaller values of τ_{ref}, c.f. section 3.1.3. Second, the steepness of the fit curve increases for smaller values, rendering the fit more sensible to deviations at small values. Even at 10 DAC, the number of neurons with $\tau_{\mathrm{ref}} = 0$ is with $\approx 1.5\%$ non-negligible, both with and without calibration. Neurons which

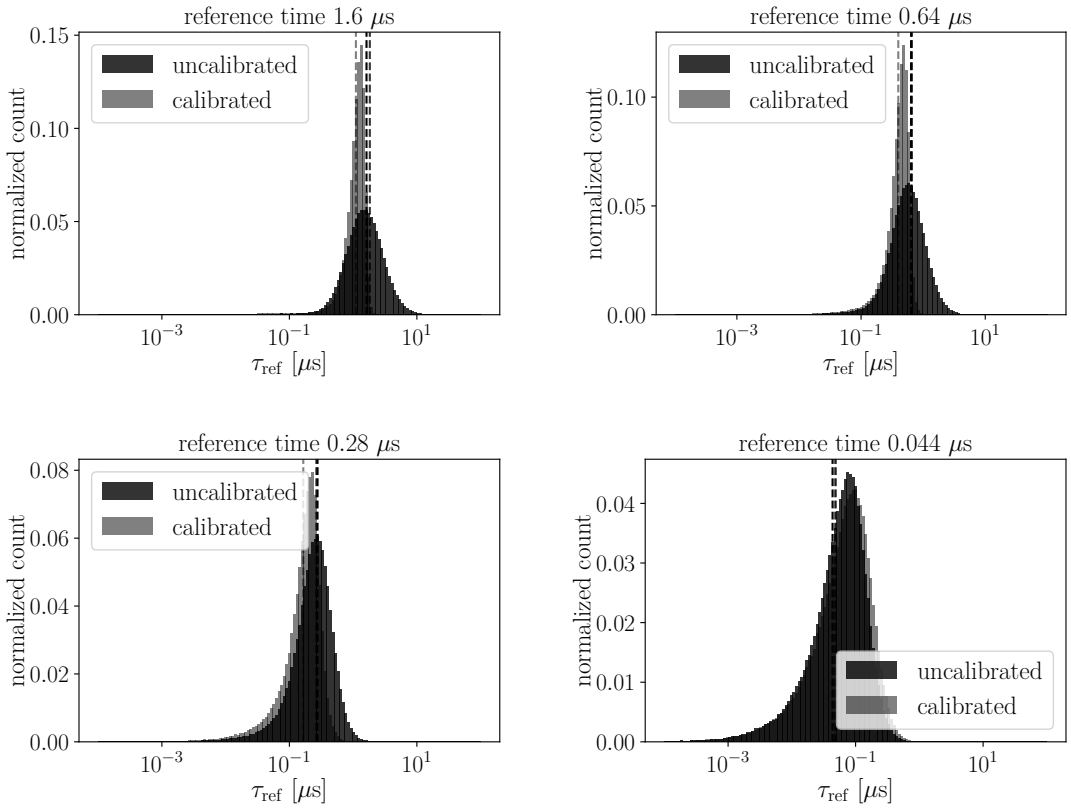

Figure 3.3.: **Histograms for four target values with and without calibration on hardware.** Dashed lines of the same color as the bins mark the mean value. The black dashed line is the target value. Long target values around 1 µs the calibration results in a higher peak and smaller width of the distribution. The shape in both cases is a skew normal distribution in log-space. The shorter the target value, the closer are the distributions with and without calibration.

have a refractory period of 0 at 10 DAC are not really utilizable, because their whole refractory period range is somewhere between 0 DAC and 10 DAC.
The 0-refractory period neurons are in all cases more with calibration than without. This is also accountable to the fit curve, which introduces a bias. This can also be seen by the smaller mean values in all but the last three cases.

For four exemplary target values of table 3.1, histograms are presented in fig. 3.3. The distributions are very similar for low values of $\tau_{\text{ref}} < 0.05$ µs. For larger values > 0.5 µs the distribution with calibration is narrower than without. The mean value is shifted to lower values of τ_{ref}. Values with a refractory period of zero are excluded.

In conclusion, it is demonstrated that the calibration reduces the width of the distributions of the refractory time for refractory times > 0.5 µs. This translates

to 5 ms bio time. The calibration introduces a bias to the mean value, that results in smaller values than the target value. The next section identifies error sources and estimates the minimum variation of τ_{ref} in experiments due to the trial-to-trial variations.

3.1.3. Validation of the Measurement Procedure

The calibration method described in section 3.1.1 has the advantage to be easily implementable. However, calculating the difference of two variables with errors leads to potentially very small results, while at the same time the error squared is summed. Furthermore, different kinds of errors are present during calibration. These are:

- Accuracy of the peak detection in software
- Sample rate of the ADC readout of 10.4 ns
- Trial-to-trial variations when setting neuron parameters

The accuracy of the peak detection is limited in two ways: First, the peak time can only be detected with a limited accuracy because the trace is noisy. This is a statistical error and can be averaged out when measuring enough spikes. Then, individual peaks may not be detected at all. If this happens only scarcely, the outliers can be detected by comparing each measured value by the mean value for all measurements: if the deviation from the mean is too high, it is most probable that one spike time was not detected and the value can be discarded. This method introduces a bias if many spikes are not detected, but for a test set of 8 HICANNs, only $489/25462451 \approx 2 \times 10^{-4}$ spikes were not detected correctly.

The sample rate limits the measuring accuracy due to the fact that the trace falls down rapidly after a spike, and if the sample is shortly after the spike, the recognized spike time will approximately be the real spike times minus the inverse sampling rate. When taking time differences, this leads to fluctuations around $\pm$ the inverse sampling frequency. This is counted as a statistical error and can be reduced when recording multiple spikes. The same fluctuations are also caused by an imperfect detection of the spike peak due to noise. Both errors combined are identified as the standard error, and calculated with N measurement samples and the standard deviation $\sigma(t_i)$ as:

$$\sigma_{\text{mean}}(t_i) = \frac{\sigma(t_i)}{\sqrt{N}} \tag{3.4}$$

The last error source is caused by the floating gates that set the neuron parameters. They have a limited precision, as explained in section 4.1.3. This causes fluctuations around the mean value with a certain width σ_{FG}. This can be averaged out in the calibration by recording multiple repetitions while re-setting the floating gates between measurements. In an experiment however, this error limits the total precision with which a parameter can be set. This error is therefore counted as an additional error to the mean value.

Using the usual calibration procedure described in eq. (3.1), the statistical error can be calculated using error propagation as

$$\Delta\tau_{\text{ref,stat}} = \sqrt{\sigma^2_{\text{mean,i}} + \sigma^2_{\text{mean,min}}} \tag{3.5}$$

$$= \sqrt{\frac{\sigma^2_{\text{t,i}}}{N^2} + \frac{\sigma^2_{\text{t,min}}}{M^2}}. \tag{3.6}$$

The additional trial-to-trial error then is added linearly, but these errors itself are also summed squared. The total error then is

$$\Delta\tau_{\text{ref}} = \sqrt{\sigma^2_{\text{mean,i}} + \sigma^2_{\text{mean,min}}} \pm \sqrt{\sigma^2_{\text{FG,i}} + \sigma^2_{\text{FG,min}}} \tag{3.7}$$

First, the statistical error is examined for different numbers of repetitions. A set of 8 HICANNs on wafer 33 was used to record 100 repetitions for three values of $I_{\text{pl}} \in \{20, 200, 1023\}$. This measurement took about 11 h per HICANN and is therefore unsuitable for a real calibration of multiple wafers. A set of 6 repetitions has been selected for comparison. To get good results for the calibration curve fit, about 8 times t_i have to be recorded for different values of I_{pl}. This takes about an hour to measure for 6 repetitions.

The results are shown in fig. 3.4. The quantile is defined as the fraction of neurons that have a standard deviation below the standard deviation on the x-axis. The absolute times $\tau_{\text{ref,i}}$ vary from neuron to neuron. This can be seen by calculating the mean and standard deviation over all neurons: $\tau_{\text{ref,20}} = (0.90 \pm 0.63)\,\mu\text{s}$, $\tau_{\text{ref,200}} = (0.050 \pm 0.032)\,\mu\text{s}$. Therefore it was chosen to present the errors on the x-axis as relative errors, which should be comparable between neurons. The top panel show the quantile for $I_{\text{pl}} = 20\,\text{DAC}$ and 6 and 100 repetitions. As expected, the relative error decreases when increasing the number of repetitions. The median m_j also decreases from $m_6 = 0.045$ to $m_{100} = 0.012$. Still, even the error of about 4.5% for 6 repetitions is low enough to guarantee that most neurons can be calibrated in this range. Close to 90% of the neurons are below 10% relative error. In the bottom panel, the standard deviation is considerable higher, with a median

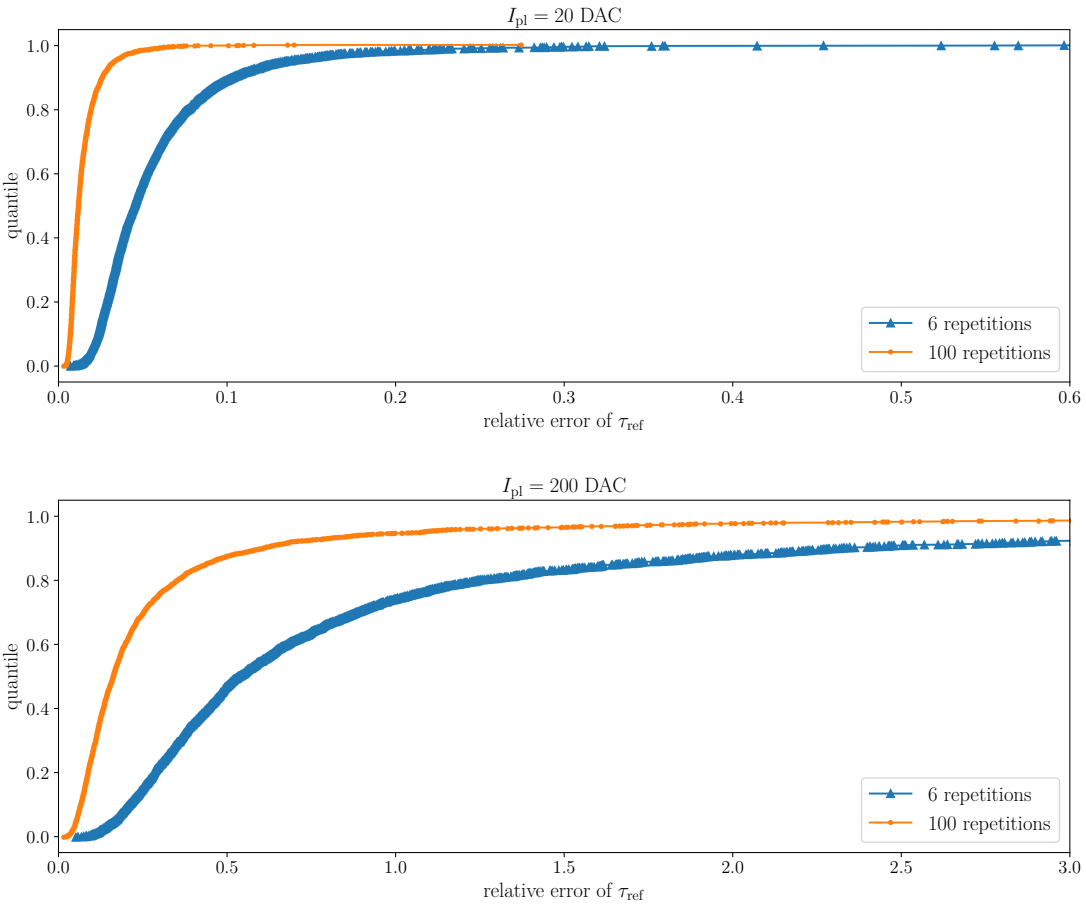

Figure 3.4.: **Quantile of the standard error of $\tau_{ref,i}$ for two values of I_{pl} on hardware.** *Top:* The current is set to $I_{pl} = 20\,\text{DAC}$, a low value. The error is below 10% for about 90% of the neurons at 6 repetitions. This should suffice to calibrate the neurons in a range around $I_{pl} = 20\,\text{DAC}$. *Bottom:* A higher value of $I_{pl} = 200\,\text{DAC}$ results in much higher errors. For 6 repetitions, the number of neurons with a relative error below 10% is less than 1%. For 100 repetitions it is about 25%.

of $m_6 = 0.535$ and $m_{100} = 0.163$. This can be attributed to eq. (3.1). The difference results in small values, but the errors for both times are squared and then summed. For 6 repetitions, a value of 200 for I_{pl} is not reasonable. Even for 100 repetitions, still half of the neurons have a relative error of at least 16%. In conclusion, the calibration method is only feasible for small values of I_{pl}. Next, the trial-to-trial variations are investigated.

To measure the trial-to-trial variations, 100 repetitions have been recorded for three values of I_{pl}: 20 DAC, 200 DAC and 1023 DAC. Quantiles of the absolute stan-

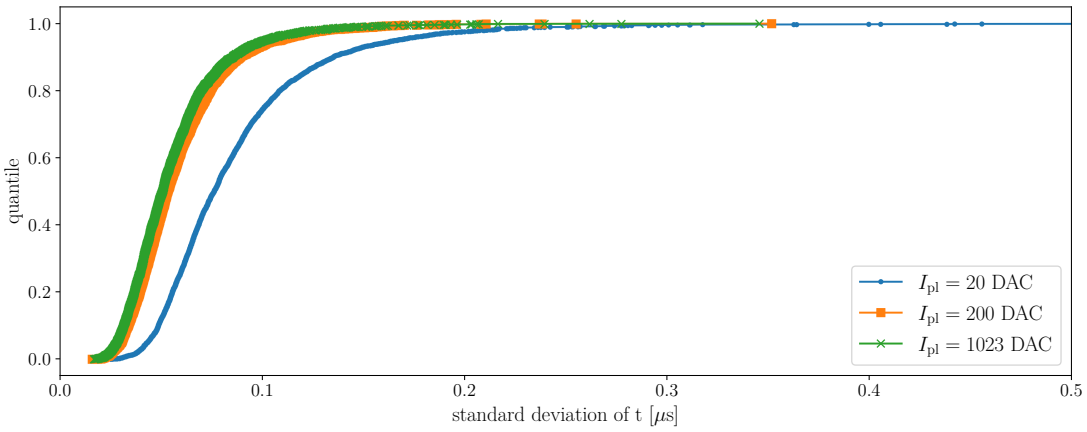

Figure 3.5.: **Quantiles of the trial-to-trial variation for three values of I_{pl} on hardware.** The quantile for $I_{\text{pl}} = 20\,\text{DAC}$ is always below the other quantiles.

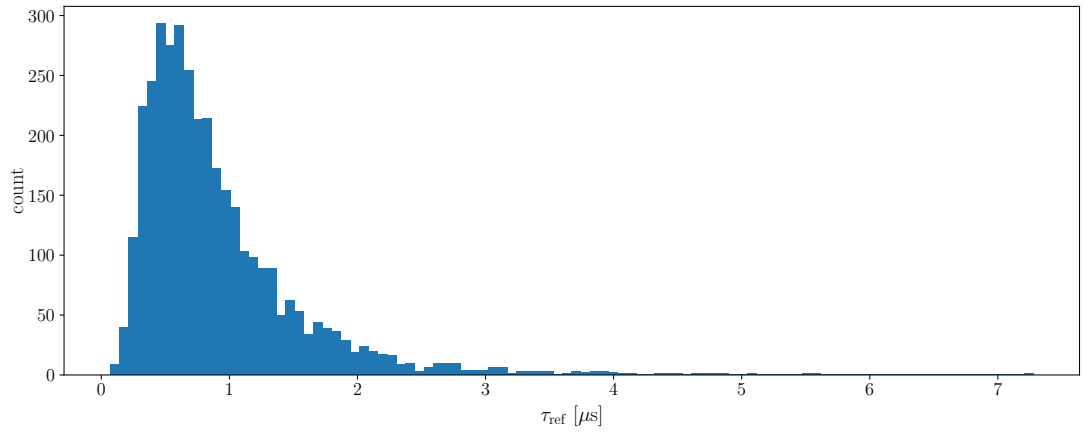

Figure 3.6.: **Refractory times for $I_{\text{pl}} = 20\,\text{DAC}$ on hardware.** These values are used to estimate the maximum value of τ_{ref}. This is done because the measurements for lower values of I_{pl} are influenced even more by the trial-to-trial error. The median is $m_\tau = 0.73\,\mu\text{s}$.

dard deviation are shown in fig. 3.5. The medians m_I for I_{pl} are $m_{20} = 7.64 \times 10^{-8}\,\text{s}$, $m_{200} = 5.34 \times 10^{-8}\,\text{s}$ and $m_{1023} = 5.10 \times 10^{-8}\,\text{s}$. As the error $\Delta\tau_{\text{ref}}$ scales approximately linear with the absolute trial-to-trial error $\sigma_{FG,i}$ (c.f. eq. (3.7)), the relative trial-to-trial error on average is about 8% for low values of I_{pl}. This means that on average, the refractory period can only be set with a precision of 8%. In the current method used, the two trial-to-trial errors of t_i and t_{min} are added quadratically. The error for τ_{ref} at $I_{\text{pl}} = 20\,\text{DAC}$ can therefore be estimated as $\Delta_{\text{FG}} = 9.2 \times 10^{-8}\,\text{s}$.

In conclusion, the statistical error for small values $\tau_{\text{ref}} < 5 \times 10^{-8}\,\text{s}$ can not be

lowered enough for many neurons to allow a good calibration, as the number of repetitions necessary is too high. For larger values of τ_{ref}, a calibration is possible, but limited by the trial-to-trial error. It is expected to have about 10% variations for values of about 1 µs and larger variations for smaller values. The relative standard deviation of τ_{ref} of about 70% for a fixed value of I_{pl} indicates that the range of the refractory period is very diverse between neurons. Using values for I_{pl} lower than 20 DAC increases the trial-to-trial error. The maximum value for the refractory period can therefore be estimated by the median $m_\tau = 0.73\,\mu\text{s}$ in fig. 3.6.

3.1.4. Conclusion

The previous section showed that the error for the refractory period is too high for values of $I_{\text{pl}} > 200\,\text{DAC}$. This can not be improved much by measurement, as the calibration measurements are ultimately limited by the sample rate of the ADC readout of 10.4 ns which is the same order as more than half of the values for the refractory period. In addition, the trial-to-trial error is about 50 ns, which requires many repetitions to average the error out. At the same time, even after calibration the refractory period can not be set more accurately than the trial-to-trial error. In a range of about 1 µs however, the trial-to-trial is only about 10% and the statistical error can be reduced enough by a small number of repetitions. A direct measurement of the refractory period, e.g. by determining the minimum values of the trace is not expected to give significantly better results. The major improvement is to only have the trial-to-trial error of one measurement instead of two. However, it has been shown that this error is already low enough for large values of the refractory period.

3.2. Wafer-Scale Calibration

This section discusses the requirements for a framework to calibrate as many HICANNs at once as possible, keeping track of all calibrations and automating the process. Results for three wafers in the BrainScaleS system are presented.

3.2.1. Method

The cake calibration package is designed only for single HICANNs. As the calibration of individual HICANNs is independent, it is possible to start a calibration run for each HICANN separately. To acquire additional information about each run and automate data visualization, a four-step pipeline has been designed during the course of this thesis. A schematic is depicted in fig. 3.7. In the first step, preliminary requirements are checked: loaded software packages, version of the wafer and state of the calibration parameter files. If these checks are passed, an

Figure 3.7.: **Schematic of the calibration pipeline.** Each step (blue boxes) is executed when the previous step was successful.

actual calibration run is started. On finishing this run, an evaluation run is started and at the same time plots are generated for the available data. If the evaluation finishes, plots for this data are generated, too. The data format has been changed to pandas DataFrames written on disk in the Hierarchical Data Format (HDF5). This simplifies data access and analysis and decreased the file size. Furthermore, the data visualization step has been updated to use DataFrames. Information about the current state of each calibration is written on disk. This together with the calibration data can be used to visualize the calibration state. An example is shown in fig. 3.9.
The access to specific wafers, FPGAs and HICANNs is managed by SLURM. Section 2.3.2 gives a brief explanation of SLURM's features, a detailed explanation can be found in *Mauch* (2016). If a specific resource is already used, SLURM puts this job in a queue until the previous job finished. Therefore, the calibration jobs for all HICANNs can be put in the queue and then will be executed as fast as possible. Each HICANN needs one analog out for the calibration measurements. As only two outputs are shared by eight FPGAs, This limits the number of parallel calibration runs to $48 : 8 \times 2 = 12$. A 96 channel digitizer has been developed by *Ilmberger* (2017) and is currently integrated into the system, which will increase the number of parallel calibrations significantly.
The time of a single HICANN calibration stronlgy depends on the number of data points to be recorded for each parameter. A sensible set of calibration ranges has been established in *Kleider* (2017); *Koke* (2017); *Schmidt* (2014), which results in a total calibration time of about 8 h per HICANN. In the ideal case of 12 calibrations in parallel, the calibration of a whole wafer module takes about 256 h or about 11 days. If a calibration has to be interrupted or fails, it can be resumed from the last successful data point measurement.

The pipeline is designed to automate the process of calibrating whole wafers. A single command starts the pipeline for all HICANNs passed to the command. Folders are created automatically for each HICANN and measurement results like fit curves and distributions of data points are plotted. If a calibration fails for some reason, it can be restarted by the same command. To keep track of all calibrations,

Wafer	21	33	37
calibrated	98 848	111 321	70 457
blacklisted	12 768	14 119	28 359
not yet calibrated	19 968	12 800	36 352
excluded	65 024	58 368	61 440
calibrated relative to blacklisted [%]	88.6	88.7	71.3

Table 3.2.: **Neuron counts after calibration for three wafers.** Not yet calibrated neurons are on HICANNs where the calibration did not finish. Excluded neurons are all neurons on HICANNs that could not be initialized. The percentage of calibrated neurons relative to blacklisted is about 17 percentage points higher on wafer 21 and 33 compared to wafer 37.

the user can plot a 2D wafer map as shown in fig. 3.9.
Before the pipeline is started, it is important to ensure that all FPGAs and reticles that should be calibrated are turned on. In between calibration runs, it is advisable to reprogram the FPGAs and initialize the reticles to ensure that the system is in a sane state. Then, calibrations ca be (re)started.

The next section presents the results for the three version 4.1 wafers 33, 21 and 37. All measurements where recorded using the new pipeline script and analyzed using the helper functions that were implemented for this matter.

3.2.2. Results

An overview of the current calibration state for the three wafer systems 33, 21 and 37 is given in this section. Results are shown in figs. 3.9 to 3.11. The figures show 2D views of the respective wafer with a color-coded calibration status. The left panel shows the calibration status and green HICANNs could be calibrated successfully. On the right is the evaluation view, again green HICANNs could be evaluated without error. To use the calibration, it is not necessary to do the evaluation, but it is essential to check if the calibration contains obvious errors.
In the following, the errors that occured are described. A number of reticles had to be excluded because the highspeed initialization does currently not work for at least one HICANN on this reticle. These are marked in a dark gray. Currently, it is not possible to initialize the other HICANNs on that reticle. This will be integrated in the software in the future. The cause for highspeed init failures is under investigation.
The most prominent error is that only very few or no spikes are measured in the HICANN preout. This is a check done before starting the calibration. If the HI-

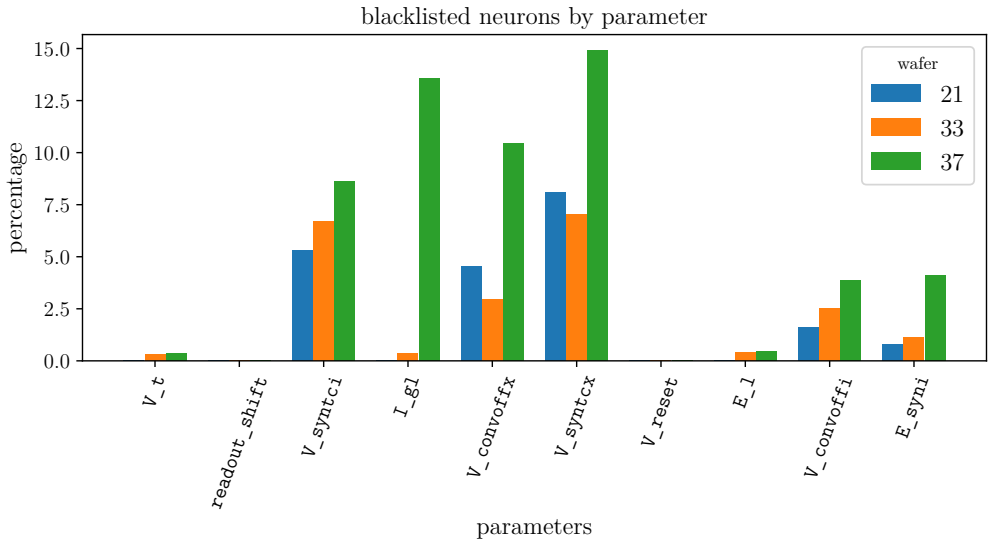

Figure 3.8.: **Blacklisted neurons by parameter calibration routine.** The percentage is calculated by dividing the number of blacklisted neurons by all neurons. Each routine tests each neuron, even if it has been blacklisted in a previous calibration step. Therefore, the range for each bar is from 0 to 100. The percentage of blacklisted neurons for wafer 21 and calibrations `V_t` and `E_l` is below 0.002 %. Wafer 37 has the highest count of blacklisted neurons for all calibration. The count is comparable for wafer 21 and wafer 33. The number of blacklisted neurons for the I_{gl} routine is significantly higher for wafer 37 than for the other wafers.

CANN preout does contain only very few spikes, it is probable that the Layer1 which transports the spikes between neurons has an issue. In most cases this is due to a neighboring HICANN or the HICANN itself not being properly initialized. However, if the highspeed connection does not work, it is not possible to put this HICANN in a sane state for measurements. It can be seen that most HICANNs with the weak preout signal are next to a HICANN where the highspeed initialization does not work or on the edge of the wafer.
The connection timeout occurs if at least one packet is lost between host and FPGA. This can be due to a high network load or because the FPGA is turned off or defective. In the future, a link with error control will be installed to make the connection more reliable.
Problems with the Analog Readout Boards only occur rarely but the reason for a non-responsive board is still under investigation. To get an overview of all possible errors and possible causes, Electronic Vision(s) maintains a frequently updated *ErrorFAQ* (2018).

In fig. 3.12 the number of blacklisted neurons are shown for each HICANN on all free wafers as a 2D wafer view. The color encodes the number of blacklisted neurons

as depicted in the colormap. As more HICANNs are green on wafer 37 than on the others, it can be seen that the blacklisted rate is higher on this wafer than on the others. One defective reticle was found during calibration. Reticle 16 on wafer 33 was not calibratable because the measurements only return corrupt data. It is assumed that there is a problem with the analog readout.

A summary of the neuron count is given in table 3.2. The number of not yet calibrated neurons and blacklisted neurons is considerably higher on wafer 37 compared to wafer 21 and 33. An explanation can be deducted from fig. 3.8. This figure presents the relative count of blacklisted neurons itemized by parameter calibration. It can be seen that the calibration for I_{gl} lead to much more blacklisting on this wafer than on the others. Additionally, the blacklisted neuron count on wafer 37 is higher for all other calibrations.
The blacklisted neurons for wafers 33 and 21 stem mostly from the $V_{\mathrm{syntc,x}}$ and $V_{\mathrm{syntc,i}}$ calibration. The method to blacklist neurons is described in *Koke* (2017, pp. 130–132). An investigation on two HICANNs of wafer 21 revealed that all blacklisted neurons came from the rejection of data points after measurement because the signal-to-noise ratio defined for this calibration was to low. Further investigation is needed to examine if the signal-to-noise ratio can be increased or the tolerance is set to low.

Concluding, 656 HICANNs on wafers 33, 21 and 37 have been calibrated successfully. The average rate of not blacklisted neurons is about 88% for wafers 21 and 33 and about 71% for wafer 37.

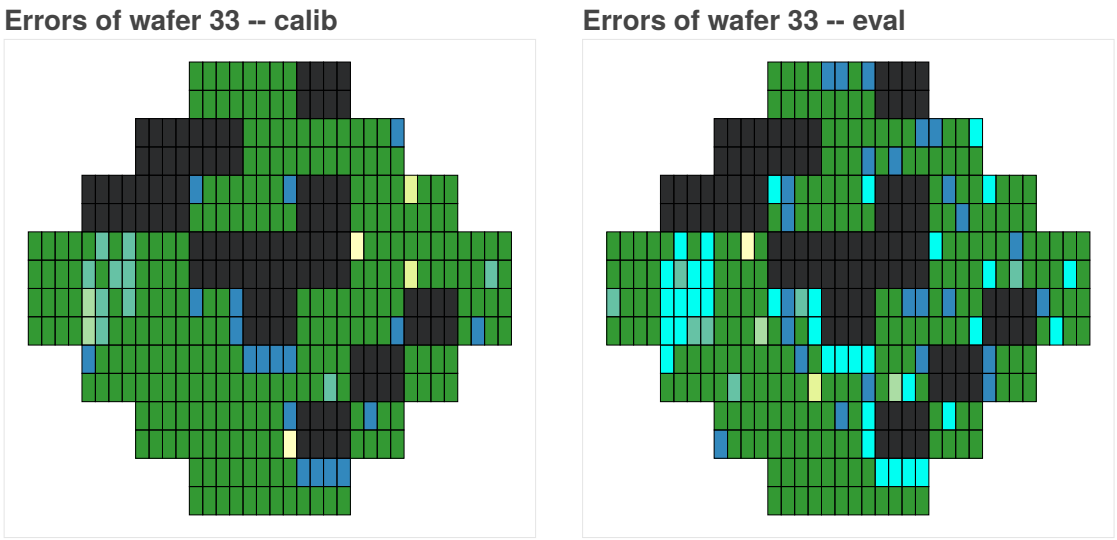

Figure 3.9.: **2D wafer view of the current state of the calibration (wafer 33).** Each small rectangle represents one HICANN chip. The errors are color-coded, separately for calibration and evaluation. The dark gray HICANNs and their corresponding FPGAs are turned off due to connection problems on the highspeed link. The weak preout signal error is always next to an uninitialized HICANN. It indicates that there is a problem on Layer1 (see section 2.2.1). As each HICANN has hardwired connections to its direct neighbors, it is expected that Layer1 fails for such HICANNs. Connection timeout can happen for many reasons, see the main text.

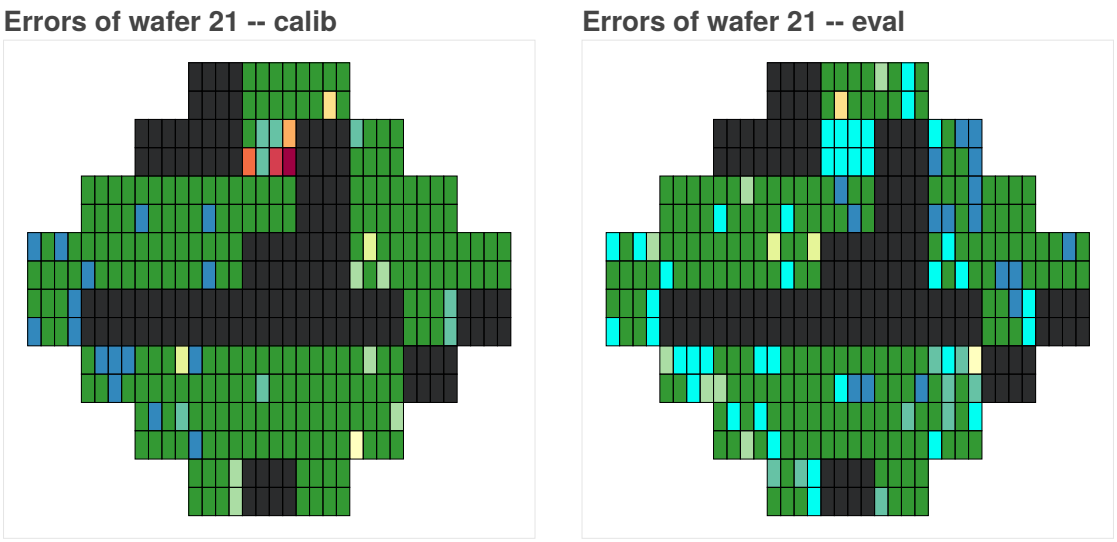

Figure 3.10.: **2D wafer view of the current state of the calibration (wafer 21).** For a description of the color codes, see fig. 3.9. One section of the wafer has a problem with an Analog Readout Board. Weak preout signals on this wafer almost always happen next to uninitialized HICANNs. Currently there is no explanation for the few HICANNs having a weak preout while being around initialized HICANNs.

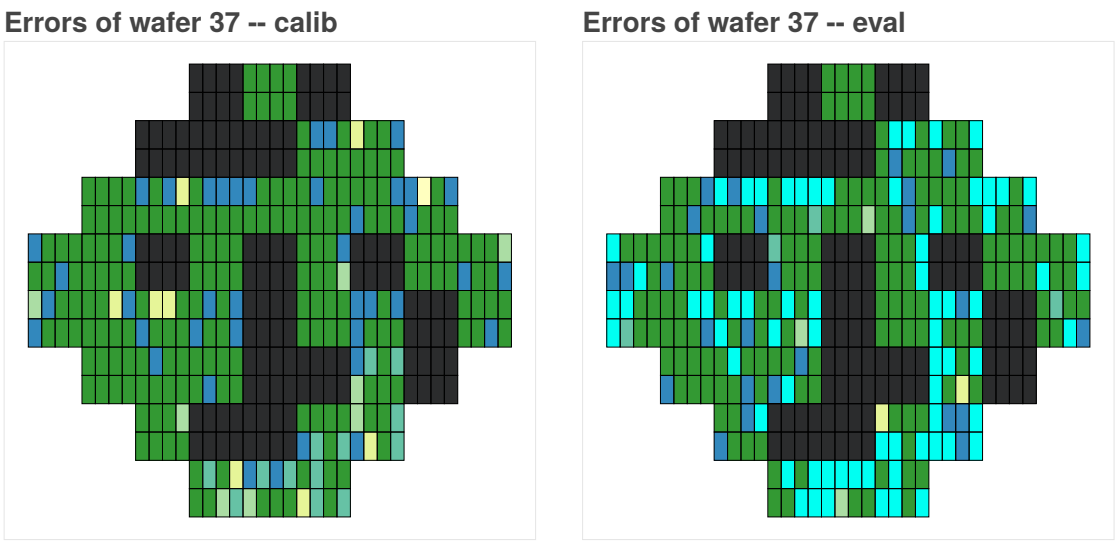

Figure 3.11.: **2D wafer view of the current state of the calibration (wafer 37).** For a description of the color codes, see fig. 3.9. The number of HICANNs that show the a weak preout are significantly higher for this wafer. The analog out of P2 bottom/master (c.f. fig. 2.3) does not work properly, explaining the Analog Readout Board errors.

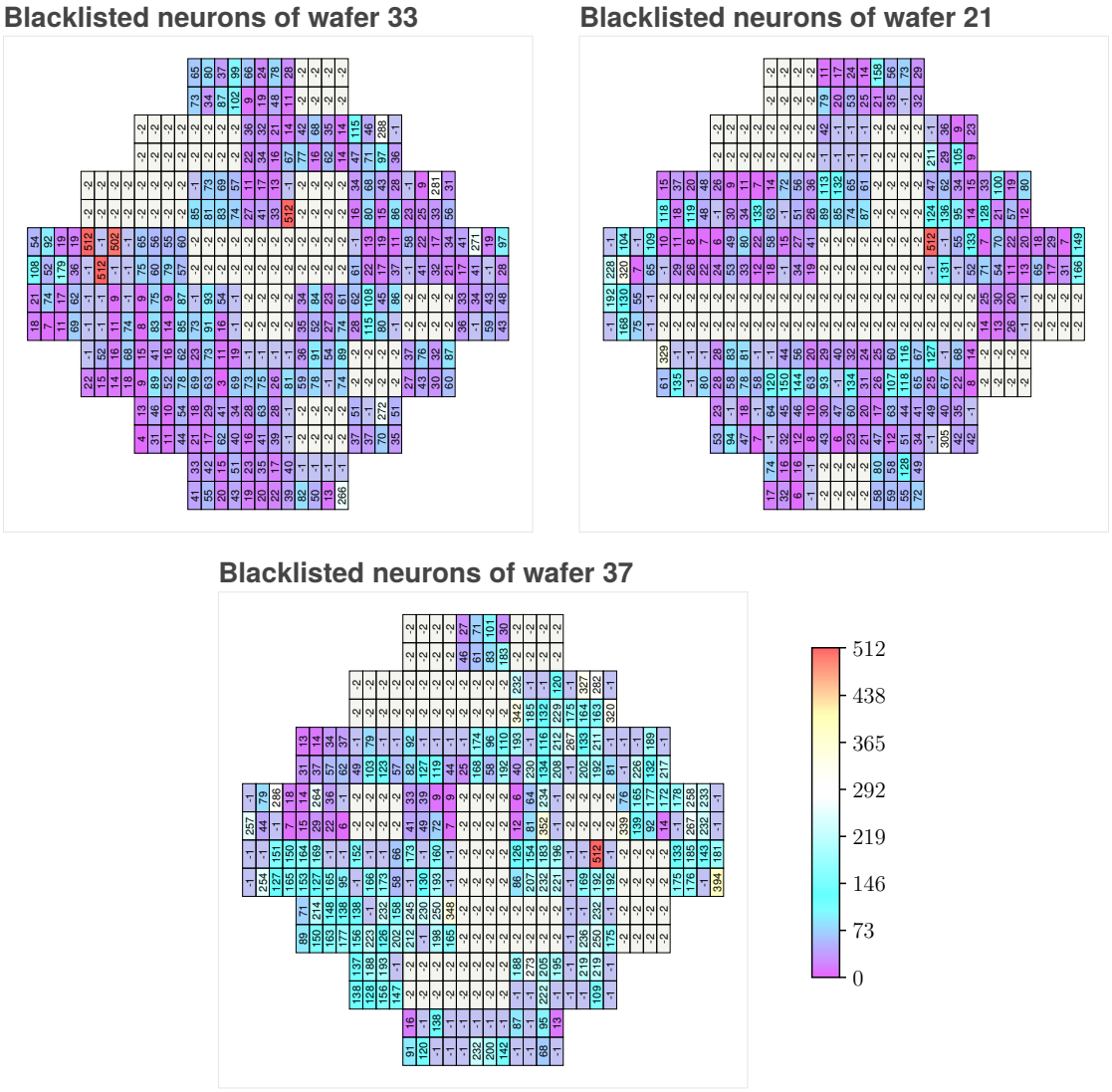

Figure 3.12.: **2D wafer view of the blacklisted neurons for the three wafers.** *Left top:* Wafer 33. *Right top:* Wafer 21. *Bottom:* Wafer 37. HICANNs that are shut down are marked with medium light gray and the number −2. Not yet finished calibrations are marked with a −1 and a light gray color. The color code encodes the number of blacklisted neurons from 0 (blue) to 512 (red) as depicted in the colormap. Reticle 16 (c.f. fig. 2.3) of wafer 33 has almost only blacklisted neurons. The distribution of blacklisted neurons shows no obvious structure. On wafer 37, the number of blacklisted neurons is significantly higher.

4. Constraint Satisfaction Problems

The following chapter describes the basics of Constraint Satisfaction Problems (CSP). One example for a CSP is the number-placement puzzle sudoku. A method to solve this problem using a neural network is presented. Simulations in software and on the BrainScaleS system are done to show its performance. The last section introduces an in-the-loop procedure similar to the one in *Klähn* (2017); *Schmitt et al.* (2017) to train the weights to increase the network performance on hardware for arbitrary sudokus. All times and voltages in the following sections will be in the biological domain, c.f. chapter 2 for the translation to the hardware domain.

4.1. Theory

4.1.1. Constraint Satisfaction Problems

The definitions in this paragraph are taken from *Dechter and Pearl* (1987, pp. 2-4). The constraint satisfaction problem is defined as a set of m variables $X_1, \dots X_m$ with domains $D_1, \dots D_m$ and relations $\rho \subseteq D_1 \times \dots \times D_m$. Binary constraints are a subset of the Cartesian product of two variables,

$$R_{ij} \subseteq D_i \times D_j. \tag{4.1}$$

The solutions of a network of binary constraints define the set

$$S = \{(x_1, \dots, x_m) \,|x_i \in D_i \text{ and } (x_i, x_j) \in R_{ij} \;\forall\; i, j\}\,. \tag{4.2}$$

Before discussing the solving strategy to find the set S, the definitions are applied to the case of the number-placement puzzle sudoku. This puzzle in its most general form is an $n \times n$ grid of cells, itself consisting of blocks with size $k \cdot l = n$. In its most popular form, $n = 9$. An example for an even smaller puzzle with $n = 4$ can be seen in section 4.1.2. Each cell is either empty or filled with exactly one number. The goal is to fill the empty cells, such that the following four constraints are fulfilled:

Rule 1 One of each number in each row

Rule 2 One of each number in each column

Rule 3 One of each number in each block

Rule 4 One number per cell

Bringing the concept of CSPs and sudokus together, each cell of the sudoku defines a variable X_i with its domain $D_i = \{1, 2, \dots n\}$. The sudoku rules define relations ρ, e.g for Rule 1 when $(X_1, \dots, X_p)$ are n cells in the same row:

$$\rho_{\text{row}} = \{(x_1, \dots, x_p) | x_i \neq x_j \ \forall \ i \neq j\} \tag{4.3}$$

and similar for the other rules. These relation can be split up in binary constraints, namely

$$R_{ij} = \{(x_i, x_j) | x_i \neq x_j \text{ for } i \neq j \text{ if } X_i \text{ and } X_j \text{ are in the same row, column or block}\}. \tag{4.4}$$

In the case of an unambiguous sudoku, the solution S contains only one element, the solution to the sudoku. Finding the solution S solves the CSP. Deterministic methods exist to find such solutions. However, depending on the problem at hand, it can be time-consuming to search for the solution in the solution space $\Omega = D_1 \times \dots \times D_m$. For general n, sudoku is an NP-hard problem. This means there is no known algorithm which can solve a sudoku in polynomial time, but given a solution, it can be verified if the solution is correct. A system, that generates solution from which some are correct is therefore a possible way to solve sudokus faster, as each solution can be checked in polynomial time. The next section presents such a method using a Spiking Neural Network.

4.1.2. CSPs and Spiking Neural Networks

A stochastic method using spiking neural networks is presented here, which is an adapted version of the network described in *Habenschuss et al.* (2013, pp. 9-13). It is only suitable for CSPs with a finite number of elements in each domain D_i. Thus, it is suitable to solve sudokus. The general idea is as follows: Each element in each domain of the CSP is represented by a neuron or a set of neurons[1], as depicted in the right panel in fig. 4.1. On taking a time average over the spiking activity of all neurons in a domain, the number associated with the neuron with the highest firing rate is taken as the suggested solution for this variable. Clues, which in the case of sudoku are the given numbers receive a high excitatory stimulus such that they spike with a high frequency. Constraints are set by drawing inhibitory connections between conflicting numbers. A random stimulus is applied to each neuron, leading

[1] In the aforementioned publication, the network is even more fine-grained, with one neuron per constraint per element per domain.

to an exploration of the solution space. Since conflicting numbers inhibit each other, it is expected that on average the network state will be very close to the solution. It is preferable to have a network that neither gets stuck in a certain state nor explores the states wildly, because then the correct solution can be found most reliably. This is achieved by letting each neuron excite itself, reducing the exploration and increasing exploitation. Summarizing, the tuneable parameters are:

- the number of neurons N_i per variable X_i
- the Poisson noise rate ν
- the rate for the clue neurons ν_c
- the strength of the connections between two neurons w_{jk}
- the neuron model, e.g. LIF, AdEx
- the neuron parameters
- the time window to average over Δt
- the method to average over Δt

An overview of the methods used to analyze the network dynamics is given in section 4.1.5. Additionally, the network structure can be varied depending on the problem at hand. The next section explains different structures and their scaling with the neuron size.

4.1.3. Network Structure

This section presents four different connections schemes that can be used to implement the CSP solver. The schemes are compared to the scheme used recently on SpiNNaker (*Fonseca Guerra and Furber*, 2017). Finally, the connection scheme is chosen to be the *standard* scheme presented in eq. (4.5) with one neuron per number per cell.

As an intuitive start, one can count the connections of one neuron in fig. 4.1 in the case that every sudoku rule is implemented one after another. The number of binary constraints is identical for every neuron, thus the connections per neuron can be multiplied by the total amount of neurons n^3 to get the total amount of synapses in the network, with $n = kl$ as defined above.

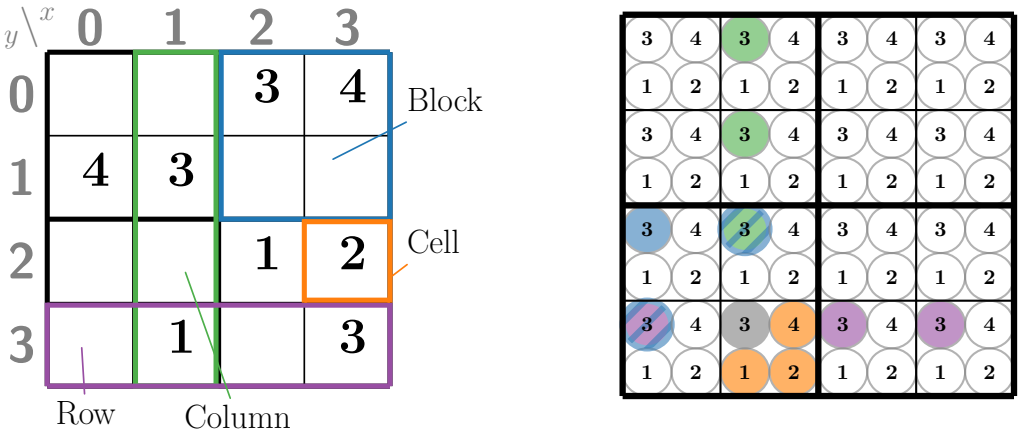

Figure 4.1.: **Sudoku structure and network.** *Left*: A typical 4×4 sudoku and its nomenclature. *Right*: The neural network as a representation of a 4×4 sudoku. The color code exemplary marks the inhibitory connections of the **gray neuron** to neurons in the same cell (**orange**), row (**violet**), column (**green**) and block (**blue**). Each neuron is connected in this way and additionally excites itself.

The same row, column, block or cell each contain $n - 1$ neurons that have to be connected. Additionally, the neuron can be connected to itself. This already gives

$$C_s = 4(n-1) + 1 \tag{4.5}$$

connections per neuron. Under consideration that some neurons are in the same row/column and block, this number can be reduced to the minimum amount of connections

$$C_m = 4(n-1) + 1 - (\frac{n}{l} + l - 2). \tag{4.6}$$

To randomly excite the neurons, an additional connection per neuron has to be drawn for both eq. (4.5) and eq. (4.6). Furthermore, it will prove necessary to inhibit each neuron externally (c.f. section 4.3), again adding one connection. In the case where each domain value should be represented by a set of neurons instead of one neuron, further possibilities emerge. In the simplest case, each set of neurons is all-to-all connected to each other set, in general

$$C_a = [4(n-1) + 1]N^2 \tag{4.7}$$

with N neurons per set (assuming the same number for each set). Then, $2N$ additional connections for individual excitatory and inhibitory external stimulus is necessary. Yet another option is to simply instantiate multiple sudoku networks in

parallel, with one accumulating neuron per variable X_i. The sudokus are *stacked* in the sense that they all perform in parallel without connections between the sudokus. The numbers of synapses when stacking t sudoku networks is

$$C_t = t(C_{\mathrm{m/s}} + 1). \tag{4.8}$$

Figure 4.2 shows the dependency of the number of synapses for the proposed models on the sudoku shape and the number of neurons per set. The number of synapses can vary several orders of magnitude between the different network structures. Although they scale equally with the sudoku size, it is still important to consider how many synapses are computationally affordable.
Especially on a system like BrainScaleS where synapses are realized with limited hardware resources, fewer synapses are preferable. The first two variants, *min* and *standard*, corresponding to C_m and C_s respectively, explicitly only allow single neurons to represent one domain value d_j of one variable X_i of the CSP. This of course can result in a more unstable network, as the dynamics are more limited if a single neuron competes with another neuron compared to two whole sets of neurons competing. Therefore, a tradeoff between set size and the number of synapses to be realised has to be made. In a recently published paper by *Fonseca Guerra and Furber* (2017), a network with $N = 25$ is realised on the SpiNNaker system which is able to solve 9×9 sudokus. They claim that the network size is mandatory to achieve stable solutions. In total, they utilize 36,675 neurons and 86,154,125 synapses, i.e. an average input count of 4700 synapses per neuron. It should be noted however that the actual number of synapses is smaller because some of the synapses are an artefact of the network topology of SpiNNaker, caused by the distribution of the neurons on multiple cores. These synapses have zero weight and effectively do not contribute to the system. For experiments on the BrainScaleS system it is desirable to start with a lower input count to make good use of the limited hardware resources. Implementing a minimum connection or the stacked sudoku network requires only $35N$ connections per neuron, compared to approximately 4700 synapses in the SpiNNaker model. In conclusion, the small network structure can be mapped to the BrainScaleS system. In this thesis, the connection scheme will be the "standard" connection scheme C_s, with one neuron per variable per domain value. The number of neurons will be n^3 with the number of digits per cell n.

4.1.4. Sudoku Difficulty Rating

When the network is used to solve sudokus, it is handy to know how difficult it will be for the network to find the correct solution. There are many ways to do this, and there is not an established way. Therefore, multiple methods are presented in

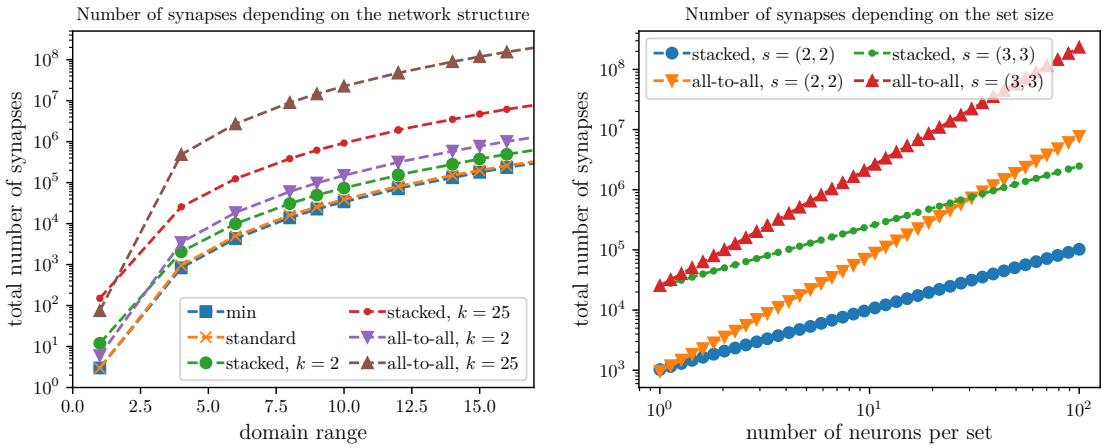

Figure 4.2.: **Synapse count for different network structures.** *Left:* Number of synapses per domain range. The domain range is defined by the possible numbers that can be put in a cell. That is 9 in the case of a 9×9 sudoku, and kl in the case of a $kl \times kl$ sudoku. The difference between the minimum number of connections and an all-to-all connected setup is about 3-4 orders of magnitude. *Right:* The two connection formula C_s and C_a are compared depending on the set size. For small set sizes < 3, the number of synapses is roughly equal, but for set sizes > 10 the number of synapses is at least one order of magnitude higher in the all-to-all method.

this section. An easy way to do this is to count the empty cells and set a limit to establish discrete sudoku difficulties. This is obviously a good method for a fast and coarse estimate of the sudoku difficulty: A sudoku where only one number is missing will be easier than one with over 25 missing numbers. Depending on the sudoku configuration though, it can still be easier to find a correct solution, when many fields allow only one number directly from the start. This can be accounted for by doing the rating by counting the number of possible numbers in each empty cell, depending on the starting condition, adding these numbers up and dividing by the total number of empty cells. This gives a number ≥ 1, where a difficulty rating of 1 is a very easy sudoku where all numbers can be filled in directly. This kind of rating is especially interesting, as it reflects the way the spiking neural network is going to solve a sudoku: In the case where only 1 number is possible in a cell, $n - 1/n$ neurons will be inhibited from the given numbers and in principle only one neuron will spike in this cell. This again will reduce the number of possible solutions. At a point where the network is in a state where no empty cell can be directly solved, the stochasticity of the input spikes helps to find the correct solution nevertheless. Figure 4.3 shows two 4×4 and two 9×9 sudokus, which will be solved in the following sections either in simulation or on the BrainScaleS system. Two more rating strategies exist, which are explained briefly. The first rates the sudoku depending

on the methods that are necessary for humans to solve them. An overview over the methods can for example be found in *Pelánek* (2014). One of the newest ratings was established in *Ercsey-Ravasz and Toroczkai* (2012), which uses chaos simulations to get the escape rate κ out of the probability $p(t)$ that the solution is not found by time t, which decays exponentially in hyperbolic dynamical systems. As this value requires to define a solver, it is not as general as the other ratings, and thus is not considered further.
As a last step, before using the BrainScaleS, methods to analyze the network dynamics are discussed in the next section.

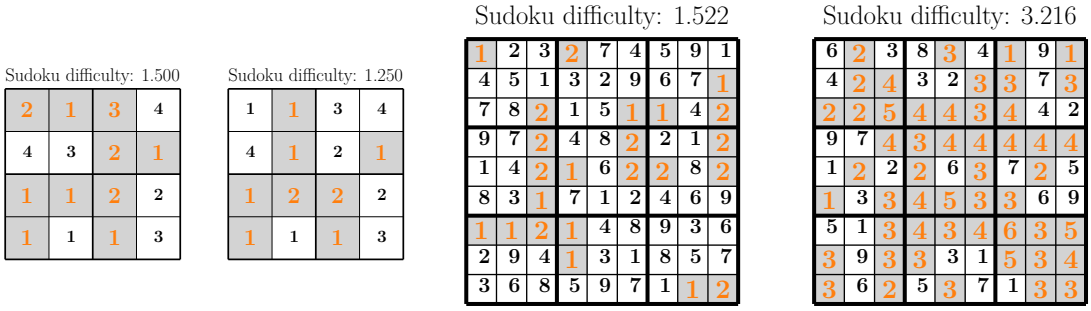

Figure 4.3.: **Sudokus and their respective difficulty.** The black numbers on white ground mark given numbers, called clues. The orange numbers on gray ground are the number of possible numbers that can be put in this cell under consideration of the clues. The difficulty is calculated by adding the orange numbers and dividing this number by the number of empty cells. The sudokus shown are solved in the following sections in simulation or on hardware.

4.1.5. Methods for Network Dynamics Analysis

In the last section it was already mentioned that an important step to utilize Spiking Neural Networks as CSP solvers is to interpret the spike of a neuron as "suggestion" that this domain value is the correct solution. As the dynamics of the network is stochastic, obviously wrong suggestions will occur and therefore a method to determine the correct solution out of all solutions has to be developed. A simple first approach can be to take the domain value of the last spiking neuron in each domain as the result. If the network converges to the correct solution and the network has already evolved some time, this is a sensible approach. However, it is unclear when "some time" exactly is and in addition, due to the stochasticity the network can in principle always explore new solutions (c.f. section 4.1.2). Taking averages reduces statistical fluctuations of the solution and there are multiple options to average over the spike times. The most general way is to define n time bins

$\{\Delta t_j : j \in \{1, \ldots, n\} \wedge \sum_j \Delta t_j = t_{\text{total}}\}$, and do for every variable of the CSP:

1. Measure the spiking activity, i.e. the number of spikes n_s for each domain value
2. Calculate the fraction of spiking activity for each domain value: $p_s = \frac{n_s}{\sum_i n_i}$

The result for a domain of size m are m probability-like values p_s which add up to one and can be interpreted as the "certainty" with which the network predicts the domain values. These values in turn can be used to calculate the information entropy (Shannon entropy) of the system. It is defined as (*Shannon*, 1948):

$$H = -\sum_k p_k \log_2 p_k = -\sum_{l=1}^{n_v} \sum_{s=1}^{m} p_s(l) \log_2 p_s(l) \tag{4.9}$$

with the number of variables n_v and all other variables as defined above. Its maximum is the logarithm of the number of random variables in the system $H_{\max} = \log_2 n_v m$ at $p_s = \frac{1}{n} \; \forall \; s$ and its minima are at $p_s = 1 \wedge p_j = 0 \; \forall \; j \neq s$ (see Appendix A for details). Calculating the Shannon entropy gives us information about the state of the system: If the value is far away from zero, the network explores different network states. If in turn it is very close to zero, the network does only explore very little. Also, one of the minima is the correct solution (see Appendix A). Therefore one could hope that the correct solution is found when the entropy converges to zero. However, due to the fact that the entropy has multiple minima it could also be that the network did simply not converge to the correct solution, for example when one cell of the sudoku puzzle is guessed wrong. Still, it is a valuable method to get information about the network dynamics. As a last remark, it is important to chose suitable time bins Δt_j such that for each variable of the CSP at least one domain value was active. Otherwise, all probabilities p_s will be 0 in that time bin. Therefore the time bins are chosen to be the maximum time distance of the spikes in each domain to ensure that all variables of the CSP have a solution at all times.

In the next section, the network is simulated in NEST with suitable parameters.

4.2. Simulation: Solving Sudokus with PyNN and Nest

This section describes the software simulation with pyNN and NEST of a spiking neural network being able to solve sudokus. This is done as a preliminary study to the implementation on the BrainscaleS system.

To simulate the network, the abstract description in the previous section has to be put in concrete terms. First, the neuron model is chosen to be the conductance based leaky integrate-and-fire model with an exponentially growing current close to the threshold. In pyNN, this model is called `IF_cond_exp`. This is a subset of the AdEx model on the HICANN chip, c.f. section 2.1.2. Neuron parameter settings are chosen to be compatible with the hardware parameter range as well. The neuron parameters used in this section are given in listing 1. They are chosen such that the neurons can be mapped on hardware and additionally such that each neuron has to receive more than one spike to spike while at the same time the traffic is kept on a moderate level. To achieve the second requirement, a suitable synapse weight was chosen. The synapse delay has been set to 1 ms, which is on the same order of the delay of the hardware synapses.

```
cell_params = {
    'tau_m'      : 30.0,    # (ms)
    'tau_syn_E'  : 5.0,     # (ms)
    'tau_syn_I'  : 5.0,     # (ms)
    'e_rev_E'    : 40.0,    # (mV)
    'e_rev_I'    : -100.0,  # (mV)
    'tau_refrac' : 5.0,     # (ms)
    'v_rest'     : -21.0,   # (mV)
    'v_reset'    : -21.0,   # (mV)
    'v_thresh'   : -10.0,   # (mV)
    'cm'         : 0.5}     # (nF)
synapse_weight = 0.014 #(mu s)
synapse_delay = 1. #(ms)
```

Listing 1: **Neuron parameters used for all neurons of the sudoku network.** The synapse delay is of the order of the synapse delay on hardware ($\approx 5\,\mathrm{ms}$). Neuron parameters are chosen such that they are in the range of the hardware neuron parameters. The difference between V_t and V_rest together with the synapse weight is set heuristically, such that the network dynamics are maximized (many spikes until spike) while at the same time the total bandwidth is not too high.

The network structure is the "standard" connection scheme, as explained in fig. 4.1. Recurrent excitatory connections are set for all neurons.

As a starting point, a 4×4 sudoku will be solved. The input spike rate for neurons that represent a given number of the sudoku is set to 180 Hz, that for unknown numbers to 70 Hz. The spike pattern is regular for the neurons that represent given numbers and Poisson noise for all neurons representing a number in an empty cell. Due to the these rates in combination with the neuron parameters and the synapse weight, neurons of given numbers show a spike rate of $(83.5 \pm 2.5)\,\mathrm{Hz}$. Neurons

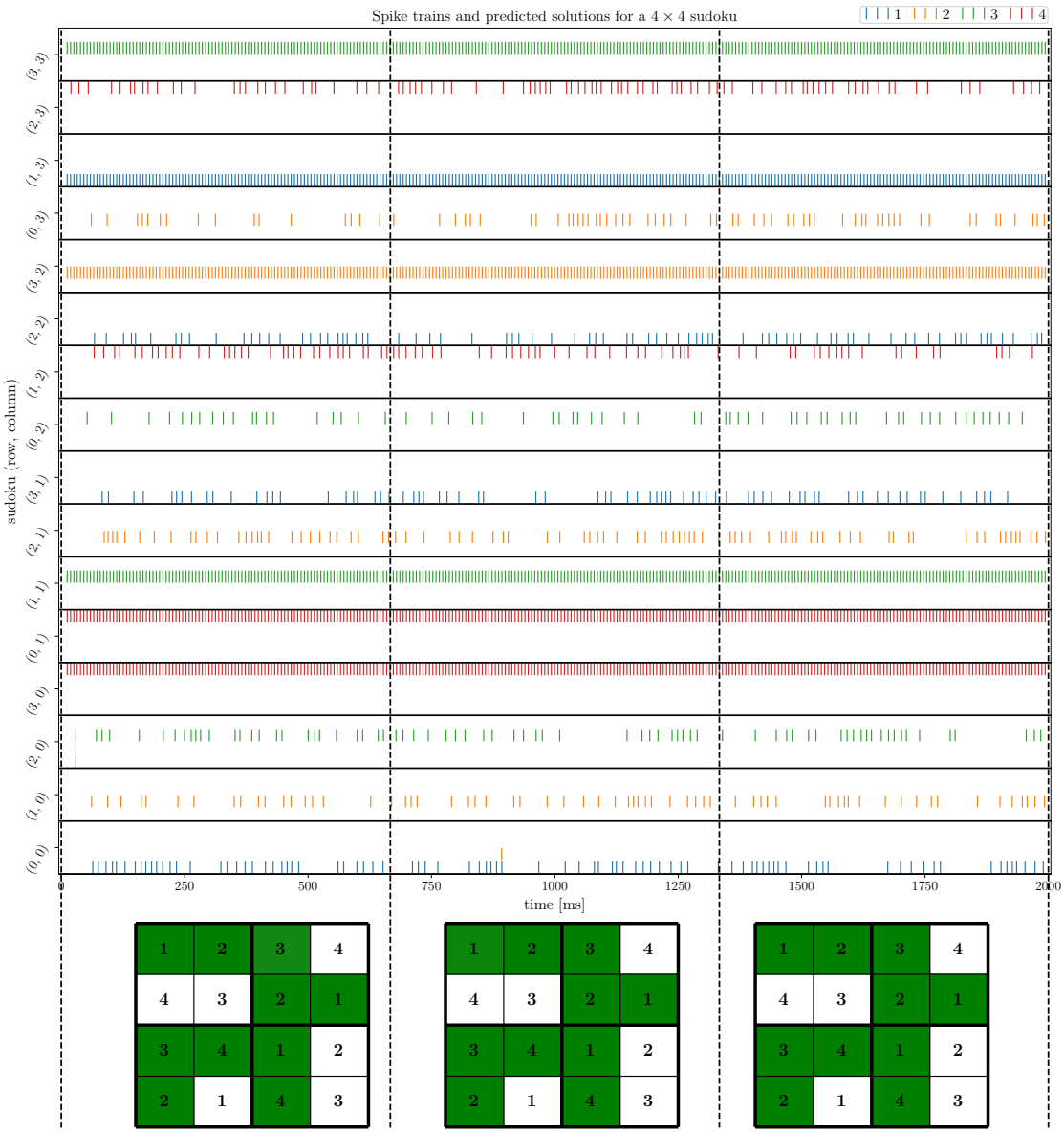

Figure 4.4.: **A** 4×4 **sudoku in simulation.** The black horizontal lines mark the different cells of the sudoku. When a number is already given, the corresponding neuron spikes steadily (e.g. $(3, 0)$). In free cells, up to three out of the four neurons spike, but on average, the neuron representing the correct number spikes the most (e.g. $(2, 0)$). This is also depicted with the transparency of the color in each cell: The lighter the cell, the lower is the relative spike rate of the correct neuron compared with all neurons in this cell, c.f. section 4.1.5. White cells are given numbers. The time averaging window in this example is 667 ms.

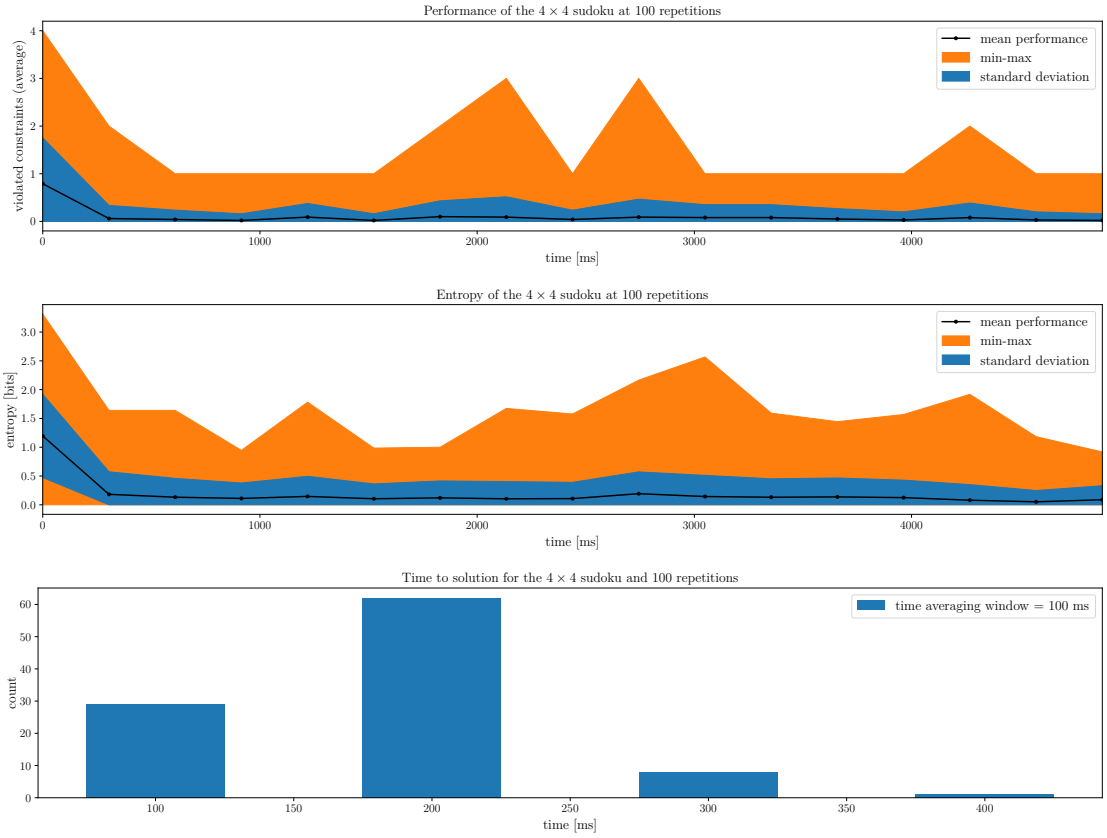

Figure 4.5.: **Performance of the** 4×4 **sudoku in simulation.** The performance is defined as the number of incorrectly filled sudoku cells. The time average was taken such that even the sudoku with the slowest dynamics did not contain any empty cells at any time step. 100 repetitions with different seed have been recorded. For the same time step size, the entropy is calculated as described in Appendix A. The entropy together with the performance show that the network converges very fast to the correct solution at about 200 ms, after which the entropy is approximately constant. This implies that the network still explores new solutions, which is due to the fact that the random input stimulation is not decreased with time. To find the first time when the correct solution is found, a smaller time averaging window is chosen. It can be seen that over 90% of the time, the correct solution is found in the first 200 ms.

in the same cell as a given neuron receive no external input. The spike trains for a specific sudoku as well as the predicted solutions are shown in fig. 4.4. The predicted solutions are calculated by taking regularized rates (c.f. section 4.1.5) in time intervals of 667 ms, which was chosen for a simpler depiction of the sudoku evolution. Figure 4.4 shows the spikes of all neurons, the colors encode the number the neuron represents and black horizontal lines divide the different cells. The black vertical lines mark the points in time where an average is taken and the sudoku depicted below the raster plot is the current prediction in this time frame. White cells correspond to given numbers, colored cells to unknowns. If the prediction is correct, the cell is green, and red if it is wrong. The transparency of the color is the probability, c.f. section 4.1.5. For example it can be seen in cell $(2, 0)$ in the first part that the green is lighter compared to $(1, 0)$, as neurons $1, 2$ and 3 spiked in cell $(2, 0)$, while in the latter cell, only neuron 2 spiked.

To analyze the network dynamics, 100 repetitions have been simulated with different random seeds, which determine the input spike times for unknown numbers. Results are shown in fig. 4.5. The performance as well as the entropy converge after about 500 ms to a stable value close to zero. At least one sudoku is always wrong at a time step (orange area), but the low number of violated constraints indicates that close to all sudokus are solved at each time step after 500 ms. The low entropy value shows that the network is stable in the solution found and other possibilities are not much explored. As the input spike rate does not decrease over time, some entropy will always be present. The bar graph at the bottom shows the time to solution for 100 repetitions. In 91/100 cases the solution is found in the first 200 ms. The time averaging window of 100 ms was taken as low as possible without introducing too many evaluations where a sudoku cell is not filled with any value because no neuron of that cell spiked.

Overall, the network is able to find the solution very fast. This was expected due to the small solution space. On a closer look, the sudoku under investigation has six cells where 3/4 neurons are inhibited by clamped neurons ($(1, 0)$, $(3, 1)$, $(0, 2)$, $(1, 2)$, $(0, 3)$ and $(2, 3)$), three cells where only two numbers are possible ($(0, 0)$, $(2, 1)$ and $(2, 2)$) and only one cell where three numbers are possible at $(2, 0)$. Therefore, in the next paragraph the network is scaled up to solve 9×9 sudokus.

The rate for known numbers is increased to 360 Hz to push the network faster to the correct solution and stabilize it. The output rate then is (118.1 ± 1.8) Hz. Unknown numbers are stimulated with a rate of 120 Hz. Simulation time has been increased to 2000 ms. The wall-clock runtime is about 20 s. The sudoku has 22 free cells and can be easily solved, but two correct solutions are possible. Therefore, it is expected that the network will alternate between the two solutions. Figure 4.6 shows the predicted sudokus over time. Green cells again mark correct

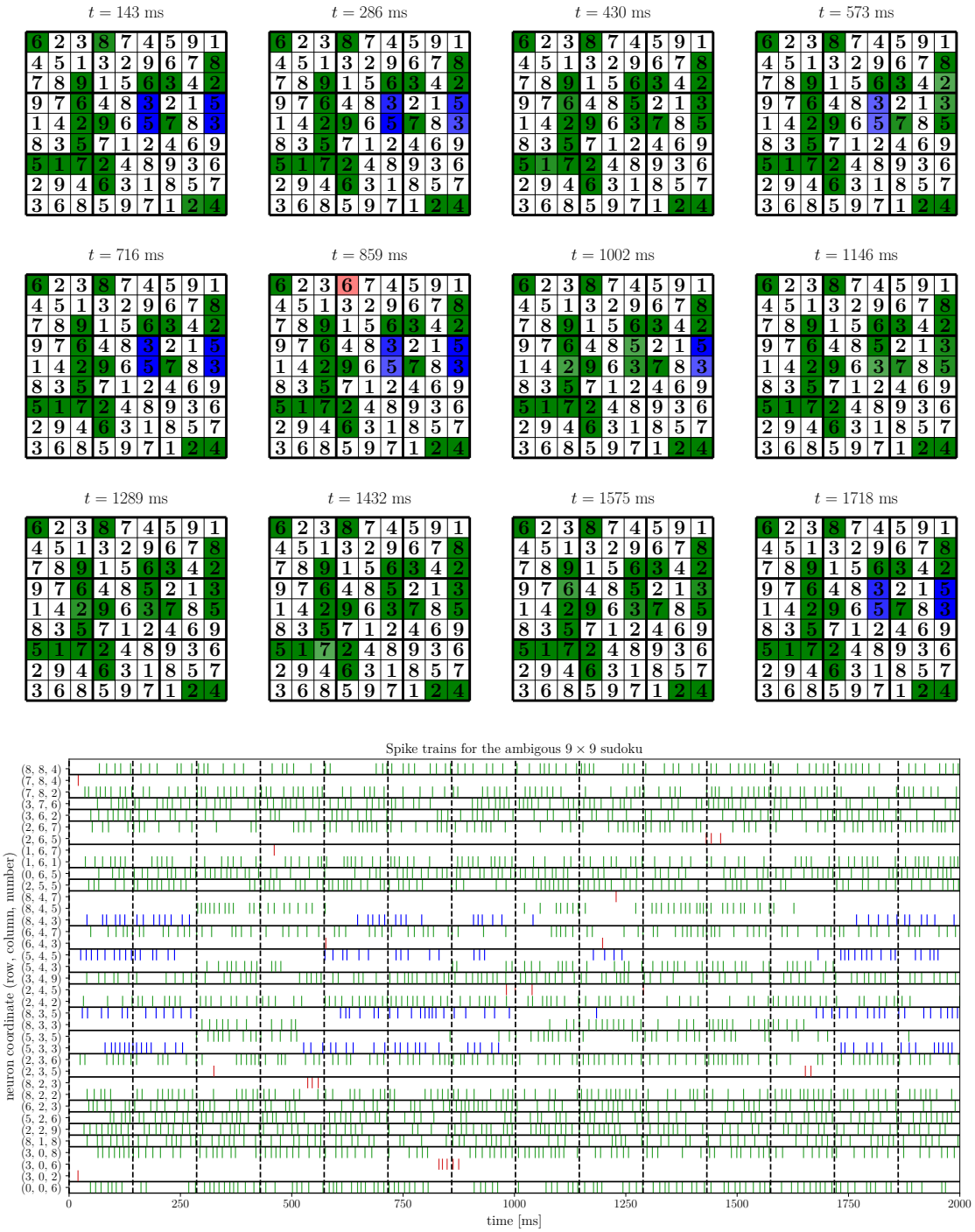

Figure 4.6.: **The ambigous 9×9 sudoku in simulation.** Two solutions are valid and, as expected, the network switches back and forth between the two solutions. If all cells are green, the sudoku is in one solution state. The four blue cells mark the second valid solution. From the spike trains shown on the bottom it can be seen that almost only the correct neurons are active and in the case where two neurons can be correct, the activity switches back and forth.

solutions, but in addition 4 cells are marked blue when they predict the second correct solution. The sudoku is correct, if either all cells are colored green or for cells are colored blue. The raster plot only shows the neurons that spiked at least once in empty cells. Black horizontal lines mark the different cells. Only in cell $(3, 0)$ three neurons are competing against each other, in all other cells only one to two neurons are spiking. This is due to the small amount of empty cells, which drastically limits the solution space. As expected, the sudoku prediction switches between the two correct solutions back and forth, with only one prediction being wrong at $t = 859\,\text{ms}$. Sometimes the transition from one solution to the other takes place in one time step, e.g. from $t = 1575\,\text{ms}$ to $t = 1718\,\text{ms}$, at other times the transition takes longer (e.g. $t = 430\,\text{ms}$ to $t = 716\,\text{ms}$. In the following section, the sudoku difficulty is increased but the solution to the sudoku is unique.

The parameters for this network are again increased to get more stable solutions: The input rate is 4 kHz, the Poisson spike rate 400 Hz and the simulation time is increased to 15 s. This increases the wall-clock simulation time for a single run to about 60 s. Figure 4.7 presents the predicted solutions for one of the 100 repetitions. It is 1/48 repetitions where a correct solution was found at all. With its 30 given numbers it is a medium sudoku, according to the rating described in section 4.1.4. The entropy, depicted in fig. 4.8, again decreases rapidly, indicating that the network gets stuck in a particular solution, and does only explore few new solutions, even if the solution is not correct. Also, the time to solution increases rapidly and the dynamics of the system decrease: The time averaging window had to be increased to 1000 ms to get solutions without empty cells for all 100 repetitions. The average solving time increased to about 6000 ms, excluding the 62 repetitions where no solution was found at all. It is expected that this behavior can be reduced by introducing multiple neurons per number and varying the random Poisson rate over time. On the other hand, this technique has proven to work in principle and therefore the next step is to implement the network to the BrainScaleS system. The simple network structure is an ideal starting point and can be scaled to more complex structures on demand.

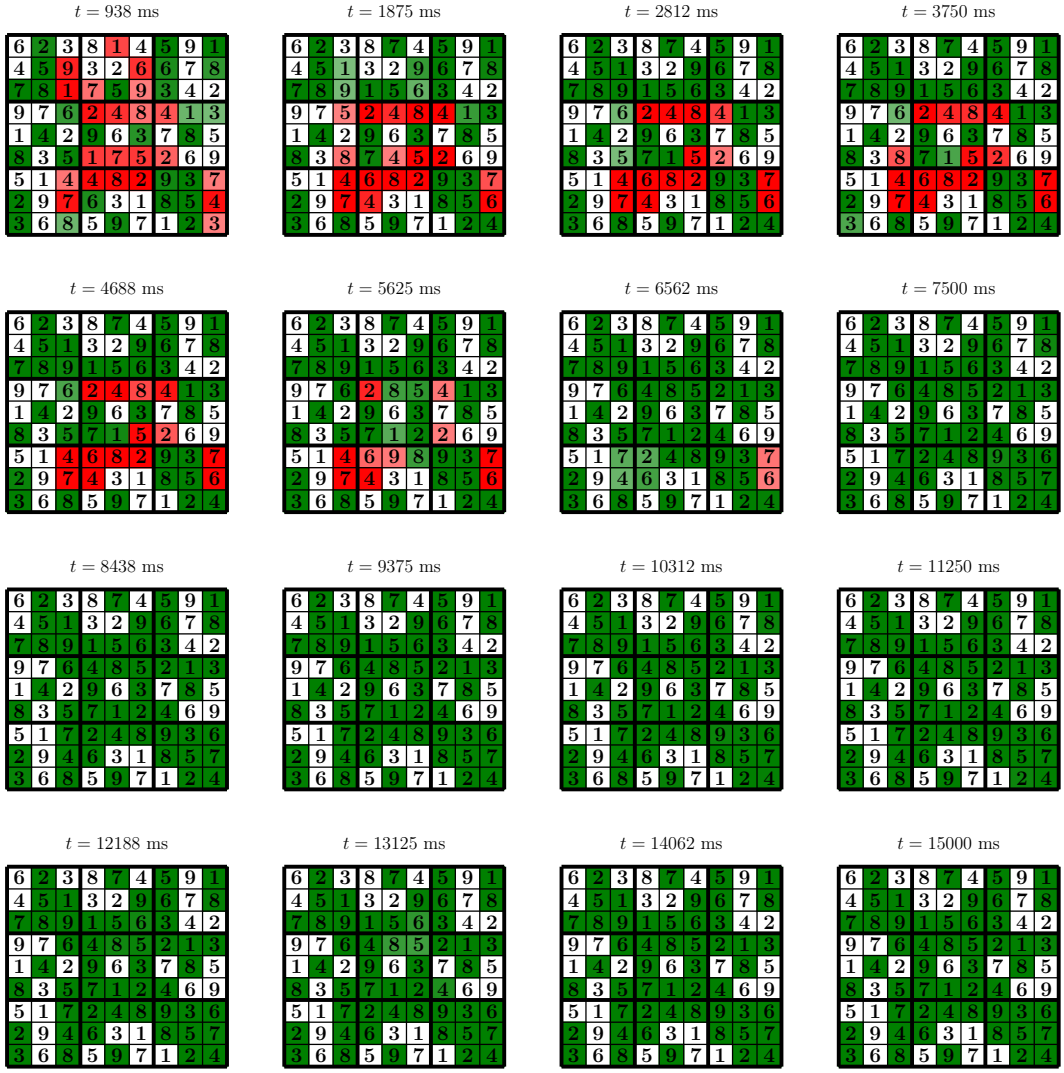

Figure 4.7.: **The "hard" 9×9 sudoku in simulation.** The transparency value indicates the ratio of the neuron that spiked most in the cell over the summed rates of all neurons in that cell. This sudoku with many empty cells can in principle be solved using the presented network. However, the results vary heavily depending on the random seed used to generate the Poisson noise.

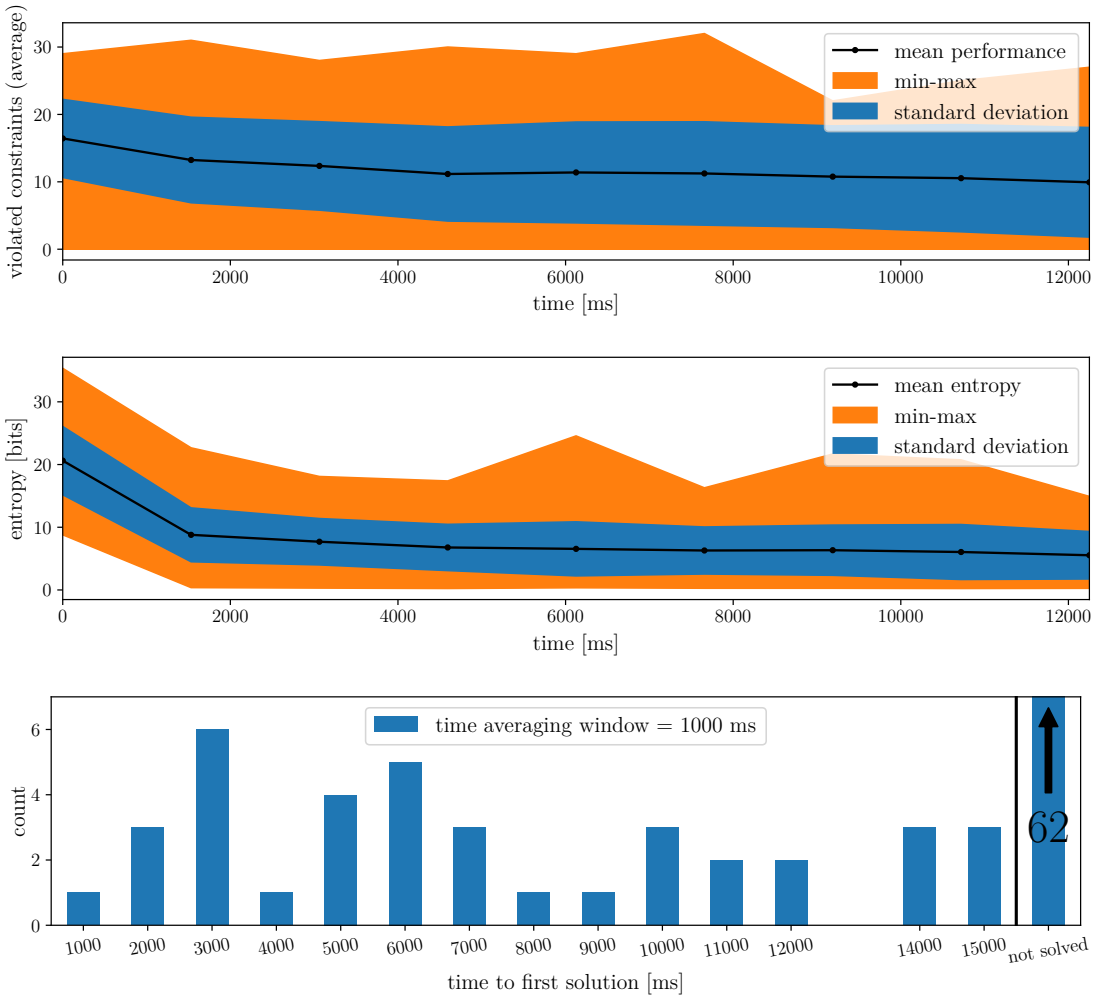

Figure 4.8.: **Performance of the "hard" 9×9 sudoku in simulation.** The performance is worse than in the case of the 4×4 sudoku. The time average again was taken such that even the sudoku with the slowest dynamics did not contain any empty cells at any time step, which in this case was much higher at about 1000 ms. 100 repetitions with different seed have been recorded. The network does not converge to the correct solution on average. The time to solution is now widely spread, with 62 tries that did not converge after 16 s. The entropy again decreases rapidly in the beginning, indicating that the solution space is only explored in the first 2 s. An approach to increase the exploration and thus to have a better chance to find the correct solution is to start with a higher rate and decreasing it over the course of the experiment.

4.3. Experiment: Solving Sudokus on the BrainScaleS System

In this section, the neural network is transferred from the NEST simulator to the BrainScaleS system. A few important differences to the software simulation should be noted: First, the neurons on hardware are inherently different due to transistor mismatch. Therefore, their neuron parameters will vary as well as the synaptic input strengths for each synapse. To get a general idea about these differences, multiple measurements are presented in the next section. Then, a first result using the network as is is presented. To improve the results, a training for the synaptic weights is discussed and presented in section 4.4. Finally, results are presented using the weight matrix from the training.

4.3.1. Premeasurements

This section describes how to set up the network correctly on the chips. Due to the bandwidth limitations between HICANNs and from FPGA to HICANN this can be a non-trivial endeavor. In addition, not all HICANNs are calibrated and not all reticles function for all wafers, as described in section 3.2. When only one experiment is executed with fixed input stimulus spikes, the mapping software can take care of the process of selecting enough HICANNs and FPGAs to distribute the input spikes. Then, all that has to be done by the user is to specify the HICANNs where the neurons should be placed. The following experiments however often require to change the input rates between two measurements. This can not be accounted for by the mapping software, but can be estimated manually by determining the highest possible input rate in the experiment and comparing it with the maximum value of section 2.2.4.

In the following, most experiments use approximately 200 Hz random Poisson-distributed input noise per neuron, and it is desirable to use only about 80% of the bandwidth to have some clearance. The amount of neurons that can be served by one HICANN thus are (c.f. section 2.2.4):

$$N = \lfloor 17.8\,\mathrm{MEv/s} \div 10^4 \div 200\,\mathrm{Hz} \cdot 0.8 \rfloor = 7. \tag{4.10}$$

This in principle does not mean that we have to use multiple HICANNs, as input can also be supplied by neighboring HICANNs. On the other hand this reduces the HICANN-to-HICANN connections, and therefore the neurons are placed on multiple HICANNs. The number of HICANNs needed can directly be calculated with the total number of neurons n_{tot} to be $N_{\mathrm{HC}} = \lceil \frac{n_{tot}}{N} \rceil = \lceil 9.14 \rceil = 10$. Distributing the

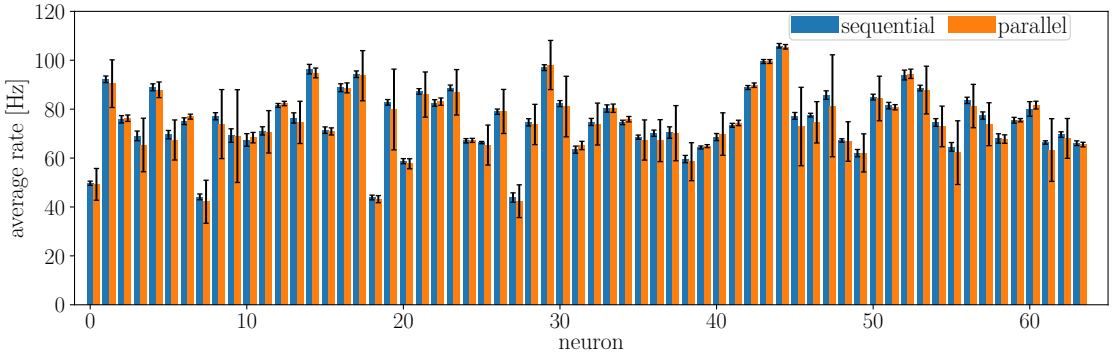

Figure 4.9.: **Spike rates for 64 neurons under regular stimulation with** 200 Hz **on hardware.** The measurement procedure is as follows: In a first step, each neuron receives the 200 Hz input sequentially. Then, all neurons receive the spike input at the same time. Comparing both output rates of the neurons allows to see if the neurons and external stimuli have been distributed correctly. The standard deviation is shown in gray. This bar gives additional qualitative information about the spike loss. The difference between rates is at most 5 Hz, or 7 %. The different rates between neurons is caused by fluctuations in the neuron parameters and the synaptic input strength.

neurons should not introduce any bandwidth problems, as the bandwidth between HICANNs is higher than from FPGA to HICANN and it is not expected that the network activity is as high as the input stimulus due to the inhibitory connections. To confirm that the external stimulus arrives at all neurons, a measurement is conducted once when a set of HICANNs is selected: Each neuron receives the maximum stimulus, first one after another and then all at once. This stimulus is linearly spaced instead of random to have better control over the results. An exemplary result can be seen in fig. 4.9 where the 64 neurons of a 4×4 sudoku have all been measured. The rates between different neurons vary from about 40 Hz to 120 Hz, which is expected due to different neuron parameters on hardware and fluctuations in the strength of the synaptic inputs. Both are caused by manufacturing inaccuracies as explained in section 2.1.2. The variations of one neuron between parallel and sequential spike input is at maximum at about 15 Hz. Including the standard deviation, the rates of the parallel measurement are always in a 3σ range to its sequential pendant and therefore the system is considered sane.

A second important precaution is to always verify that all repeaters are locked and all digital tests pass at least once when a distinct set of HICANNs is selected. This test checks among others the repeaters, the synapse decoders and the synapse weight memory. These can be faulty and will lead to spike loss or problems with the synaptic input. The software to blacklist them and ignore them is still under

development. So, the test has to be done manually and if it fails, a new set of HICANNs should be considered to conduct experiments on.

4.3.2. Results

After determining a set of HICANNs without errors and enough bandwidth, the network can be set up for a first experiment. In the following, wafer 33 is used together with HICANNs 296, 297, 298, 320, 321, 322, 323, 341 and 342, coordinates as depicted in fig. 2.3. Three adaptions are made to facilitate the task for the network: First, it can be that there are spikes on the communication layer from a previous experiment, or the voltage traces are at different starting values. This can be compensated for by sending inhibitory spikes to all neurons in the network. Second, the neurons representing given numbers receive their input earlier as the other neurons. This drives the network faster to the solution, as neurons that are directly constrained by a given number are strongly inhibited before they are excited and thus will never fire. The last adaption concerns the sudoku itself: it is simplified to have 8 empty cells, where 6 cells only allow one neuron to spike, i.e. 3/4 neurons are directly inhibited by given numbers in these cells. 2 cells, $(1, 2)$ and $(2, 2)$ allow two neurons to spike. Rates are set to 200 Hz for given numbers and 130 Hz for unknowns. The inhibitory spike rate at the beginning is set to 120 Hz and the rate for the given numbers before the random noise starts is set to 180 Hz. The network is inhibited for 500 ms, followed by a pause of 1200 ms. Then, the actual experiment starts and runs for 10 000 ms. The raster plot together with the predicted solutions is shown in fig. 4.10.

The three time phases can be identified by the colored vertical dashed lines: Until the yellow line, all neurons are inhibited. From the yellow to the green line, the given numbers are excited. And starting from the green line, the unknown cells receive random Poisson input. As in the simulation, in most cells only a single neuron, which is also the correct solution, spikes. However, in cell $(1, 2)$, neurons 3 and 4 compete and only in tiles 2 and 3, neuron 4 spikes more often than neuron 3. This is at first glance unusual, as neuron 3 in this particular cell is inhibited by 2 other neurons, namely $(0, 2, 3)$ and $(1, 1, 3)$, while neuron 4 is only inhibited by $(2, 2, 4)$. In addition, neurons $(2, 0, 3)$, $(3, 0, 4)$ and $(3, 3, 3)$ do not spike in the yellow to green tile, although they receive the same input as other given numbers. Both phenomena can be explained by the different synaptic input strengths of individual synapses on hardware. That exactly the two cells $(1, 2)$ and $(2, 2)$ show the most unstable behavior can be explained by comparing the sudoku with its difficulty rating in fig. 4.3. The two cells are the only ones that have more than one neuron which is not constrained by a clue neuron. As in the simulation, 100 repetitions

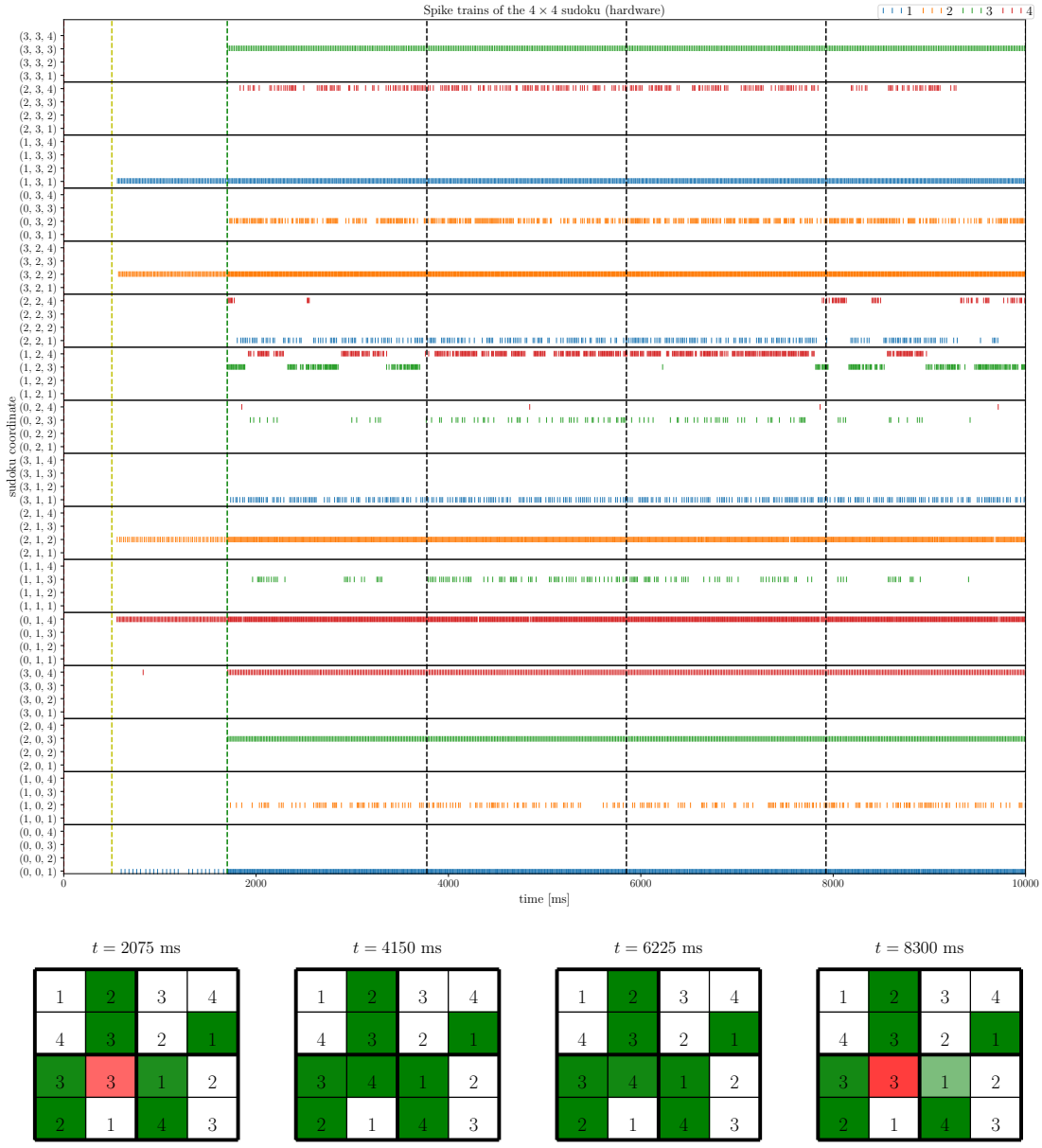

Figure 4.10.: **The** 4×4 **sudoku on hardware.** The raster plot shows the spikes of all neurons. Black solid horizontal lines divide the different cells, the number is color coded. The experiment starts at the green dashed line. Until the yellow line, all neurons are inhibited to ensure equal starting conditions. Between the yellow and green line, the clue neurons are stimulated. The sudoku on the bottom shows the solutions the network came up with, the time bins marked in the raster plot with black dashed lines. The transparency value encodes the relative number of spikes of the neuron that spiked most. For example the bright 1 of the fourth slice in cell $(2, 2)$ resembles the competition of neuron 1 and 4 in this cell.

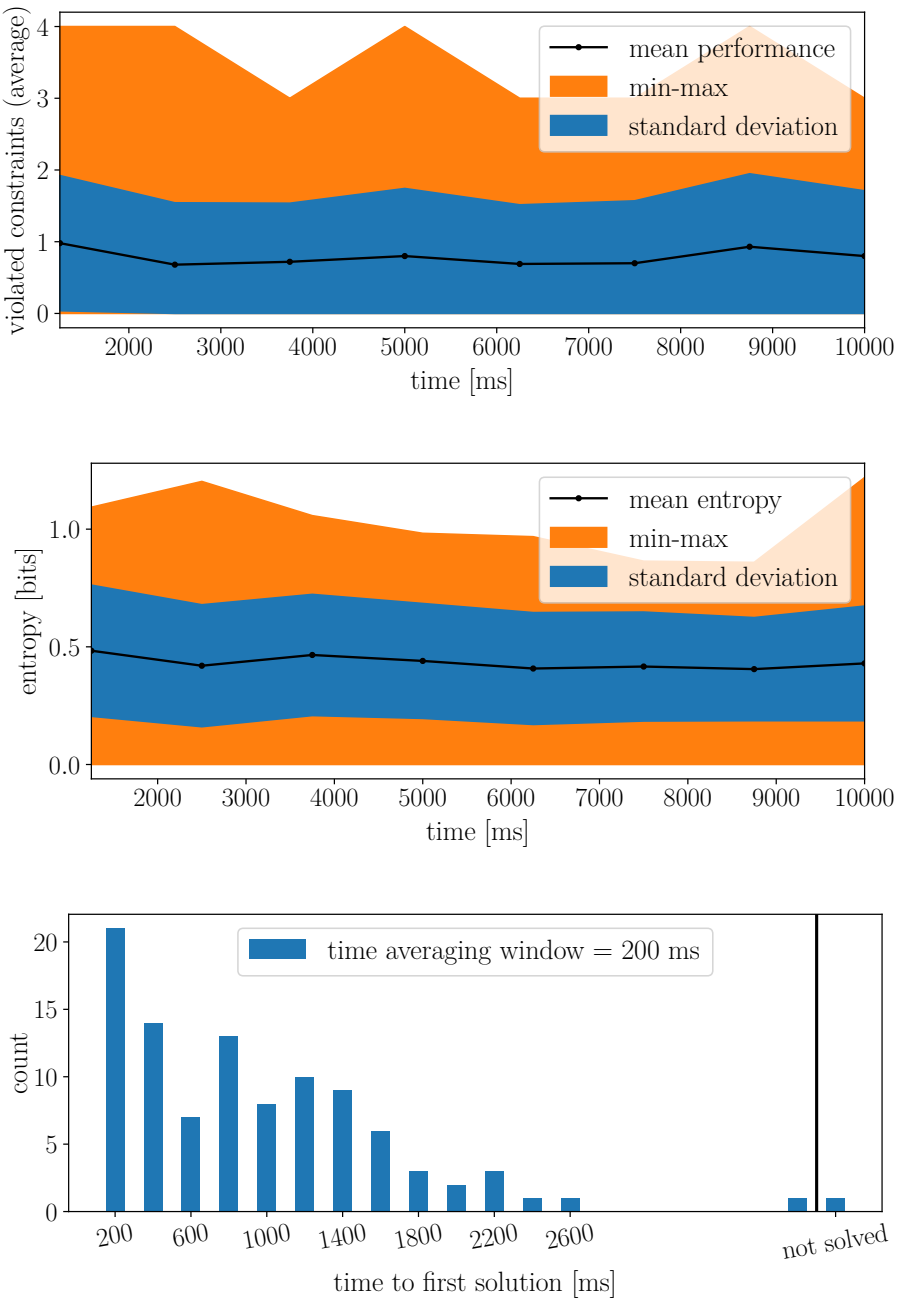

Figure 4.11.: **Performance of the 4×4 sudoku on hardware.** The mean performance is constant at about 1, indicating that the network is all the time close to solving the sudoku but one cell is often determined wrongly. The entropy is almost constant too and below 1, indicating that only very little solutions are explored. An entropy value close to 0.5 is an indicator that two neurons in one particular cell compete. The time to solution is higher than in the simulation, which corresponds to the higher number of violated constraints on average. 90% of the solutions are found after 2000 ms.

of the network dynamics are recorded to determine the performance, entropy and time to solution. Results are shown in fig. 4.11. The entropy and performance are approximately constant with values about 1 and 0.4, respectively. This is in both cases more than in the simulation, which decays in the first 500 ms to about 0/0.2. The standard deviation and min-max are higher for the performance on hardware, but of the same order for the entropy. The maximum value of the entropy is with 1.2 in fact smaller than the maximum value in simulation of about 4.2. The average value of 0.4 can be an indicator that all but one cell are solved correctly and in the last cell, two neurons compete. If the neurons spike approximately equally often, the entropy would be $-\sum_{n=1}^{2} \frac{1}{n} \log_2 \frac{1}{n} = 0.5$. The time to solution is more widespread than in simulation and 90% of the solutions are found after 2000 ms. The initiation time has already been subtracted from that value, so this is the actual solving speed. However, as the hardware runs 10^4 times faster than biological time, the actual result is still obtained very fast. In conclusion, the network on hardware works not as well as in simulation. This is expected, as the network configuration is not very stable against fluctuation of the synaptic input strengths, neuron parameter variations or spike loss: In the best case, all neurons should be exactly equal, as they all should compete equally against each other. Due to the fact that imperfections on analog electronics can never fully be controlled, another approach has to be found to make the network more stable against imperfections. A simple way to do so is to use one of the more sophisticated connection schemes described in section 4.1.3. This also would increase the number of neurons, but it is desirable to keep the network simple at first to be able to survey the whole network. Another option is to draw the same connection multiple times. This can average out the effect of individual synapses, but the number of synapses needed is unknown and can probably be high. As the endeavor is to keep the network simple, this option is unfeasable. On the other hand, the weights also control the synaptic strength and are until now all set to their highest value. The next section describes a method to tune the weights such that the synaptic input strengths are evened out and the network performs better.

4.4. Experiment: Pretraining of the Sudoku Network Weights

The last section demonstrated that it is in principle possible to solve 4×4 sudokus on the BrainScaleS system. However, the performance of the neural network is worse than in simulations. This is on the one hand caused by the lower input rate, which is bandwidth-limited from the FPGA to the HICANNs. However, it is assumed that also the variations between the neurons and between the synaptic inputs are a reason for the lower performance compared to simulation. Therefore,

a training algorithm is presented in this section that determines the best weights for all synapses in the sudoku to increase the solving capabilities. As a remark, the network is not trained to solve specific sudokus better, but such that the solution is found faster and more reliably for any sudoku.

The key of this training is to divide the whole network into small, independent and all-to-all connected networks. This can be for example all the neurons in the same cell. Each of these units can then be trained individually corresponding to the algorithm depicted in fig. 4.12. In essence, the all-to-all connected neural network is "unfolded" to a feedforward artificial neural network. This is done in four steps. First, N sets of input rates are defined. The neurons then receive Poisson-distributed spike input according to the rates. The N sets are used one after another with a small relaxation period between each set, where all neurons receive inhibitory spike input. As a result, we get N output spike rates. These values together with the input spike rates are put in an artificial fully-connected feedforward network (ANN) of the same network size as the spiking neural network unit. This network is implemented in TensorFlow, details can be found in Appendix B. Heuristically, the following approche was found to be viable: The input to the network is the summed input of the source neuron rates and the network neuron rates. The activation is overridden with the output rates instead of calculated from the input rates. And the input rates are transformed to a one-hot vector by taking the maximum and then set as the correct solution of the network. This effectively results in a network that tries to optimize its behavior such that only the neuron with the highest input spikes. For our CSP this in turn results in a high sensitivity to the sum of the random excitatory stimulus and the inhibitions from the other neurons. After inserting the rates as described above, weight updates for both the source weights and the network weights of the artificial neural network are calculated. These weights then have to be translated back to the weights in the spiking neural network. This step is non-trivial, because the weight range of the synapses in the ANN is not limited by default. On the hardware however, only digital weights from 0 to 15 can be set. One option to translate is to crop the artificial weights too, and then simply rescale. Another way to do the translation is to take the maximum weight of the ANN weights and translate it to the maximum hardware weight. All other weights are rescaled accordingly. This allows the artificial weights to steadily increase and decrease without limits. On the other hand, the weights on hardware may jump using this translation. Still, this translation proved to give better results in training and is therefore used. As in every iterative approach, the four steps of the weight update are performed multiple times, until a target classification rate is achieved. In principle, all network units representing the same sudoku rule can be trained at once, because they share no connections. As the rates used for training

are in the range of about 200 Hz per neuron, the bandwidth is limited on hardware and to have more control over the units, each unit is trained sequentially.

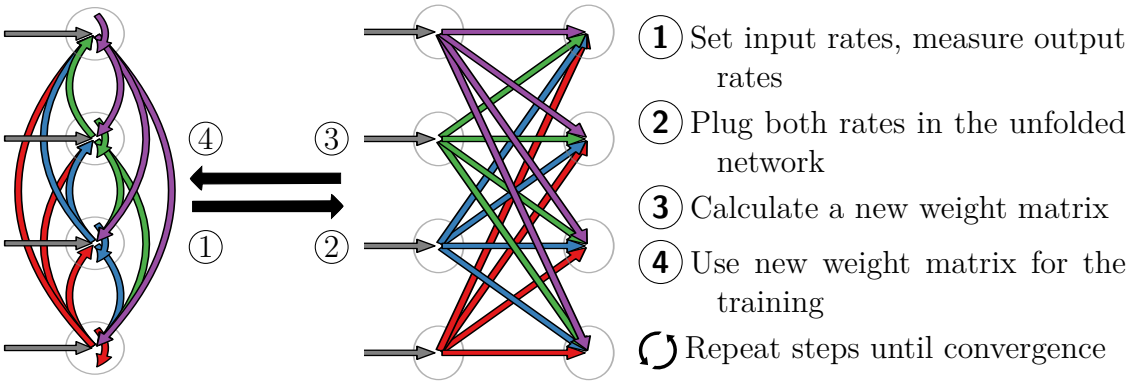

Figure 4.12.: **The WTA training.** Each independent unit of the sudoku (each cell, row, column or block) can be seen as an all-to-all connected inhibitory network with external stimulus. The objective is to train each unit such that the neuron with the highest input spikes and suppresses the other neurons. This is a form of a Winner-Take-All (WTA) network. The network is stimulated using Poisson noise with a constant rate for each neuron. The resulting output rates are measured and both rates are put in a single-layer ANN modeled in TensorFlow. The weights are then translated to hardware weights and the same experiment is executed again, until convergence is reached.

In the following, an example of the training will be illustrated and then results for all units are presented. Each unit is trained with fixed rates of 50, 100, 130 and 200 Hertz.

These rates are randomly distributed over the four neurons and then exponentially distributed [2] spike times are drawn for these rates. These spikes are presented to the network in 2000 ms. This is called a single batch. Multiple batches are recorded on hardware before the results are passed to the ANN. The chosen batch size is 400, with a pause of 200 ms between batches. During the pause intervals, all neurons are inhibited such that for every batch, the neurons are approximately in the same state. The total runtime on hardware therefore is 88 000 ms for each training step. Finally, the spike times are read out, cut in batches and passed to the ANN, as explained in the last paragraph.

Figure 4.13 shows the evolution of weights and correctness for unit 48. This

[2]The number of spikes in a given interval is Poisson distributed. One can show that the spike times then are exponentially distributed.

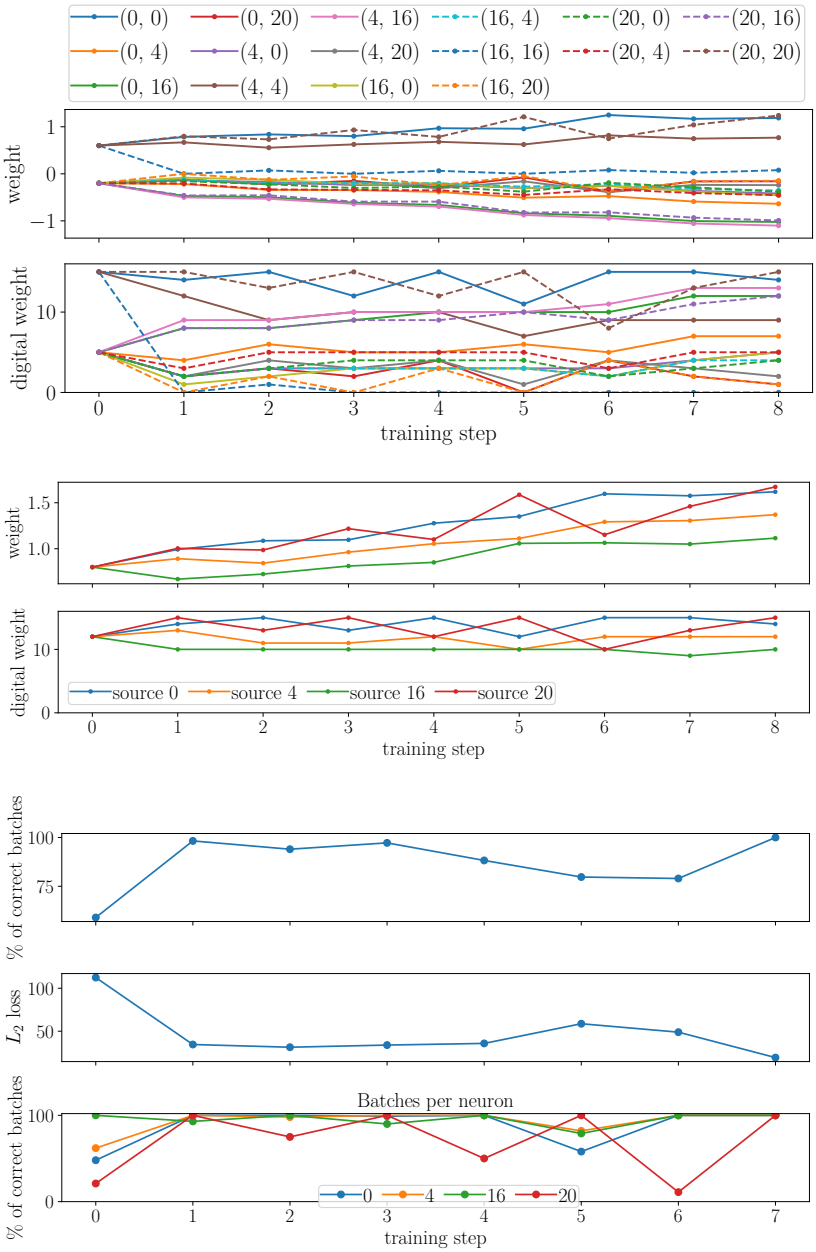

Figure 4.13.: **Weights and correctness for a single unit on hardware.** The unit consists of the four neurons representing a 1 in row 0. *Top*: Evolution of the weights of the ANN and on hardware. Due to the chosen weight translation method, weights on hardware can jump from 0 to 15 in one step, e.g. for $(16, 16)$ in step 1. *Center:* The weights of the sources that provide the external excitatory stimulus to the neurons. *Bottom:* Correctness, Loss and Correctness per neuron for each training step. Correctness is defined as the fraction of correct batches, where the neuron with the highest input rate also had the highest output rate.

unit comprises of the four neurons representing a 1 in column 0. The top two panels show the weights of this unit. The numbers in the legend are the unique IDs of the neurons in the network. The synapse (n, m) transfers spikes from m to n. Inhibitory connections in the Spiking Neural Network are represented by negative weights in the Artificial Neural Network. The excitatory self-connections are represented by the four lines above 0. The weight translation uses the maximum weight as explained in section 4.4. As a result, the hardware weights can jump in the full range from 0 to 15 in one training step. In the center panels, the weights of the external stimuli is shown. They are initialized at 11, such that the stimulus is sufficient to induce spikes. The last three panels enable to quantify the traning. The correctness is defined as the number of batches where the neuron with the highest spike rate (200 Hz) spiked the most. It reaches 100 % after 8 training steps and therefore the training is aborted. As expected, the correctness and the loss are anti-correlated. The last panel shows the correctness per neuron. It jumps by as much as 80 percentage points between single training steps. For the first training steps, the increase in correctness of one neuron leads to a decrease for at least one other neuron. This effect is expected as the correctness increases for one neuron if it is less inhibited, also leading to more inhibition of the other neurons, which can decrease their correctness if they are inhibited too much.

The example at hand shows that the training in principle works. The weights of the unit are tuned such that for the given input rates, 100 % of the time the neuron with the highest input rate fires the most. The correctness for all units using this training is shown in Appendix C. Next, it will be shown that the correctness for other than the training rates is increased for all units using this training method. Finally, the performance when solving sudokus is analyzed.

The devised test presents a set of rates to the network such that the highest rate is fixed and all other rates are sweeped. The set with the highest rates is $\{200, 200, 154, 77\}$ Hz. For each set of rates, 100 spike trains are drawn from a Poisson distribution per neuron. All units are tested sequentially to avoid too high traffic from FPGA to HICANN. Results for the trained and untrained case are shown in fig. 4.14. The x-axis shows the scaling factor between the highest and the second highest rate of the test set, because this is the main competitor for the tested neuron. The measured correctness is averaged over all units. After training the correctness improves significantly. It is constant at 100 % until 0.65, where before training the correctness already decreased to about 60 %. The standard deviation decreases, minimum and maximum are shifted towards higher values. At a scaling value of 1, two neurons have the same rate. It is expected that the network can only be correct about 50 % of the time, as both neurons are equally strong excited. The observed

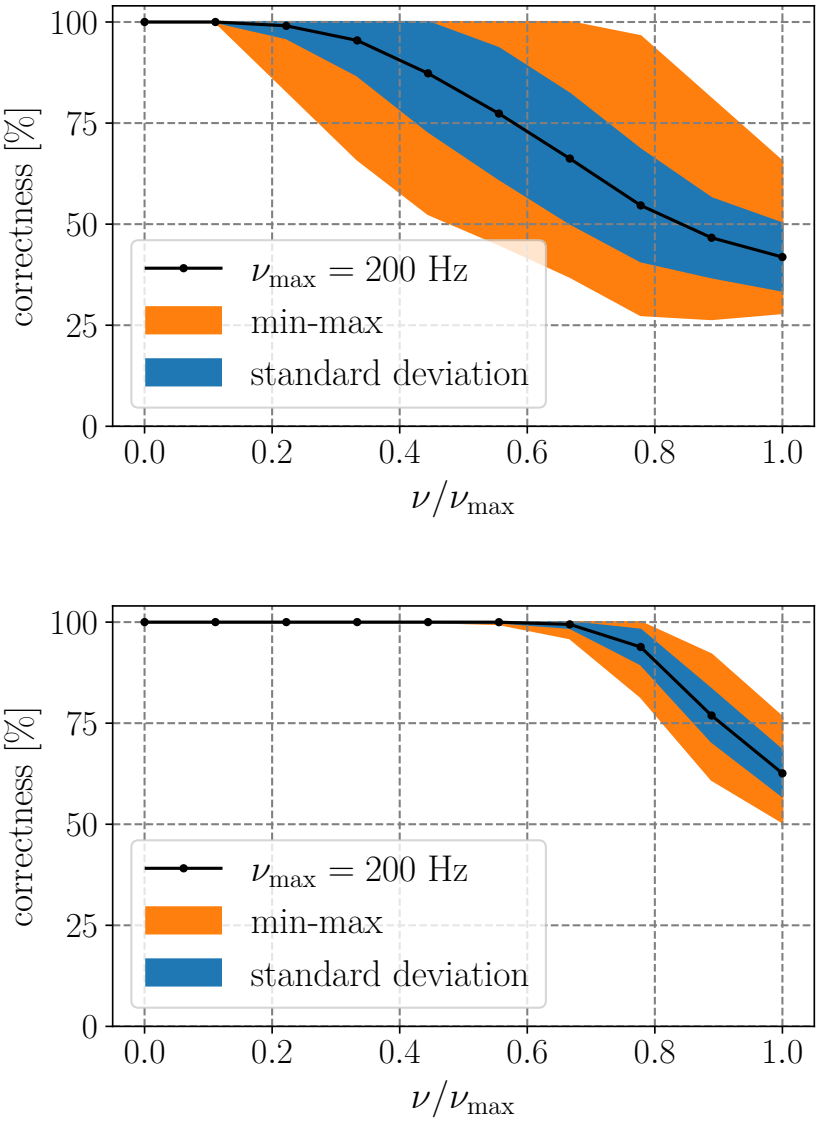

Figure 4.14.: **Correctness for given rates before (left) and after training (right) on hardware.** All units are stimulated sequentially with a set of rates, where all but the highest rates are sweeped. The maximum rates are $\{200, 200, 154, 77\}$ Hz. The training significantly improves the correctness in this test.

correctness is slightly higher, at about 60 %.
Summarizing, the devised test shows that the correctness can be improved for a larger set of rates than the units were trained for. The network with trained weights is therefore utilized in the next section to solve the sudoku from section 4.3 again. Furthermore, the solving accuracy is tested for multiple sudokus in sequence.

4.5. Experiment: Solving Sudokus with Trained Weights

In this section, the sudoku from section 4.3.2 is revisited, using the weights from the training in section 4.4. Furthermore, it is tested if multiple different sudokus are solved better on average with training.

The same setup as in section 4.3.2 is used, with the only difference that this time all weights are adjusted to the values determined in training. The raster plot and the found solutions to the sudoku are shown in fig. 4.16. As in the untrained case, the neurons first are inhibited and then the clue neurons are stimulated. In contrast to the untrained case, less clue neurons fire in this pre-experiment phase. Again, the spike times have been tiled in four parts from the beginning of the experiment, and the solutions found by the network are shown as sudokus with white cells for clues and red and green cells for wrong and correct solutions. The correct solution is found in all four slices. However, the individual cells are in part lighter colored, meaning that also wrong neurons spiked in that cell during that slice. An example is cell $(1, 0)$, where up to three neurons spike during one time step. This is in accordance with our training objective: The correct neuron is on average the neuron that is inhibited the least, and should therefore spike the most.

Average performance, entropy and the time to solution is presented in fig. 4.15. The performance with trained weights is close to zero. Also the maximum value is below the maximum value without training. All in all, the performance has increased because of the training. In contrast, the entropy is higher than in the untrained case, averaging about 1 at each step. This is in accordance with the observation that the training did not optimize such that only one neuron spikes, but rather such that the correct neuron spikes the most. The correct solution is therefore found much faster, 90 % of the solutions are found after 800 ms. The longest it took to find the solution was 1600 ms, also much better than the 4800 ms without training.

To show that the training did not only improve the solving speed and accuracy of this specific sudoku, multiple sudokus are tested for different difficulties. The number of empty cells of a 4×4 sudokus is at most 12. Therefore, 100 sudokus are

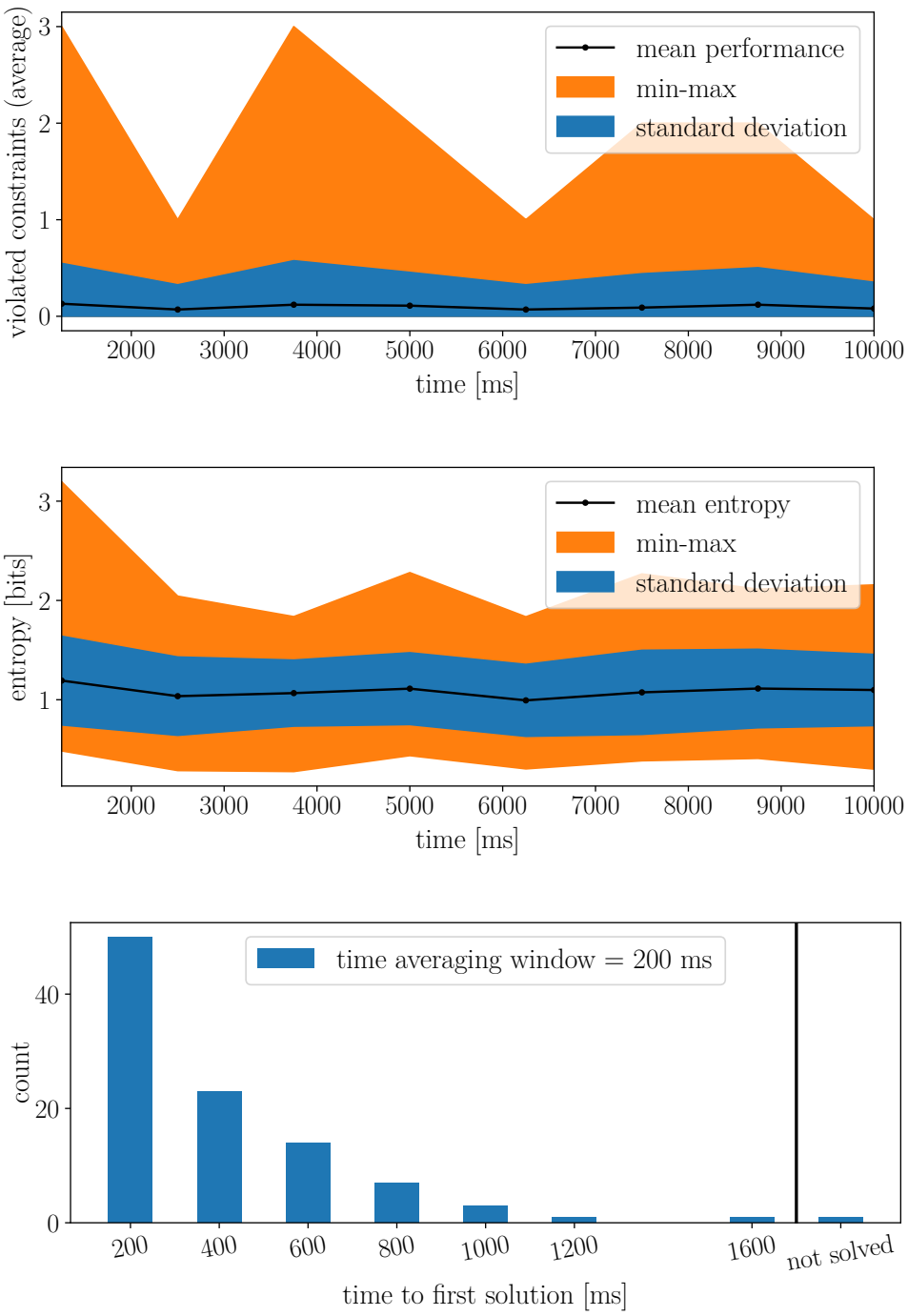

Figure 4.15.: **Performance of the 4×4 sudoku on trained hardware.** The mean performance is close to zero, implying that the network always finds the correct solution on average. The entropy is almost constant at about 1. The higher entropy results from more wrong neurons spiking. The time to solution is lower compared to the untrained network, but still higher than in simulation. Over 90% of the solutions are found after 800 ms.

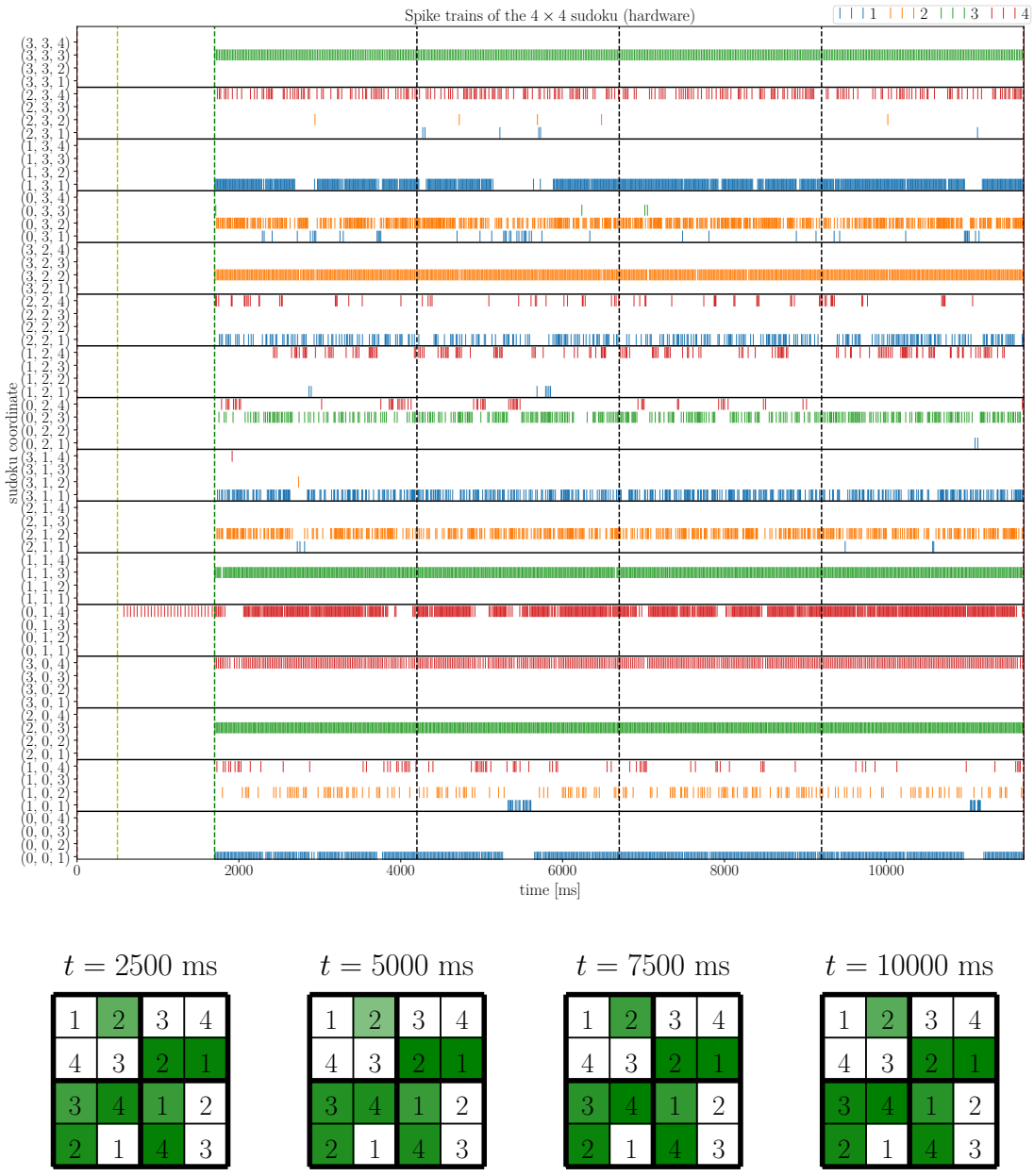

Figure 4.16.: **One example of a 4×4 sudoku on trained hardware.** *Top:* Raster plot of the spikes of all neurons. On the y-axis are the individual neuron coordinates of the sudoku. Additionally, the number that a neuron represents is color coded. As in the untrained case, all neurons are inhibited until the yellow line, to ensure equal starting conditions. Between the yellow and green line, the clue neurons are stimulated. *Bottom:* The solutions the network found. White cells are given. Green cells are correctly predicted, red cells are not. The transparency value encodes the relative number of spikes of the neuron that spiked most. The solutions are correctly predicted all times, but at the same time, wrong neurons spike more often than in the untrained case.

drawn at random for each possible number of empty cells from 0 to 12. Spiketrains are then drawn for each sudoku and the outgoing spiketrains are recorded and sliced. For each slice, it is determined if the solution presented by the network is correct. If at least one solution is correct, this sudoku is said to be solved. This is done for all 100 sudokus and all 13 numbers of empty cells. For this measurement, the floating gate has not been reconfigured after the training. Therefore, the neuron parameters did not change. The whole measurement procedure is done 4 times, results are averaged. In the untrained case, each weight is set to its maximum value.

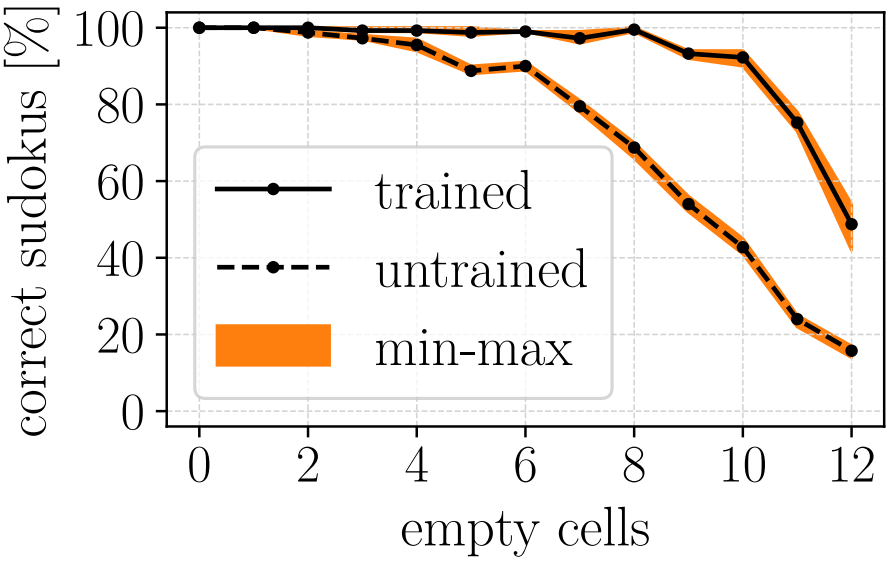

Figure 4.17.: **Correctly solved sudokus for different sudoku difficulties before and after training on hardware.** 100 sudokus are presented to the network for each data point. A sudoku is defined to be correct, when at least one of the produced solutions is correct. The x-axis shows the number of empty cells of the sudokus. The measurement was repeated four times with the same input spiketrains. In orange, the minimum and maximum value for each data point are shown. The variations for a single data point are always lower than ±5 %. The training improves the number of correct sudokus significantly. This measurement was conducted without resetting the floating gates after the training.

Results are shown in fig. 4.17. The percentage of correctly solved sudokus increases significantly with the weights from training. In the case of 0 empty cells, both networks perform optimally, with a correct percentage of 100 %. This is expected, as with no free cells, only the correct neurons are stimulated. Having less than 100 % correct sudokus is an indicator that spikes are lost. At 8 empty cells, the

untrained network accuracy declined significantly below 70 %, while for the trained case it is still close to 100 %. From 10 empty cells on, the accuracy also declines for the trained network, and ends at about 50 %. The untrained network accuracy in this case is below 20 %. Concluding, the network with trained weights vastly outperforms the untrained network at solving arbitrary sudokus.

5. Discussion and Outlook

This thesis accomplished to solve a Constraint Satisfaction Problem at the example of the number placement puzzle sudoku on neuromorphic hardware. To achieve this goal, multiple prearrangements have been realized. The refractory period was scrutinized in chapter 3 to ascertain the attainable precision of the calibration and in-experiment use. Also in this chapter, the calibration framework was scaled up to allow an efficient calibration of a whole wafer module of the BrainScaleS system. Three wafer modules were calibrated using the framework, allowing to conduct multi-HICANN experiments.
In chapter 4, a method is presented to solve general Constraint Satisfaction Problems. This method is utilized to solve specific instances of sudokus in simulation and on hardware. The dynamics of the Spiking Neural Network are investigated in detail.
It is observed that the network on hardware a priori does not perform as well as in simulation. This is accounted to the neuron parameter variations between neurons and variations of the synaptic input strength of each synapse, which is at the time of writing not yet calibrated. A training algorithm is presented which trains single, all-to-all connected subsets of the Spiking Neural Network to detect the highest input rate, hereby resembling Winner-Take-All networks. An Artificial Neural Network is set up with TensorFlow to train the weights of the Spiking Neural Network. After training, results are presented to show that the performance of solving sudokus improved with the training. Future experiments can benefit from this training algorithm as it is not tailored specifically to the network proposed here to solve sudokus.
In the following, an outlook is given for the calibration, the Spiking Neural Network to solve sudokus, the training algorithm and large-scale experiments extending the ones presented in this thesis.

Calibration

The result of scrutinizing the calibration of the refractory period it that it is only possible to get a sensible calibration in a small range due to the large variations between neurons. Furthermore, the trial-to-trial variations further restrict the calibration range. As an experimenter, one should always expect at least 30 % variation

of the refractory period for larger networks. However, if precise refractory period settings are needed the BrainScaleS still can be an option. It could be realized in the future to include the variations in the calibration framework, such that an experimenter can select the neurons with the lowest variation. Also, it is already possible to measure the refractory period of each neuron before conducting the experiment. As it is not necessary to rewrite the floating gates when conducting the same experiment multiple times (*Klähn*, 2017), it is sufficient to measure them once after setting the floating gates. In the newest version of the HICANN-DLS chip, the successor to the HICANN chip, the refractory period is digital, and therefore does not have distinct variations.

Scaling the calibration up to wafer-scale resulted in three of the newest wafers that can be used for large-scale experiments. In total, 656 HICANNs could be calibrated, from which on average about 80 % of the neurons have not been blacklisted. Large-Scale experiments are now in reach using tens of thousands of neurons. Tuning the calibration hyperparameters like the number of calibration steps it could be possible to further reduce the amount of blacklisted neurons. The time to calibrate a single wafer module was found to be about 10 days. Due to the pipeline, the user interaction during that time is on a moderate level. As the calibration can be stopped and resumed at any point, the calibration can be done overnight or when the wafer system is idle. Furthermore, as different wafer modules can be calibrated in parallel, the only limit for calibration time when scaling up is the number of host computers that run the calibration software analyze the data. Moreover, with the new 96 channel digitizer of *Ilmberger* (2017), the parallelization is expected to increase significantly.

Solving Constraint Satisfaction Problems

It could be shown that the sudoku-solving Spiking Neural Network simulated with NEST can be translated to the BrainScaleS system hardware. The complete software stack to implement an abstract neural network in PyNN and map it on the hardware had already been developed. Therefore, it is effortless to transform a network that is already implemented with PyNN and a simulator backend like NEST to the hardware.
However, the chosen network structure is susceptible to variations of the neuron parameters and changes in the strength of the synaptic input. As the BrainScaleS system uses analog neurons, parameter variation are inevitable. Therefore at first, the solving capabilities of the network were worse on hardware than in software.
To increase the solving capability of the Spiking Neural Network on hardware, a

training algorithm has been devised that trains small subsets of the network, called units. Although each unit is trained separately, it could be shown that the network performs better after training. For the maximum number of 12 free cells in a 4×4 sudoku, the percentage of correctly solved sudokus could be increased from below 20 % to 50 % for 100 randomly drawn sudokus. At 8 free cells, the solving accuracy with training still is at 100 % while it declines below 70 % without training. This is a vast improvement, especially regarding that the training algorithm is not designed to solve specific sudokus better.

Still, there is room for improvement. Manual tuning was necessary to find the optimal training hyperparameters. This could be circumvented by automatic tuning for example with Bayesian inference.
The focus of this thesis was to solve Constraint Satisfaction Problems in the case of sudokus. However, there are more CSPs that can be solved, for example the map coloring problem. Using the code developed in this thesis, it should be practicable to implement other CSPs quickly. The training algorithm can also be adapted to other CSPs.
The connection structures presented in section 4.1.3 can be tested to improve the performance of the Spiking Neural Network: It has already been shown in simulation and on SpiNNaker that a population of neurons per number is more efficient in solving sudokus. This would also naturally decrease the variations between different cells and in the same cell of the sudoku.
This thesis investigated only 4×4 sudokus on hardware, instead of the common 9×9 sudokus. The software and training however are already implemented for arbitrary neuron sizes. A suitable way has to be found to feed more neurons externally, for example by using the background generators on chip or a sea of noise population. Then, the network can be scaled up to solve 9×9 sudokus and the performance of the training can be investigated for a large network.

Appendix

A. Derivation of the Extrema of the Shannon Entropy

Finding the extrema can be seen as an optimization problem and therefore be solved using Lagrange multipliers. The function under investigation is the Shannon Entropy, taken for a single variable X_l of the CSP:

$$H_l = -\sum_k p_k \log_2 p_k \tag{.1}$$

with the constraint that the probabilities have to sum up to 1:

$$\sum_k p_k = 1. \tag{.2}$$

Implicitly, it is assumed that there is only a finite set of p_i's. Each probability p_k is seen as an independent variable. The Lagrangian then reads

$$L = -\sum_k p_k \log_2 p_k + \lambda(\sum_k p_k - 1) \tag{.3}$$

and its derivative after p_i is

$$\frac{\partial L}{\partial p_i} = -\log_2 p_i + \lambda - 1 \tag{.4}$$

As all λ have to be equal it follows that the extremum is at

$$\log_2 p_i = \log_2 p_j \ \forall i, j \Leftrightarrow p_i = p_j \tag{.5}$$

which can only be fulfilled if $p_i = \frac{1}{n}$ is uniformly distributed. This is the only extremum and actually the maximum which can be seen by simply plugging in any other value for the p_i's and observing that it is smaller. The value for the extremum can be simply calculated as:

$$H_{\max} = -\sum_{k=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = -n\frac{1}{n} \log_2 \frac{1}{n} = \log_2 n \tag{.6}$$

Now the boundaries of the function H_l have to be investigated. The boundaries for p_i are $[0, 1]$. The value for a term of H_l at the boundary is $H_l(0) = H_l(1) = 0$. As $H_l(x) < 0 \ \forall \ x \in (0, 1)$, it follows that for multiple p_i with the constraint in equation eq. (.2), the minima of H_l are:

$$H_l = 0 \text{ for } p_i = 1 \wedge p_j = 0 \ \forall \ j \neq i \tag{.7}$$

B. Artificial Neural Network Implementation in Tensorflow

The artificial neural network used for the pretraining is a one-layer, fully-connected and feedforward neural network. The neuron size is equal to the number of neurons per cell of the sudoku that the whole network represents. In mathematical form, it can be written down as (using Ricci calculus)

$$\nu_i = \sigma(w_{ij}\mu_j) \tag{.8}$$

with the activation function σ, in- and output rates μ/ν and the weight matrix w_{ij}. In the case that multiple batches are presented between training steps, ν and μ simply become matrices instead of vectors. The activation function used in the course of this thesis is the `tf.nn.elu` function, which is defined as

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \exp x - 1, & \text{otherwise} \end{cases}.$$

as a cost function, the L_2 loss is used, in TensorFlow defined in `tf.nn.l2_loss` and defined as

$$L = \frac{1}{2}\sum_i (\nu_i - \sigma(w_{ij}\mu_j))^2 \tag{.9}$$

the error is minimized through gradient decent (`tf.train.GradientDescentOptimizer`). Now, updates of the weight matrix w_{ij} can be calculated using backpropagation. This is in this case tractable to calculate, as the network consists of only one layer. Before the calculation, eq. (.8) is slightly modified: it is desired to not only update the network weights between neurons but also the weights from a source to a neuron in the spiking neural network. This is done by introducing a new variable $s_j = z_j \cdot \mu_j$ with the elementwise multiplication $\cdot$ (thus no summing over indices) and replacing $\mu_j \to s_j$ in each equation. This method scales the output by its source weights, which is in principle also a measure for the strength of the source input to the neuron. Now, weight updates can be calculated for w_{ij} and z_k. In gradient descent, the weight update is defined as the negative derivation of the loss function after the weight, multiplied by a small constant value α called the learning rate. The idea is to minimize the loss by descending along the steepest direction of the loss function. To converge to the minimum, the learning rate ensures that the steps are not too

large, which would lead to pass over the actual minimum. In formula:

$$\Delta w_{ji} = -\alpha \frac{\partial l}{\partial w_{ji}} \tag{.10}$$

$$\Delta z_i = -\alpha \frac{\partial l}{\partial z_i} \tag{.11}$$

So all that has to be done is to calculate the partial derivatives. This can be one by simply applying the chain rule twice:

$$\frac{\partial L}{\partial w_{ji}} = \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}} (\nu_k - \sigma(w_{kl} s_l))^2 \tag{.12}$$

$$= -\sum_k (\nu_k - \sigma(w_{kl} s_l)) \sigma'(w_{kl} s_l) \frac{\partial (w_{kl} s_l)}{\partial w_{ji}} \tag{.13}$$

$$= -(\nu_i - \sigma(w_{il} s_l)) \sigma'(w_{il} s_l) s_j \tag{.14}$$

$$= -(\nu_i - g_j) \sigma'(w_{il} s_l) s_j \tag{.15}$$

where σ' is the derivative of σ and in the last step we introduced the new variable $g_j = \sigma(w_{jl} s_l))$. Why this is important will be explained shortly. But first, a similar calculation for the source weight updates gives

$$\frac{\partial L}{\partial z_i} = \sum_k (v_k - g_k) \sigma'(w_{kl} s_l)) w_{ki} \cdot \mu_i \tag{.16}$$

Observe that the sum does not vanish in this case because the last derivative does not give a δ_{kj}. The derivative of the activation function is

$$\sigma'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ \exp x, & \text{otherwise} \end{cases}.$$

This concludes all that is needed to calculate weight updates. Now all that is left is to determine the variables in our spiking neural network. One choice can be to use the input rates for μ and the output rates for ν. However, heuristically it seems that a more natural description of the system is to define the output rates as g and the input rates as a one-hot encoded ν. This approximately says that it is desired that the neuron with the highest input rate spikes, while all others do not. As for μ, the sum of output and input rates is taken, which is justified by the fact that the spiking neurons receive this summed input as spike input.

C. Training Results for all Units

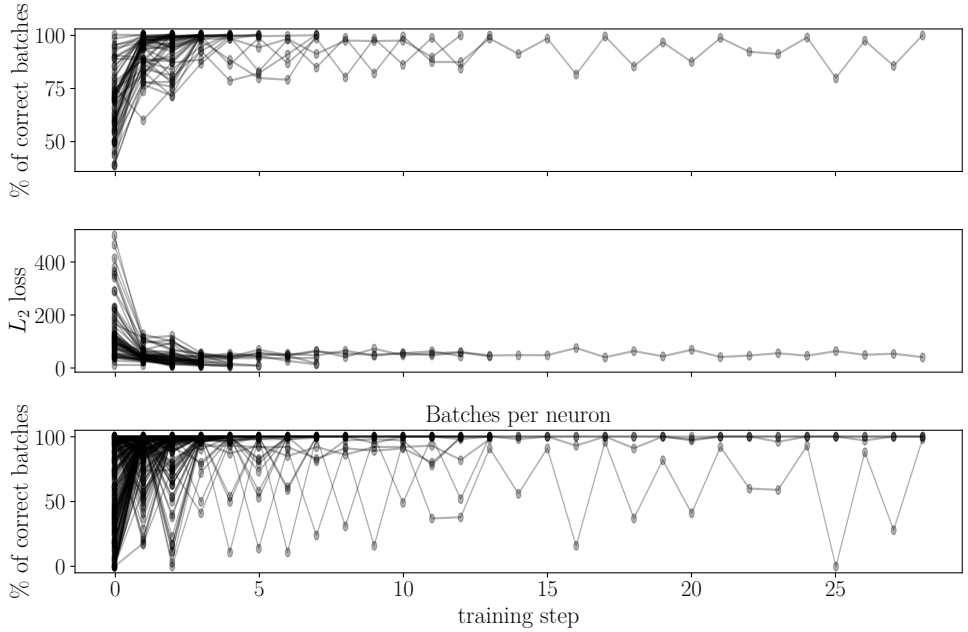

Figure .1.: **Correctness of all units on hardware.** The correctness per neuron before training ranges from 0 % to 100 %. After 28 training steps, all WTA units reach a correctness of over 99.7 %. Training is aborted when the correctness surpasses this percentage. The loss and correctness are anti-correlated.

The training of all units happens in serial using the method described above. The maximum training steps are set to 30. If a unit scores over 99.7 % correctness, the training is aborted and the next unit is trained. This decreases the overall training duration. The results for all units are shown in fig. .1. After 28 training steps, all units reach a correctness over 99.6 %. The correctness per neuron can be improved by about the same percentage. As expected, the L_2 loss can be reduced for all units. Most units abort training after at most 7 steps. Two units take slightly longer and abort after 12 steps. A last unit finishes after 28 steps. From the bottom panel it can be seen that it is one neuron in particular which does not score a high correctness for a long time.
In conclusion, all WTA units can be trained to achieve a correctness of at least 99.7 %.

Bibliography

Abadi, M., et al., Tensorflow: A system for large-scale machine learning, *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

Binas, J., G. Indiveri, and M. Pfeiffer, Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers, *CoRR*, *abs/1511.00540*, 2015.

Brüderle, D., Neuroscientific modeling with a mixed-signal vlsi hardware system, Dissertation, Ruprecht-Karls-Universität Heidelberg, doi:10.11588/heidok.00009656, 2009.

Cessac, B., and T. Vieville, On Dynamics of Integrate-and-Fire Neural Networks with Conductance Based Synapses, *ArXiv e-prints*, 2007.

Davison, A., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, Pynn: a common interface for neuronal network simulators, *Frontiers in Neuroinformatics*, *2*, 11, doi:10.3389/neuro.11.011.2008, 2009.

Davison, A. P., Hbp neuromorphic computing platform, 2018, `https://collab.humanbrainproject.eu/#/collab/19/nav/35258`, retrieved 2018-01-25.

Davison, A. P., E. Müller, S. Schmitt, B. Vogginger, D. Lester, and T. Pfeil, Hbp neuromorphic computing platform guidebook, 2017, `https://electronicvisions.github.io/hbp-sp9-guidebook/`, retrieved 2017-12-10.

Dechter, R., and J. Pearl, Network-based heuristics for constraint-satisfaction problems, *Artificial intelligence*, *34*(1), 1987.

Du, K.-L., Clustering: A neural network approach, *Neural Networks*, *23*(1), 89 – 107, doi:10.1016/j.neunet.2009.08.007, 2010.

Ercsey-Ravasz, M., and Z. Toroczkai, The chaos within sudoku, *Scientific Reports*, doi:10.1038/srep00725, 2012.

ErrorFAQ, Error faq of electronic vision(s), 2018, `https://brainscales-r.kip.uni-heidelberg.de/projects/symap2ic/wiki/ErrorFAQ`, retrieved 2018-02-06.

FET, Future and emerging technologies (fet), 2018, `http://ec.europa.eu/programmes/horizon2020/en/h2020-section/fet-flagships`, retrieved 2018-01-20.

Fonseca Guerra, G. A., and S. B. Furber, Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems, *Frontiers in Neuroscience*, *11*, 714, doi:10.3389/fnins.2017.00714, 2017.

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

Goodman, D., and R. Brette, Brian: a simulator for spiking neural networks in python, *Frontiers in Neuroinformatics*, *2*, 5, doi:10.3389/neuro.11.005.2008, 2008.

Habenschuss, S., Z. Jonke, and W. Maass, Stochastic computations in cortical microcircuit models, *PLoS Comput Biol 9(11): e1003311.*, doi:10.1371/journal.pcbi.1003311, 2013.

Hagan, M. T., and M. B. Menhaj, Training feedforward networks with the marquardt algorithm, *IEEE Transactions on Neural Networks*, *5*(6), 1994.

Hornik, K., M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, *2*(5), 359 – 366, doi:10.1016/0893-6080(89)90020-8, 1989.

Ilmberger, J., Development of a digitizer for the brainscales neuromorphic hardware system, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2017.

Intel, Intel movidius, 2018, `https://developer.movidius.com/`, retrieved 2018-01-20.

Jeltsch, S., A scalable workflow for a configurable neuromorphic platform, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2014.

Jouppi, N. P., et al., In-datacenter performance analysis of a tensor processing unit, *CoRR*, *abs/1704.04760*, 2017.

Kiene, G., Evaluating the synaptic input of a neuromorphic circuit, Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

Klähn, J., Training functional networks on large-scale neuromorphic hardware, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2017.

Kleider, M., Neuron circuit characterization in a neuromorphic system, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2017.

Koke, C., Device variability in synapses of neuromorphic circuits, Dissertation, Ruprecht-Karls-Universität Heidelberg, doi:10.11588/heidok.00022742, 2017.

Kunkel, S., et al., Nest 2.12.0, doi:10.5281/zenodo.259534, 2017.

Martins, et al., Selective floating gate non-volatile paper memory transistor, *physica status solidi (RRL) – Rapid Research Letters*, *3*(9), 308–310, doi:10.1002/pssr.200903268, 2009.

Martins, R., L. Pereira, P. Barquinha, N. Correia, G. Gonçalves, I. Ferreira, C. Dias, and E. Fortunato, Floating gate memory paper transistor, *Proc.SPIE*, *7603*, 7603 – 7603 – 11, doi:10.1117/12.841036, 2010.

Mauch, C., Commissioning of a neuromorphic computing platform, Masterarbeit, Universität Heidelberg, 2016.

Millner, S., Development of a multi-compartment neuron model emulation, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2012.

Müller, E., Novel operation modes of accelerated neuromorphic hardware, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2014.

Nass, R., and S. Przedborski, *Parkinson's Disease: Molecular and Therapeutic Insights From Model Systems*, Elsevier Science, 2011.

Pelánek, R., Difficulty rating of sudoku puzzles: An overview and evaluation, *CoRR*, *abs/1403.7373*, 2014.

R, N., M. N, C. C, and G. W., Firing patterns in the adaptive exponential integrate-and-fire model, *Biological Cybernetics*, doi:10.1007/s00422-008-0264-7, 2008.

Schemmel, J., J. Fieres, and K. Meier, Wafer-scale integration of analog neural networks, *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 431–438, doi:10.1109/IJCNN.2008.4633828, 2008.

Schemmel, J., D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, A wafer-scale neuromorphic hardware system for large-scale neural modeling, *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, doi:10.1109/ISCAS.2010.5536970, 2010.

Schmidt, D., Automated characterization of a wafer-scale neuromorphic hardware system, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

Schmitt, S., et al., Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system, *CoRR*, *abs/1703.01909*, 2017.

Schwartz, M.-O., Reproducing biologically realistic regimes on a highly-accelerated neuromorphic hardware system, Dissertation, Ruprecht-Karls-Universität Heidelberg, 2013.

Serrano, G., and P. Hasler, A floating-gate dac array, *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, *1*, I–357–I–360 Vol.1, doi:10.1109/ISCAS.2004.1328205, 2004.

Shannon, C. E., A mathematical theory of communication, *The Bell System Technical Journal*, *27*(3), 379–423, doi:10.1002/j.1538-7305.1948.tb01338.x, 1948.

Acknowledgments

First and foremost, I would like to thank Prof. Karlheinz Meier and Johannes Schemmel for the opportunity to carry out a master thesis in the magnificent field of neuromorphic hardware.
Prof. Dr. Michael Hausmann, thank you for examining this thesis as a second auditor.
A **huge** thank you to Sebastian Schmitt who supported me with his endless knowledge and experience in many ways, provided ideas for experiments and proofread many parts of my code as well as this thesis. A substantial part of this thesis was built on your visions.
David, thank you for your unlimited enthusiasm for programming and science, and for proofreading this thesis.
Thanks to the other proofreaders, Maurice, Eric, and Christian M. that made sure that at least most mistakes could be eradicated.
Thanks Johann and Eric for vastly broadening my knowledge about software development in general.
Thank you to Christoph and Andreas for the fun time in Paris.
Thanks to all the container people for support and all the fun we had at, between and after work.

Last and certainly not least, thanks to my friends and family. Mama and papa for always supporting me in all my endeavors. Johannes for being an awesome brother. Finally, Anna for being by my side, supporting me and always cheering me up.

Statement of Originality (Erklärung):

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

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

Heidelberg, February 9, 2018

.....
(signature)