Department of Physics and Astronomy
University of Heidelberg

Master Thesis
in Physics
submitted by
Timo Wunderlich
born in Freising
2019

Demonstrating Advantages of Neuromorphic Computation

This Master thesis has been carried out by Timo Wunderlich
at the
Kirchhoff Institute for Physics in Heidelberg, Germany
under the supervision of
Dr. Johannes Schemmel
and
Prof. Dr. Karlheinz Meier

Abstract

Demonstrating Advantages of Neuromorphic Computation: The goal of neuromorphic computation is to develop a substrate that mimics the brain and reproduces its capabilities in terms of efficient, adaptive and highly parallelized computation. In this pilot study we demonstrate and quantify these advantages regarding emulation speed, energy efficiency and robustness to noise. We use a prototype chip of the BrainScaleS 2 neuromorphic system to implement a reinforcement learning experiment that uses a biologically plausible learning rule, reward-modulated spike-timing-dependent plasticity, to learn to play a simplified version of the Pong arcade video game by smooth pursuit. The neuromorphic chip contains physical model neurons and synapses as well as a digital on-chip plasticity processing unit, which we also use to simulate the virtual environment. Compared to biological real-time, the chip operates with a 1000-fold acceleration, while consuming $57\,\mathrm{mW}$ of power. Compared to an equivalent software simulation on a commercial processor, our neuromorphic emulation is at least an order of magnitude faster and three orders of magnitude more energy-efficient. We find that fixed-pattern noise, which is inevitable in analog substrates, can be mitigated using learning, decreasing the need for calibration: the learning rule implicitly adjusts synaptic weights so as to compensate neuron parameter variations. At the same time, trial-to-trial variability due to temporal noise is not a nuisance but rather used as a computational resource for action exploration.

Zusammenfassung

Demonstration der Vorteile des neuromorphen Rechnens: Das Ziel des neuromorphen Rechnen ist es, ein Substrat zu entwickeln, welches das Gehirn nachempfindet und seine Fähigkeiten in Bezug auf effiziente, adaptive und hoch parallelisierte Informationsverarbeitung reproduziert. In dieser Pilotstudie demonstrieren und quantifizieren wir die wesentlichen Vorteile des neuromorphen Rechnens bezüglich Emulationsgeschwindigkeit, Energieeffizienz und Robustheit. Wir verwenden einen Prototypen des neuromorphen Systems BrainScaleS 2 um ein Experiment zu implementieren, welches verstärkendes Lernen mittels einer biologisch plausiblen Lernregel, Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP), verwendet um eine vereinfachte Version des Videospiels Pong durch Verfolgung des Balls zu lernen. Der neuromorphe Chip enthält physische Modellneuronen und -synapsen, sowie einen Plastizitätsprozessor, welcher in dieser Arbeit auch dazu dient, die virtuelle Umgebung zu simulieren. Im Vergleich zur biologischen Echtzeit operiert der Chip mit einer 1000-fachen Beschleunigung und kommt mit einem Leistungsbudget von $57\,\mathrm{mW}$ aus. Im Vergleich zu einer equivalenten Softwaresimulation auf einem kommerziellen Prozessor ist die neuromorphe Emulation mindestens eine Grössenordnung schneller und drei Grössenordnungen energieeffizienter. Wir stellen fest, dass festes Musterrauschen, welches auf analogen Substraten unvermeidbar ist, durch Lernen ausgeglichen werden kann: die Lernregel verändert die synaptischen Gewichte implizit derart, dass neuronale Parametervariationen ausgeglichen werden. Gleichzeitig wird Versuch-zu-Versuchsvariabilitaet, die aufgrund von Rauscheffekten besteht, als eine Ressource für das Lernen verwendet, statt eine Störung darzustellen.

Note

Parts of the thesis at hand have been submitted to Frontiers in Neuromorphic Engineering and are currently under peer-review. A pre-print is available (Wunderlich et al., 2018).

Acknowledgement

This work is dedicated to my dear family, friends and colleagues who have made the past year an enjoyable and fruitful one.
It is especially dedicated to the late Prof. Karlheinz Meier who left an indelible mark in our hearts and on this work.
I wish to express my gratitude to Ákos F. Kungl, Dr. Eric Müller and Dr. Mihai Petrovici for their kind supervision.
Special thanks to Dr. Johannes Schemmel for his unwavering support and to Prof. Dr. Hausmann for reviewing this thesis.

Contents

1 Introduction

> *The brain is just the weight of God,*
> *For, lift them, pound for pound,*
> *And they will differ, if they do,*
> *As syllable from sound.*
>
> — **Emily Dickinson**

Curiosity drives humans to study and conceptualize the physical world. As a result, scientific theories now explain physical phenomena on diverse temporal and spatial scales: from the interaction of elementary particles on the shortest time scales possible, to the formation of galaxy clusters over millions of years. However, the physical system that predicates scientific inquiry and indeed, any inquiry at all, remains largely elusive: the brain. Modern neuroscience has embarked on a journey to change this; researchers acquire data and engage in modeling at a rapid pace.

A thorough understanding of the brain would enable the reproduction of its computational abilities and high-level functions, as is the declared goal of many researchers. As of today, this objective remains unfulfilled. Traditional computers in the vein of Turing and von Neumann, despite sometimes being abusively termed "electronic brains" in their early days, operate in a fundamentally different way. The divide becomes apparent when considering that computers are very bad at tasks that are trivial and effortless to humans, such as natural language processing, while humans must exert considerable effort to solve simple arithmetic problems. The human brain is highly adaptive and robust to damages of its neural substrate (Feuillet et al., 2007), while the critical parts of computers generally rely on the proper functioning of every single one of its transistors. Besides this, the brain is strongly constrained in terms of power consumption and space.

Although the development of traditional computers proceeds at an exponential pace, as expressed by Moore's law, there exists a wide consensus that the future of computation will require the development of novel principles (Dean et al., 2018). It is a natural approach to study the basis of computation found in the brain, thereby laying the groundwork for such a paradigm shift. This requires the development of models of neural computation, which need to be implemented on some substrate. Commonly, the substrate is a digital computer that simulates models of neural networks; this approach, however, does not scale very well: simulations of large-scale neural networks on state-of-the-art supercomputers or highly specialized digital hardware achieve real-time simulation speed at best (Jordan et al., 2018; Albada et al., 2018; Mikaitis et al., 2018). This makes simulations of biological learning processes that happen on long time scales largely unfeasible.

Alternatively, neural networks can be implemented using the idea of *physical modeling*: neurons and synapses have physical representations that behave according to temporal dynamics which mimic their biological counterpart. The corresponding field of *neuromorphic computing* was pioneered by Carver Mead in the 1980s (Mead, 1989). Models of neurons and synapses can be implemented in electronic integrated circuits – allowing for the manufacture of neuromorphic chips using well-established CMOS technology (Veendrick, 2017). These models ideally retain enough biological detail to be able to reproduce the functionality of biological neural networks, while disregarding unnecessary detail. At the same time, the model dynamics can be accelerated by orders of magnitude, allowing for the evaluation of long-term learning processes.

In this work, we demonstrate and quantify the key advantages of the employed neuromorphic approach in terms of power consumption, accelerated learning and robustness to noise. To this end, we implement a reinforcement learning experiment on a prototype neuromorphic chip with analog network emulation extended by a flexible digital on-chip plasticity processor.

2 Neuromorphic Computing

What I cannot create,
I do not understand.

— **Richard Feynman**

In neuromorphic computing, researchers aim to develop computational substrates that mimic the brain. The classical approach, developed by Carver Mead in the 1980s, is to physically implement models of neural networks in analog circuits using Very Large Scale Integration (VLSI) which allows to create integrated circuits with millions of transistors on a single chip (Mead, 1989). Specialized solutions born of this approach, such as touch sensors (*Synaptics Homepage* 2019) or hearing aids (*Sonic Homepage* 2019) based on analog neuro-inspired circuits, have achieved substantial commercial success. Other solutions aim for a general computational substrate based on analog electronic circuits, in order to provide researchers with the possibility to flexibly implement models of spiking neural networks, such as the Neurogrid project (Benjamin et al., 2014), a series of real-time neuromorphic chips from the INI Zurich (Indiveri et al., 2011) and the accelerated large-scale BrainScaleS hardware (Schemmel, Brüderle, et al., 2010). Besides this physical modeling approach, there exists specialized digital hardware for the simulation of neural networks, such as IBM's TrueNorth (Merolla et al., 2014), SpiNNaker (Furber et al., 2014) and Intel's Loihi (Davies et al., 2018).

In the following, we review models of spiking neural networks and synaptic plasticity and discuss how neuron models can be emulated in electronic circuits. Then, the prototype chip that we use in this work, HICANN-DLSv2, is described.

2.1 Abstract & Physical Models

Neurons are thought to be the functional unit of the brain (the neuron doctrine), with synapses mediating the connections between neurons. They can be both mathematically modeled and implemented as physical models (Wulfram Gerstner, Kistler, et al., 2014). Both approaches will be discussed in the following.

2.1.1 Neurons & Synapses

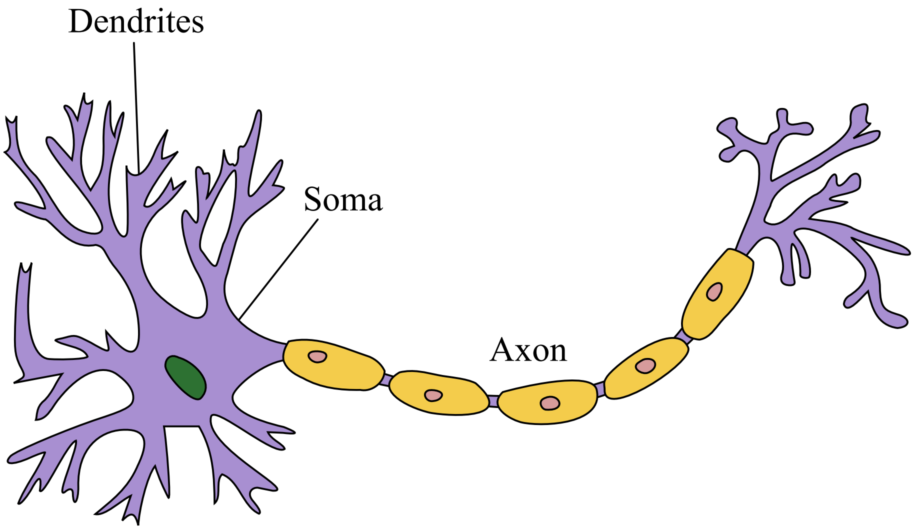


Fig. 2.1: A biological neuron, adapted from Quasar, 2009.

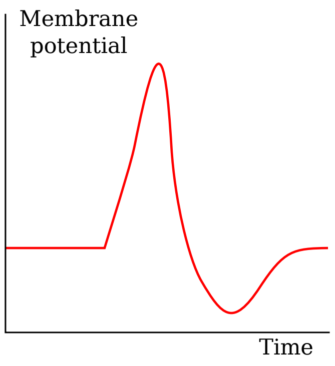


Fig. 2.2: The qualitative shape of an action potential. Taken from Chris73, 2015.

In Biology The description given in the following largely follows Wulfram Gerstner, Kistler, et al., 2014. A biological neuron is a specialized cell that is electrically excitable. It can be divided into three parts: its soma (cell body), its dendrites and its axon (see Figure 2.1). The neuron's dendrites branch out from the soma and form synapses with other neurons. The dendritic tree allows the soma to pick up signals from a large number of synaptic partners (generally in the order of 10^4). These signals can either electrically excite or inhibit the neuron, which is expressed in an increase or decrease of the *membrane potential* (the electrical potential with respect to the outside of the cell). The neuron has voltage-gated and ligand-gated ion channels as well as ion pumps which serve to transport charged particles in and out of the cell. While voltage-gated and ligand-gated ion channels are activated by membrane potential fluctuations and neurotransmitters, respectively, ion pumps use energy in the form of ATP to actively pump charged particles in and out of the cell. The *resting potential* is the stable membrane potential of the neuron in the absence of input from other neurons and is determined by the interaction of these different mechanisms. It is generally in the order of $-60\,\mathrm{mV}$.

If the net excitation and therefore the membrane potential is large enough, the ion channel dynamics will cause a rapid depolarization of the cell, causing the cell body to elicit a voltage pulse called an action potential that travels along the axon where synaptic contacts with other neurons' dendrites are activated. After an

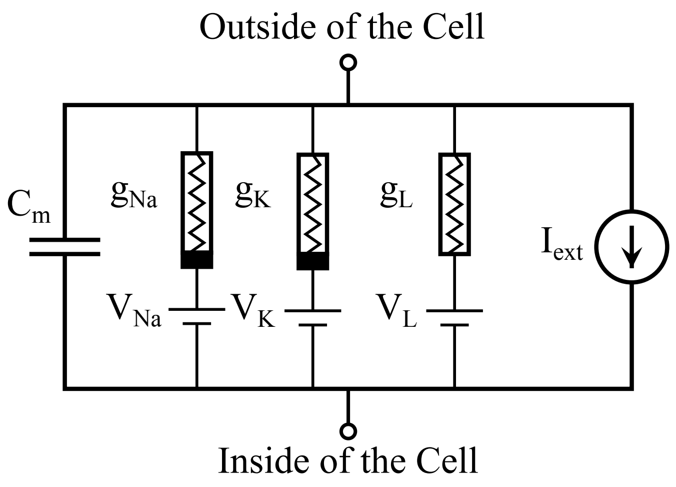


Fig. 2.3: Circuit diagram of the Hodgkin-Huxley model, adapted from Krishnavedala, 2012.

action potential, the neuron is in a *refractory state* where the membrane potential is hyperpolarized (i.e., below the resting potential) and the neuron is harder to excite. This effect lasts for a time in the order of $1\,\mathrm{ms}$. Refer to Figure 2.2 for the stereotypical shape of an action potential.

The means by which neurons communicate are precisely these action potentials which cause the release of different types of neurotransmitters at the synaptic sites, leading to the opening of ligand-gated ion channels. By this mechanism, synapses can have an excitatory or inhibitory effect on the post-synaptic neuron: the Post-Synaptic membrane Potential (PSP) of the neuron is increased in the former and decreased in the latter case. The strength or weight of the synapse is characterized by the amplitude of the elicited PSP.

The shape of the action potentials is stereotypical and contains no information. Rather, information is conveyed by the timing and frequency of the action potentials. The action potentials are also called *spikes*.

Hodgkin-Huxley Model The classic *Hodgkin-Huxley-Model* (Hodgkin and Huxley, 1952; Wulfram Gerstner, Kistler, et al., 2014) is a mathematical model of action potential generation by the dynamics of the ion channels in a neuron. It is a point-neuron model in the sense that the spatial extent of neurons is not taken into account. The idea is to view a neuron as an electrical circuit, with a capacitor C_m representing the membrane and with electrical conductances modeling voltage-dependent ion channels (see Figure 2.3). The model consists of four coupled Ordinary Differential Equations (ODEs): one for the membrane potential V and three to model the interaction of three different ion channels with corresponding reversal potentials (Sodium g_{Na} and V_{Na}, Potassium g_{K} and V_{K}, Leak g_{L} and V_{L}):

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -g_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - g_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - g_{\mathrm{L}} (V - V_{\mathrm{L}}) + I_{\mathrm{ext}}(t)\,, \tag{2.1}$$

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(V)(1-n) - \beta_n(V) n\,, \tag{2.2}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(V)(1-m) - \beta_m(V) m\,, \tag{2.3}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(V)(1-h) - \beta_h(V) h\,, \tag{2.4}$$

where $\alpha_*(V)$ and $\beta_*(V)$ are voltage-dependent opening and closing rates and $I_{\mathrm{ext}}(t)$ is a time-dependent external current. The different opening and closing rates enable the model to reproduce dynamic action potential generation: sufficient external excitatory input causes a positive feedback effect that causes a rapid up-swing of the membrane potential (caused by rapid opening and closing of sodium channels) followed by a down-swing (caused by slow potassium channel inactivation) and a slow recovery from the hyper-polarization. Figure 2.4 depicts a single action potential generated in the Hodgkin-Huxley model that is the result of a short, but strong stimulation based on an injected current.

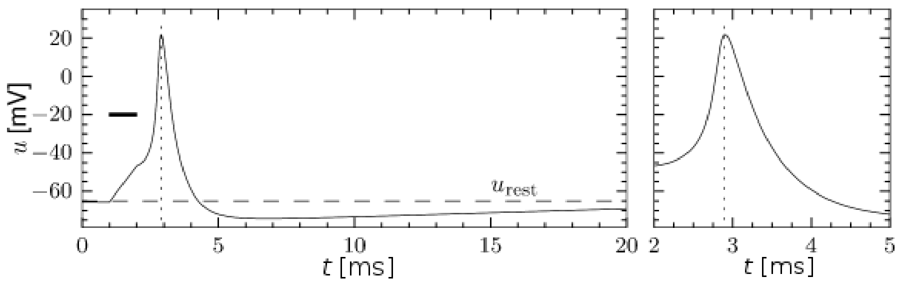


Fig. 2.4: Action potential in the Hodgkin-Huxley model after stimulation between $t = 1$ and $t = 2$ with an expanded view in the right panel. From (Wulfram Gerstner, Kistler, et al., 2014).

Leaky Integrate-and-Fire Model The computational cost of simulating neurons can be reduced by moving to the simplified Leaky Integrate-and-Fire (LIF) model, which is a simple neuron model that preserves biologically meaningful properties (Wulfram Gerstner, Kistler, et al., 2014). Because action potentials only convey information by their existence or non-existence, they are reduced to events with no temporal extent. This model is defined by a single differential equation that represents a leaky membrane, while synaptic input can be modeled either by time-varying conductances $g_i^{\text{syn}}(t)$ (conductance-based, COBA) or external currents $I_i^{\text{syn}}(t)$ (current-based, CUBA):

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -g_{\mathrm{L}}(V - V_{\mathrm{L}}) + \sum_{i=1}^{K} g_i^{\text{syn}}(t)(V - V_i) \qquad \text{COBA}, \tag{2.5}$$

$$C_m \frac{\mathrm{d}V}{\mathrm{d}t} = -g_{\mathrm{L}}(V - V_{\mathrm{L}}) + \sum_{i=1}^{K} I_i^{\text{syn}}(t) \qquad \text{CUBA}, \tag{2.6}$$

where the sum goes over the K synapses with reversal potentials V_i in case of COBA. Spiking is not part of the ODE dynamics, as in case of the Hodgkin-Huxley model, but is instead introduced by the condition

$$\text{Neuron spikes at time } t \quad \Leftrightarrow \quad V(t) > V_{\text{thresh}} \tag{2.7}$$

with threshold voltage V_{thresh}. As soon as the threshold voltage is crossed, the membrane potential is reset to the reset potential V_{reset} and held fixed for a time period τ_{ref} that models the temporal extent of the refractory state:

$$V(t) = V_{\text{reset}} \quad \text{if} \quad t \in [t_f, t_f + \tau_{\text{ref}}) \,, \tag{2.8}$$

where t_f is the time of the last spike before t. After the refractory period, the membrane potential dynamics continue according to Equation (2.5) or Equation (2.6) with $V(t_f + \tau_{\text{ref}}) = V_{\text{reset}}$ as the initial condition.

The synaptic input is a sum over the pre-synaptic spike times t_f within which the dimensionless synaptic interaction kernel $\epsilon_i(t)$ is weighted with the dimensioned synaptic weight w_i:

$$g_i^{\text{syn}}(t) = \sum_{t_f} w_i \epsilon_i(t - t_f) \qquad \text{COBA}, \tag{2.9}$$

$$I_i^{\text{syn}}(t) = \sum_{t_f} w_i \epsilon_i(t - t_f) \qquad \text{CUBA}. \tag{2.10}$$

The weights have dimensions of an electrical conductance in the case of COBA and an electrical current in the case of CUBA. A common choice for the exponential kernels $\epsilon_i(t)$ is a double-exponential normalized to one (Mihai Alexandru Petrovici, 2016):

$$\epsilon_i(t) = \theta(t) \frac{1}{\tau_{\text{rise}}^{\text{syn}} - \tau_{\text{fall}}^{\text{syn}}} \left[\exp\left(-\frac{t}{\tau_{\text{rise}}^{\text{syn}}} \right) - \exp\left(-\frac{t}{\tau_{\text{fall}}^{\text{syn}}} \right) \right], \tag{2.11}$$

with rise and fall time constants $\tau_{\text{rise}}^{\text{syn}}$ and $\tau_{\text{fall}}^{\text{syn}}$. A simpler and popular choice is to neglect the rise time, $\tau_{\text{rise}}^{\text{syn}} \to 0$, so that the synaptic input kernel becomes an exponential function:

$$\epsilon_i(t) = \theta(t) \frac{1}{\tau_{\text{fall}}^{\text{syn}}} \exp\left(-\frac{t}{\tau_{\text{fall}}^{\text{syn}}}\right) . \tag{2.12}$$

The COBA and CUBA LIF model differ in their neuronal membrane dynamics. The reversal potentials that are part of modeling gated ion channels in the case of COBA limit the dynamic range of the membrane potential and lead to a non-linear summing of PSPs, whereas the CUBA model linearly sums synaptic kernels and therefore PSPs.

Physical Neuron Models The neuron models described above can be simulated on a digital computer by integrating the respective differential equations. An alternative approach is to implement a physical model of these equations, where the system dynamics naturally evolve based on the same or sufficiently similar differential equations. The Hodgkin-Huxley and LIF model have an equivalent representation based on electronic circuits. VLSI implementations have been demonstrated for both models (Yu and Cauwenberghs, 2010; Douence et al., 1999; Häfliger et al., 1997; Schemmel, Meier, et al., 2004; Indiveri et al., 2011; Benjamin et al., 2014; Aamir, Müller, et al., 2016; Aamir, Stradmann, et al., 2018). Due to its simpler nature the LIF model can be implemented in a smaller area, allowing for more neurons per chip. Some implementations, including the one that underlies the chip used in this thesis, provide neuronal time constants that are orders of magnitude faster than the biological counterpart (Schemmel, Meier, et al., 2004; Aamir, Müller, et al., 2016; Aamir, Stradmann, et al., 2018).

2.1.2 Synaptic Plasticity

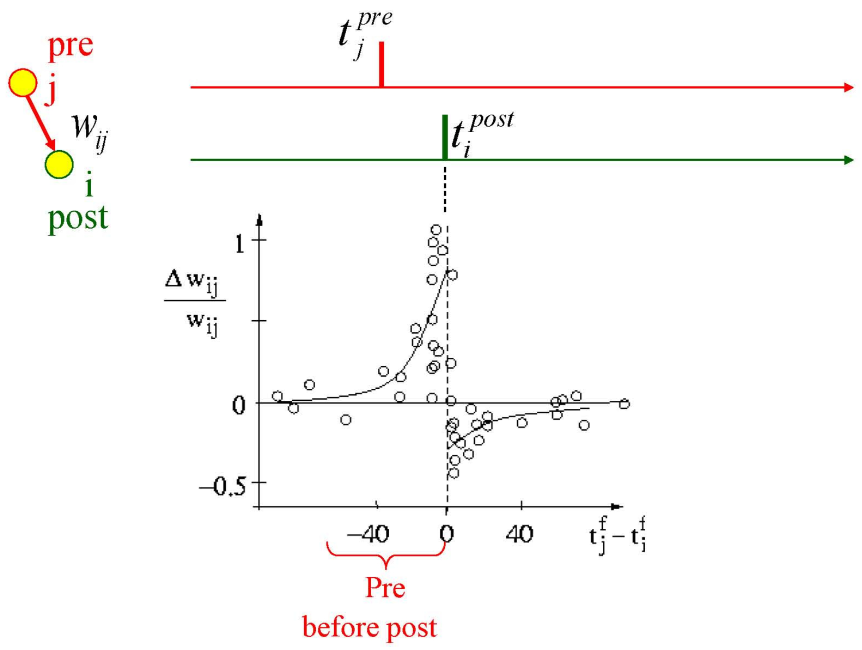


Fig. 2.5: Spike-Timing-Dependent Plasticity (STDP): pre-before-post coincidences cause LTP, while post-before-pre coincidences cause LTD. Figure from Sjöström and W Gerstner, 2010; experimental data from Bi and Poo, 1998.

The ability to learn is a crucial part of biological neural networks. In general terms, learning in neural networks refers to any adaptations that serve to improve network function with respect to some learning objective. These adaptations can be mediated by changes in the structure of the network and by changes in the properties of network components (i.e., neurons and synapses). Changes in synaptic efficacy and therefore in the coupling strength between neurons are a paramount part of such learning processes. Permanent changes to synaptic efficacy are referred to as Long-Term Potentiation (LTP) or Long-Term Depression (LTD), depending on whether the change is positive or negative. These long-term synaptic changes are thought to form the basis of memory formation and learning (Bliss and Collingridge, 1993).

The classic Hebbian theory of learning posits that "neurons wire together if they fire together" (Hebb, 1949; Lowel and Singer, 1992). In this view, synapses are strengthened (LTP) if pre-synaptic activity coincides with post-synaptic activity. Accordingly, learning rules can be defined in terms of the pre- and post-synaptic firing rates. The theory of Spike-Timing-Dependent Plasticity (STDP) takes this further by describing synaptic plasticity on the level of individual spikes. It was famously observed (Bi and Poo, 1998) that synaptic efficacy is increased (LTP) for pre-before-post occurrences (Hebbian learning) and decreased (LTD) for post-before-pre occurrences (Anti-Hebbian learning). This effect has been observed *in vivo* in rats, cats, humans and insects (Markram et al., 2011) and is commonly modeled using an anti-symmetric exponential dependence of the weight change on the spike time difference (see Figure 2.5). However, the empirical dependence is influenced by neurotransmitter concentration. For example, elevated levels of Dopamine can turn the post-before-pre part of STDP from LTD to LTP in hippocampal neurons, thereby changing the sign of the STDP window (Pawlak, Wickens, et al., 2010; Wulfram Gerstner, Lehmann, et al., 2018).

Plasticity mechanisms such as STDP can be part of neuromorphic models and are important to model and investigate biologically plausible learning processes. Each synapse of the prototype chip used in this thesis, HICANN-DLSv2, contains a correlation sensor that can be used for STDP. The experimental results given later in this thesis use these sensors for learning.

2.2 The HICANN-DLSv2 Neuromorphic Chip

HICANN-DLSv2 is the second prototype chip of the BrainScaleS 2 (BSS2) neuromorphic architecture (Friedmann et al., 2017). BSS2 is the successor of BrainScaleS 1 (BSS1), which is a wafer-scale neuromorphic system (Schemmel, Brüderle, et al., 2010). In both cases, the general approach is to implement physical models of neurons and synapses in electronic circuits using VLSI and CMOS technology. Compared to BSS1, BSS2 is manufactured using a smaller process ($65\,\mathrm{nm}$ vs. $180\,\mathrm{nm}$) and contains a number of additional features for on-chip plasticity and learning, such as embedded plasticity processors. HICANN-DLSv2 contains 32 physical-model analog neurons, 1024 dedicated synapse circuits and one plasticity-processing unit. The chips within the envisioned large-scale BSS2 system will contain more neurons and synapses and several plasticity processors connected with wafer-scale integration.

All experiments in this thesis were conducted on HICANN-DLSv2.

2.2.1 Neurons & Synapses

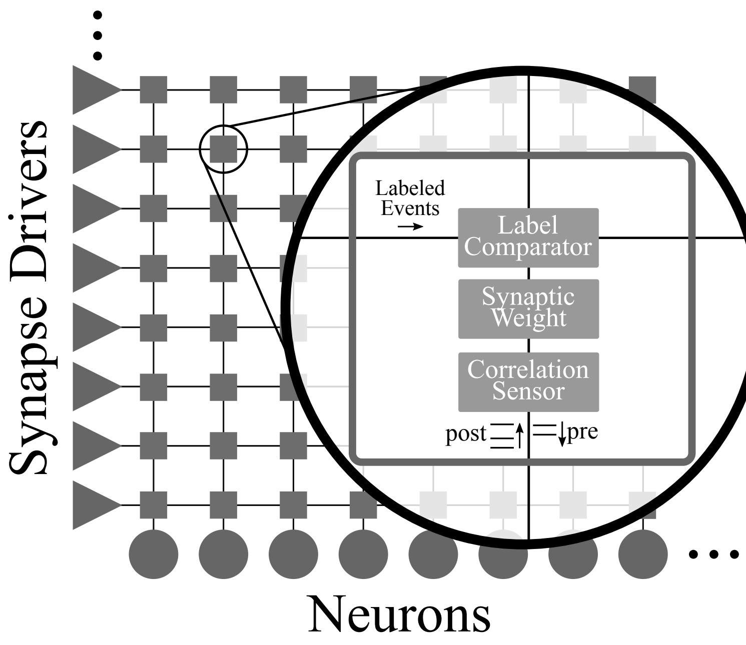


Fig. 2.6: Neural array on HICANN-DLSv2. Each neuron has a column of 32 synapses. Synapse drivers inject labeled spike events into synapse rows, where each synapse compares its label with the spike's label and generates a pre-synaptic spike if they match. Additionally, each synapse contains causal/anti-causal correlation sensors (see inset zoom).

Neurons The 32 neurons on HICANN-DLSv2 are a physical implementation of the CUBA LIF neuron model (Aamir, Müller, et al., 2016; Aamir, Stradmann, et al., 2018). Each neuron contains an 8-bit spike counter that can be read out and reset using the plasticity processing unit. Because neurons use the fast supra-threshold dynamics of CMOS transistors, the neuronal dynamics are faster by three orders of magnitude when compared to biological real time. Neuronal time constants in biology are typically in the order of milliseconds, whereas they are in the order of microseconds on HICANN-DLSv2. In this work, we always provide the real wall-clock time constants and do not convert to the biological domain if not specified otherwise.

Analog Parameter Storage The analog neurons described above require stable current and voltage biases that can be parameterized to achieve different LIF model constants (time constants and voltages). These analog parameters are provided by a single capacitive memory array (abbreviated CapMem) that contains 16 current and 8 voltage cells for each neuron individually and 16 current and 8 voltage cells that affect all neurons globally (Stradmann, 2016). A detailed account of the CapMem design can be found in Hock, 2014. Each cell within the CapMem can be programmed using a 10-bit digital value. Voltage cells convert this digital value into a voltage across a capacitor by an elaborate programming scheme, while current cells convert the digital parameter into a current bias. Both cell types are designed to have a linear input-output relationship, which was verified empirically (Stradmann, 2016). Fixed-pattern noise causes the output of CapMem cells to vary across cells for a given digital parameter (see the *Noise* paragraph for a description of the effects).

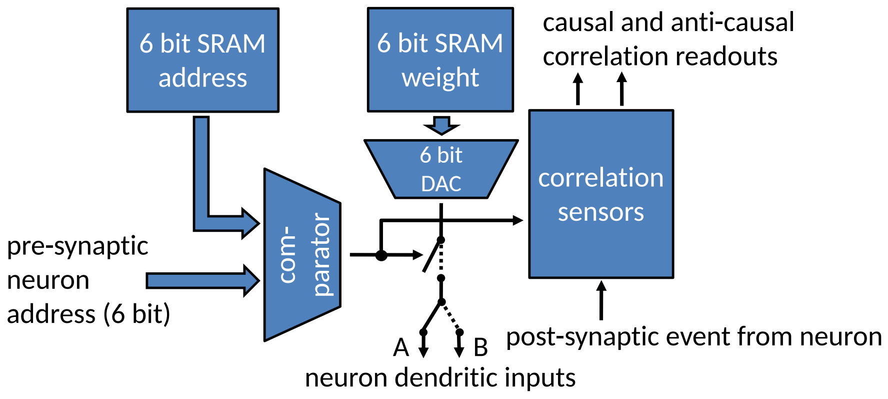

6 bit SRAM address
6 bit SRAM weight
causal and anti-causal correlation readouts
6 bit DAC
correlation sensors
pre-synaptic neuron address (6 bit)
com-parator
A
B
post-synaptic event from neuron
neuron dendritic inputs

Fig. 2.7: Synapse circuits on HICANN-DLSv2. Every synapse has a 6-bit weight, a 6-bit label (address) and causal/anti-causal correlation sensors. Adapted from Friedmann et al., 2017.

Synapses The 1024 synapses are arranged in a 32-by-32 array so that each neuron can receive input from a column of 32 synapses (see Figure 2.6). Spikes are injected into the array by row-wise synapse drivers that can be configured to have an excitatory or inhibitory effect. The injected spikes carry 6-bit labels. When a spike is injected into a row, every synapse of that row compares its label with the label of the spike and generates a current pulse (corresponding to a pre-synaptic spike) traveling “down” toward the neuron if they match (see Figure 2.7). The amplitude of this current pulse is proportional to the 6-bit weight stored at the synapse (the temporal length of the pulse is a global parameter and the same for all synapses). At the neuron, this current pulse elicits an exponentially shaped post-synaptic current of proportional amplitude on the neuronal membrane. Post-synaptic spikes are signaled (back-propagated) to all synapses of the respective column. Synapses use this signal in their correlation sensor (see Section 2.2.2).

Noise Due to their analog nature, neurons and synapses on HICANN-DLSv2 are subject to both *fixed-pattern noise* and *temporal variability*. Fixed-pattern noise refers to variability between chip components that is constant in time and caused by process variations that lead to deviations from targeted parameters. For example, two neurons on a given chip might exhibit different time constants for an equal set of hardware parameters. This kind of variability can be reduced using calibration, where parameters are tuned individually so as to compensate for variations. A neuronal calibration to compensate fixed-pattern noise is available for HICANN-

DLSv2 and was used in this thesis (Stradmann, 2016, see Section 4.2.1 for effects). Temporal variability refers to effects that vary in time and have a continuous influence on dynamic variables such as neuronal membrane potentials. It is typically caused by cross-talk, thermal noise or unstable analog parameter storage. Temporal variability leads to the trial-to-trial variation of high-level observations, e.g. firing rates (see Section 4.2.1).

2.2.2 Correlation Sensors

Every synapse on HICANN-DLSv2 contains two analog correlation sensors that record causal or anti-causal correlations of pre- and post-synaptic firing which can serve to implement STDP using the plasticity processor (Wunderlich, 2016; Friedmann et al., 2017). While the causal correlation sensor is sensitive to pre-before-post occurrences, the anti-causal sensor records post-before-pre occurrences. Specifically, the correlation sensors measure and accumulate the exponentially weighted temporal distance between nearest-neighbor spike pairs, which allows the embedded plasticity processor to model the empirically observed STDP curve shown in Figure 2.5, among other forms of plasticity. The accumulated values are stored on two separate storage capacitors corresponding to the two correlation sensors. In an idealized model, the voltages on the storage capacitors are

$$a_+ = \sum_{\mathrm{pre-post}} \eta_+ \exp\left(-\frac{t_\mathrm{post} - t_\mathrm{pre}}{\tau_+}\right) \tag{2.13}$$

and

$$a_- = \sum_{\mathrm{post-pre}} \eta_- \exp\left(-\frac{t_\mathrm{pre} - t_\mathrm{post}}{\tau_-}\right) \quad , \tag{2.14}$$

for the causal and anti-causal sensor, respectively, with decay time constants τ_+ and τ_- and scaling factors η_+ and η_-. These parameters are global parameters, which are set as digital values by the user and converted to voltages to be used in the sensor circuits using Digital-to-Analog Converters (DACs). The analog voltages representing the accumulated correlation can be read out using 8-bit column-wise Analog-to-Digital Converters (ADCs), so one row can be read out in parallel. The digitized values can be used by the plasticity processor (see following section) to calculate weight updates.

2.2.3 Plasticity Processing Unit

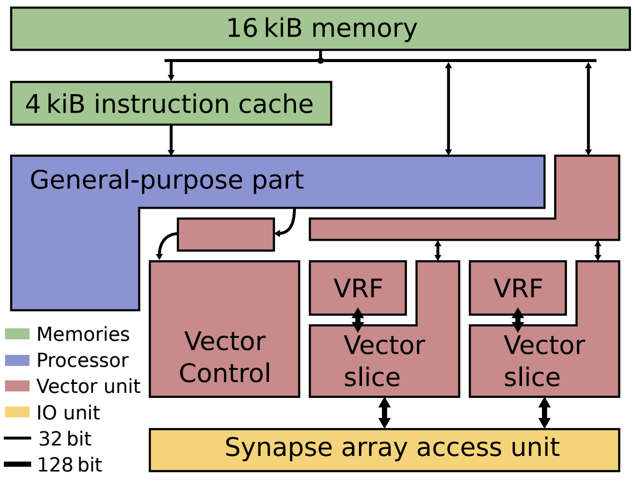


Fig. 2.8: Functional diagram of the PPU. The PPU has a general-purpose part and a customized vector unit loosely coupled to the general purpose part that accelerates plasticity computations using SIMD operations. Taken from Friedmann et al., 2017.

HICANN-DLSv2 contains a Plasticity Processing Unit (PPU), which is a general-purpose 32-bit processor implementing the PowerPC-ISA 2.06 instruction set with custom vector extensions (Friedmann et al., 2017, see Figure 2.8). This processor has access to parts of the chip's neural network: it can read and write synaptic weights and labels, read and reset correlation sensors, read and write neuron parameters, read neuronal rate counters and inject spikes into the synapse array. The access to the synaptic array is implemented using dedicated fully-parallel access ports, so that synaptic parameters can be processed in an efficient row-wise vectorized fashion using Single Instruction Multiple Data (SIMD) instructions.

The inclusion of the PPU into HICANN-DLSv2 is critical, as it allows to flexibly implement a broad range of plasticity rules using neural network observables. This is an important capability as (synaptic) plasticity is an active area of research and future findings can be easily accommodated using the PPU. In this way, HICANN-DLSv2 provides a flexible substrate for researching plasticity in spiking neural networks.

The PPU is clocked at $98\,\mathrm{MHz}$, has access to $16\,\mathrm{KiB}$ of main memory, has a $4\,\mathrm{KiB}$ instruction cache while vector registers have a width of 128 bit and can be processed in slices of eight 16-bit or sixteen 8-bit slices.

2.2.4 Software

HICANN-DLSv2 is configured using an FPGA which is in turn programmed using the host computer via a USB connection. The preliminary software stack that was used for rapid prototyping and debugging of HICANN-DLSv2 is called FRICKEL-DLS and is written in C++ with Python bindings. This is the software stack that was used for the main experiment in this thesis. It can be used for container-based configuration of the chip, including the loading of PPU programs.

PPU programs can be written in assembly and higher-level languages such as C or C++ and compiled using a customized GCC compiler (Electronic Vision(s), 2017; Stallman and GCC Developer Community, 2018). A software library called LIBNUX provides components necessary for compilation (e.g. CRT0 startup routines) as well as code to access functional units on the chip (e.g. synapses). Programs on the PPU can be debugged using the GNU debugger using custom remote debugging software extensions developed partially in the scope of this work (see Section 5.2).

3 Reinforcement Learning

Nature has placed mankind under the governance of two sovereign masters, pain and pleasure. It is for them alone to point out what we ought to do, as well as to determine what we shall do.

— **Jeremy Bentham**
Introduction to the Principles of Morals and Legislation

Feelings of pleasure and pain are fundamental to the being of humans and animals and decisively drive their behavior, as is widely recognized by philosophers and psychologists alike (Thorndike, 1911; Bentham, 1780). The theory of reinforcement learning (RL) formalizes how rewarding (or punishing) environmental feedback can lead a behaving agent to change its behavior, generally so as to increase the future expected reward. In their seminal textbook (Sutton and Barto, 1998), Richard Sutton and Andrew Barto write

> Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

RL is therefore a very general framework that can be used to model ongoing, goal-driven learning in real-world or simplified environments. In this chapter, we review the basics of RL in abstract terms, study a simple class of RL problems (multi-armed bandits), consider biological aspects of RL and describe efforts to model RL in spiking neural networks. The experiment on BrainScaleS 2 that is described in a later chapter implements RL in a spiking neural network and can be seen as solving several multi-armed bandit problems. In describing RL, we largely follow Sutton and Barto, 1998.

3.1 Abstract Reinforcement Learning

The fundamental component of a reinforcement learning problem is the *environment*, in which a learning agent inhabits certain *states* and in which it can execute *actions* that may lead to state transitions and can incur a *reward*. In contrast to other types of supervised learning, the learning entity receives evaluative, not instructive feedback and must find the appropriate actions to take on its own. In other words, it must develop a *policy* that defines how to choose (context-dependent) actions solely based on the rewarding or punishing, and potentially delayed, feedback from the environment.

If a model of the environment is available, the agent may use planning methods to develop a behavioral strategy. The model needs to specify the state transitions and rewards, both of which are generally stochastic and given in the form of a probability distribution. RL problems are commonly framed as Markov Decision Problems (MDPs), where the states compose a Markov chain, so that only the current state is relevant to the agent. The agent may use *dynamic programming* methods such as policy or value iteration to compute an optimal policy in a MDP. In these methods, the agent iteratively evaluates and improves its current policy using the MDP model and is guaranteed to converge to the optimal policy (Sutton and Barto, 1998). The policy evaluation is performed by estimating the state-value function which assigns an expected return (future reward) to a state given the current policy. Each iteration improves the agent's state-value estimate and then uses this estimate to improve its policy so as to perform actions that lead to higher-value states, given the state transition model.

Such dynamic programming methods require the full probability distributions specified by the model. If no model is available, the agent may use its own experience as samples in place of an explicit probability distribution, allowing the agent to empirically evaluate its current policy and improve it based on the sample-averaged experience. This is the basis of so-called *Monte-Carlo* methods which operate off-line in the sense that data acquisition and policy update occur in separate phases: the agent samples the MDP using its current policy until a terminal state is reached and then uses the acquired data to evaluate and update its policy. The policy evaluation is done by maintaining an estimate of the action-value function that assigns an expected return to a state-action pair given the current policy. Each iteration improves this estimate and allows the agent to pick higher-value actions without an explicit state transition model.

Temporal Difference (TD) methods are similar to Monte-Carlo methods in that they require no explicit model and use samples to evaluate the current policy, but differ in that they do not separate stepping through MDP states and evaluating the policy. Instead, they update the action-value function (and in most cases, the policy itself) while the MDP is still running (has not reached a terminal state) using the difference between empirical and expected reward. In practice, TD methods usually converge faster than Monte-Carlo methods (Sutton and Barto, 1998). The basic TD method formulation uses single state transitions (i.e., each update only considers the current state transition and the corresponding reward), but can be amended using *eligibility traces* which allow to credit previously visited states with an unexpected change in reward and can speed up convergence.

A general concern in model-free methods such as Monte-Carlo or TD methods is the *exploration-exploitation trade-off*. This trade-off arises because the agent needs to explore the action space to find high-return actions and states to which the agent is initially oblivious. At the same time, the agent should exploit its existing knowledge (given in form of the state- or action-value function) to behave in such a way as to acquire as much reward as possible. A simple way to deal with this trade-off are *ϵ-greedy* policies, which take the best action with probability $1 - \epsilon$ and a random action with probability ϵ, as discussed in the next section.

Another class of methods to solve RL problems is given by *policy gradient* methods, which do not require state- or action-value functions, but directly improve a given policy by updating the policy's parameters using gradient ascent on the expected return. This technique will be described in detail and applied to a simple class of RL problems, multi-armed bandits, in the following section.

3.2 Multi-armed Bandits

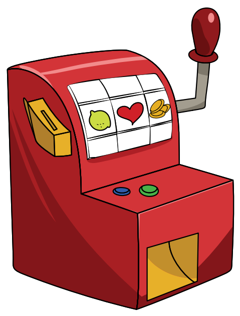


Fig. 3.1: A one armed bandit. Taken from Open Clipart, 2013.

Multi-armed bandits represent a class of simple RL problems where the agent inhabits a single state and can choose from multiple actions which yield different rewards (contrary to a one-armed bandit, where only one action can be chosen). The goal of the agent is generally to maximize its future payoff. Considering this simplified scenario allows to introduce basic concepts of RL such as the exploration-exploitation trade-off using a concrete class of problems.

Consider a k-armed bandit, with the set of actions being $\mathcal{A} = \{a_0, \dots, a_{k-1}\}$, the set of rewards being $\mathcal{R} \subset \mathbb{R}$ and the reward distribution given an action $a \in \mathcal{A}$ being $r(a) : \mathcal{R} \to [0, 1]$. For each action, the environment provides reward by sampling from the reward distribution corresponding to the chosen action. In the following, we will discuss two methods to learn agent behavior that aim to maximize payoff.

3.2.1 Action-Value Methods

In action-value methods, the idea is to have a function that provides the expected return when executing actions in a given state. Denoting the agent's action choice as A, the expected reward when choosing action $a \in \mathcal{A}$ is

$$q(a) := \mathbb{E}_r \left[R \mid A = a \right], \tag{3.1}$$

where $q(a)$ is called the action value. It is state-independent, because multi-armed bandits have only a single state in their basic formulation. If the action values are known to the agent, it can simply choose the action with maximum value in each trial, thereby trivially maximizing its payoff. If the action values are initially unknown to the agent, it may engage in explorative interaction with the environment and thereby construct estimates $Q(a)$ of the real action values as sample averages:

$$Q(a) = \frac{\text{Sum of rewards received when executing } a}{\text{Number of times } a \text{ was executed}} = \frac{\sum_{i=0}^{N} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=0}^{N} \mathbb{1}_{A_i=a}}, \tag{3.2}$$

where the sum goes over all N trials, R_i is the reward in trial i and $\mathbb{1}_{A_i=a}$ is the indicator function that is 1 if action a was chosen in trial i and 0 otherwise. A practical implementation would be a running average that requires constant memory and constant time to update the estimated value when acquiring new samples. By the law of large numbers, the estimated value $Q(a')$ of action a' will converge to the real value $q(a')$ as the number of times action a' is sampled goes to infinity, assuming that the problem is stationary (i.e., reward probabilities don't change).

ϵ-greedy Policy The *exploration-exploitation trade-off* now becomes apparent: as the agent becomes more and more confident in its action value estimates, there is less to be gained by sampling actions with low estimated value. Clearly, the agent has to balance exploration of actions, which improves the value estimates, and exploitation, which serves the purpose of maximizing payoff. A simple method to balance exploration and exploitation is the ϵ-greedy policy which chooses the highest-value action with probability $1 - \epsilon$ and a random action with probability ϵ. For $\epsilon > 0$, the estimated action values converge to the real values and the policy choosing the action with highest value eventually becomes optimal. However, this is only an asymptotic guarantee. The exact choice of ϵ that is suitable for a given multi-armed bandit depends on the number of actions and the reward distribution. In general, a higher ϵ will lead to faster discovery of high-value actions but will be inferior to low ϵ policies in the long term. A possible mitigation would be to decay ϵ over time (Sutton and Barto, 1998).

Action Value Initialization A relevant parameter is the choice of the initial action values $Q(a)$ that represents an initial bias of the agent. If a full sample average as in Equation (3.2) is chosen, this bias is relevant only before an action is chosen and disappears thereafter. However, if a running average that incorporates the initial value is used, the bias will persist for some time depending on the averaging window. A possible choice is an *optimistic initialization*, where action values are initialized with values that are high relative to the rewards provided by the environment. In a (ϵ-)greedy policy, this will encourage choosing actions multiple times until the agent is confident that its optimism was unrealistic.

3.2.2 Policy Gradient Methods

In policy gradient, the idea is to adjust the parameters of a policy using stochastic gradient ascent on the expected return. A common choice for the agent's action-choice policy π is a soft-max function in terms of action preferences $\theta(a) \in \mathbb{R}$ which define the relative likelihood of actions:

$$\pi(a) = \frac{\exp(\theta(a))}{\sum_{a'} \exp(\theta(a'))} . \tag{3.3}$$

The soft-max function ensures that $\pi(a) > 0\,\forall a \in \mathcal{A}$ and $\sum_{a'} \pi(a') = 1$, so π can be interpreted as a discrete probability distribution over the set of actions. In each trial, the agent samples an action from π.

The goal of the agent is to maximize the expectation value of the reward,

$$\mathbb{E}_\pi[R] = \sum_{a'} \pi(a')q(a')\,, \tag{3.4}$$

where $q(a)$ are the actual action values of the multi-armed bandit as before. One method to achieve this would be to perform gradient ascent with respect to the action preference values $\theta(a)$:

$$\theta(a) \leftarrow \theta(a) + \alpha \frac{\partial \mathbb{E}[R]}{\partial \theta(a)}\,. \tag{3.5}$$

However, the action values $q(a)$ and therefore the gradient $\frac{\partial \mathbb{E}[R]}{\partial \theta(a)}$ are *a priori* unknown to the agent.

As we will see in the following, the agent can perform *stochastic* gradient ascent without explicit knowledge of the action values, where the average weight updates are proportional to the gradient in Equation (3.5). The idea is to turn the gradient of an expectation value (Equation (3.5)) into an expectation value, allowing to sample the random variable in each update step, so that the average update matches the expectation value.

$$\begin{aligned}
\frac{\partial \mathbb{E}_\pi[R]}{\partial \theta(a)} &= \frac{\partial}{\partial \theta(a)} \sum_{a'} \pi(a')q(a') && (3.6)\\
&= \frac{\partial}{\partial \theta(a)} \left[\sum_{a'} \pi(a')q(a') - C \sum_{a'} \pi(a')\right], && (3.7)
\end{aligned}$$

with a “nutritious zero” $\frac{\partial}{\partial \theta(a)} \sum_{a'} \pi(a') = 0$ as $\sum_{a'} \pi(a') = 1$ that allows to introduce a fixed reward baseline $C \in \mathbb{R}$,

$$\begin{aligned}
&= \frac{\partial}{\partial \theta(a)} \left[\sum_{a'} \pi(a')(q(a') - C)\right] && (3.8)\\
&= \sum_{a'} \left[(q(a') - C)\frac{\partial \pi(a')}{\partial \theta(a)}\right] && (3.9)\\
&= \sum_{a'} \left[(q(a') - C)\pi(a')\frac{1}{\pi(a')}\frac{\partial \pi(a')}{\partial \theta(a)}\right] && (3.10)\\
&= \sum_{a'} \left[(q(a') - C)\pi(a')\frac{\partial \log \pi(a')}{\partial \theta(a)}\right], && (3.11)
\end{aligned}$$

where we have used the identity $(\log f)' = \frac{1}{f} f'$,

$$= \mathbb{E}_\pi \left[(q(A) - C) \frac{\partial \log \pi(A)}{\partial \theta(a)} \right] \tag{3.12}$$

$$= \mathbb{E}_{\pi,r} \left[(R - C) \frac{\partial \log \pi(A)}{\partial \theta(a)} \right] \tag{3.13}$$

The baseline C does not affect the average weight update (because it is multiplied with a zero-mean quantity) but allows to reduce the variance by only capturing fluctuations around the baseline reward (Sutton and Barto, 1998). A common choice that approximates the optimal choice to this end is $C = \bar{R}$, where $\bar{R}$ represents the agent's estimate of $\mathbb{E}_\pi[R]$ which can be implemented, for example, using a running sample average (Greensmith et al., 2004).

Performing stochastic gradient ascent consists of sampling Equation (3.13) and updating parameters according to Equation (3.5). Using π as defined in Equation (3.3) and the derivative of the soft-max function, the gradient can be explicitly written as

$$\frac{\partial \mathbb{E}_\pi[R]}{\partial \theta(a)} = \mathbb{E}_{\pi,r} \left[(R - \bar{R})(\mathbb{1}_{A=a} - \pi(a)) \right] , \tag{3.14}$$

where $\mathbb{1}_{A=a}$ is one if and only if the agent's action choice variable A is equal to a. The stochastic gradient ascent update rule based on Equation (3.5) becomes

$$\theta(a) \leftarrow \theta(a) + \alpha(R - \bar{R})(\mathbb{1}_{A=a} - \pi(a)) \, . \tag{3.15}$$

For a given action choice and assuming the reward is higher than the average reward, this update rule increases the likelihood of that choice and decreases the likelihood of all others. The opposite is true if the reward is lower than the average reward. This method of solving multi-armed bandits is prone to get stuck in local minima of the expected reward, as is common for stochastic gradient ascent.

3.3 Biological Aspects

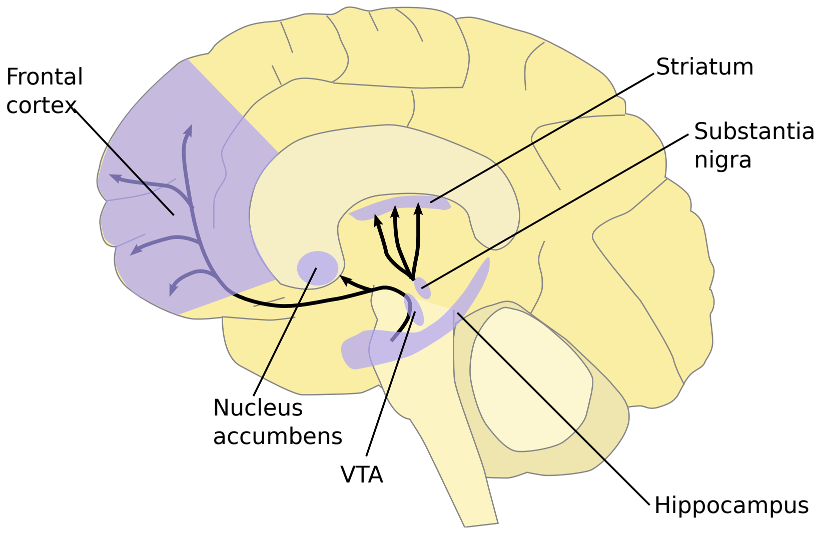


Fig. 3.2: Major dopamine pathways in the brain. Taken from NIDA and Quasihuman, 2012.

It is a basic observation that animal behavior is influenced by rewarding or punishing stimuli (Guttman, 1953; Moritz and Eberhard E Fetz, 2011; E E Fetz and Baker, 1973). The psychologist Edward Thorndike expressed this in his influential law of effect (Thorndike, 1911):

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.

This phenomenon was empirically researched by Thorndike and later by B. F. Skinner in terms of *operant conditioning* and *classical conditioning*. In operant conditioning, a voluntary behavior is associated with a rewarding or punishing consequence whereas in classical conditioning, an involuntary response and a stimulus are associated. A widely known form of classical conditioning was researched by Ivan Pavlov: dogs were conditioned to react to the ringing of a bell by starting to salivate (Pavlov, 2010).

The dopaminergic system is thought to be the neural structure that supports reinforcement learning; indeed, the administration of dopamine antagonists (substances which block dopamine) to animals prevents learning action sequences that lead to food rewards (Montague et al., 2004). Dopamine neurons, meaning neurons which release dopamine at their synaptic sites, are found in the Substantia Nigra (SN) and the Ventral Tegmental Area (VTA) (see Figure 3.2). Neurons in the VTA mainly project to the nucleus accumbens and the prefrontal cortex, whereas neurons in the SN project to the dorsal striatum. The former pathway is considered to be relevant for reward-based learning, while the latter is implicated in motor function (Wise, 2004). Importantly, the firing of dopamine neurons will lead to a spatially homogeneous increase of dopamine concentration in the targeted areas, due to the fact that dopamine diffuses over several microns before reuptake and that regions of high dopamine innervation have densely packed dopamine release sites (Dunnett et al., 2004). The activity of dopamine neurons therefore acts as a spatially diffuse neural signaling mechanism.

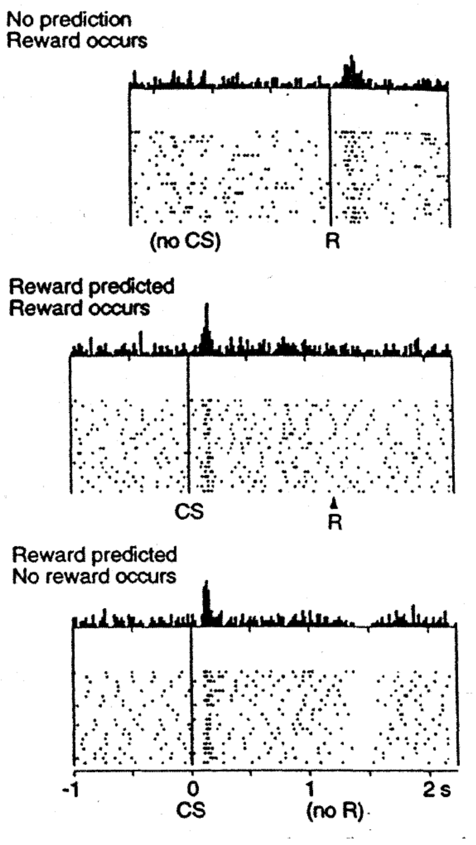


Fig. 3.3: Dopamine neurons encode predicted reward (see text). Taken from W Schultz et al., 1997.

Dopamine neurons were prominently found to encode a reward prediction error by their phasic activity, akin to the Temporal Difference (TD) error used in abstract reinforcement learning (W Schultz et al., 1997; Hollerman and Wolfram Schultz, 1998; Bayer and Glimcher, 2005). Single dopamine neuron recordings in monkeys revealed that an unpredicted reward elicits a burst of activity (see Figure 3.3, top panel). After learning, if the reward was coupled with a conditioned stimulus (CS) that occurs some time before the reward, this burst of activity occurs upon presentation of the CS but not at the time of the actual reward (middle panel). If the CS is presented but the reward is absent, a dip in the activity of the dopamine neuron is observed (bottom panel). In this sense, dopamine encodes a reward prediction error. However, the similarity to the TD error used in reinforcement learning is limited. For example, the phasic dopamine response is insensitive to temporal translations of the reward, which is not the case with the TD error, and dopamine neurons seem to also encode reward uncertainty via their tonic activity (Fiorillo et al., 2003; Fiorillo et al., 2005; Hart et al., 2015).

At the same time, dopamine receptor activation modulates spike-timing dependent plasticity (Pawlak and Kerr, 2008; Brzosko et al., 2015; Edelmann and Lessmann, 2011) and in this way relates reinforcing stimuli to synaptic plasticity and thereby learning processes. In the following section, we will discuss models of how a diffuse reward signal like that conveyed by Dopamine can influence synaptic plasticity in spiking neural networks so as to increase future reward.

3.4 Modeling in Spiking Networks

Reinforcement learning can be modeled in spiking neural networks using synapse-local learning rules amended with a global factor that models dopaminergic neuromodulation. In the following, we will discuss two classes of learning rules. The first is R-MAX (Frémaux and Wulfram Gerstner, 2015), which is derived using stochastic gradient-ascent on the expected reward. The second is R-STDP (Frémaux and Wulfram Gerstner, 2015), which is motivated heuristically but can be related to R-MAX.

3.4.1 The R-MAX Learning Rule

The approach taken in the R-MAX class of learning rules is to apply policy gradient methods (see Section 3.2.2) to stochastically spiking neuron models (Xie and Seung, 2004; Pfister et al., 2006; Baras and Meir, 2007; Florian, 2007; Urbanczik and Senn, 2009; Vasilaki et al., 2009; Fremaux et al., 2010). The learning goal is generally to reproduce a spatio-temporal pattern of post-synaptic spikes for a given pre-synaptic spike train, so reward is contingent on the precise spike times. For a given stochastic neuron model, the expected reward $\mathbb{E}[R]$ is given by integrating the reward over all possible neuronal outputs and their respective likelihood. The gradient of the expected reward with respect to a given synaptic weight, $\frac{\partial \mathbb{E}[R]}{\partial w_{ij}}$, can then be calculated in order to obtain a (stochastic) gradient ascent learning rule.

R-MAX for Spike Response Model with Escape Noise In the following, we will derive a R-MAX learning rule for the spike response model (SRM) with escape noise (Wulfram Gerstner, Kistler, et al., 2014), largely following Pfister et al., 2006. The SRM can be seen as a generalization of the LIF model. Consider a post-synaptic neuron i that is connected to N pre-synaptic neurons with synaptic weights w_{ij} where $j = 0, \ldots, N-1$. Let x_j denote the set of spike times of pre-synaptic neuron j and $\mathbf{x} = \{x_0, \ldots, x_{N-1}\}$ denote the set of all pre-synaptic spike trains.

In the SRM, neuron i has a membrane potential $u_i(t)$ and an instantaneous spiking probability $\rho_i(t) = g(u_i(t))$, where g is a non-linear function. The membrane potential given a pre-synaptic stimulus $u_i(t \mid \mathbf{x})$ is composed of the leak potential u_{leak}, the post-synaptic potentials (PSPs) evoked by pre-synaptic spikes $\epsilon(t)$ and refractory terms $\eta(t)$ that depend on the post-synaptic spike times y_i:

$$u_i(t \mid \mathbf{x}) = u_{\text{leak}} + \sum_{j=0}^{N-1} w_{ij} \sum_{t_f^{\text{pre}} \in x_j} \epsilon(t - t_f^{\text{pre}}) + \sum_{t_f^{\text{post}} \in y_i} \eta(t - t_f^{\text{post}})\theta(t - t_f^{\text{post}}) \,, \quad (3.16)$$

where $\theta(t)$ is the Heaviside function. Common choices for $\epsilon(t)$ and $\eta(t)$ are a double-exponential and an exponentially decaying negative term, respectively.

The inhomogeneous Poisson process defined by $\rho_i(t)$ allows to calculate the probability of post-synaptic spike times y_i (where y_i is the set of post-synaptic spikes caused by neuron i) in a time interval from $t = 0$ to $t = T$ (Pfister et al., 2006):

$$\begin{aligned} P(y_i \mid \mathbf{x}) &= \left(\prod_{t_f^{\text{post}} \in y_i} \rho_i(t_f^{\text{post}} \mid \mathbf{x}, y_i) \right) \exp\left(- \int_0^T \rho_i(s \mid \mathbf{x}, y_i) \mathrm{d}s \right) \\ &= \exp\left(\int_0^T (\log(\rho_i(s \mid \mathbf{x}, y_i))\gamma_i(s) - \rho_i(s \mid \mathbf{x}, y_i) \mathrm{d}s) \right) , \end{aligned} \tag{3.17}$$

where $\gamma_i(t) = \sum_{t_f^{\text{post}} \in y_i} \delta(t - t_f^{\text{post}})$ is the Dirac-delta spike train based on spike times contained in y_i. Using $P(y_i \mid \mathbf{x})$ and a reward function $R(y_i)$, the expected reward is

$$\mathbb{E}[R \mid \mathbf{x}] = \sum_{y_i} P(y_i \mid \mathbf{x}) R(y_i) \,. \tag{3.18}$$

Analogously to the derivation of policy gradient in Section 3.2.2, the derivative of this quantity with respect to synaptic weight w_{ij} is given by

$$\frac{\partial \mathbb{E}[R \mid \mathbf{x}]}{\partial w_{ij}} = \mathbb{E}\left[(R(Y_i) - C) \frac{\partial \log P(Y_i \mid \mathbf{x})}{\partial w_{ij}} \right] \tag{3.19}$$

where Y_i is the random variable underlying realizations y_i and $C \in \mathbb{R}$ is an arbitrary baseline. This gradient can then be used in stochastic gradient ascent, where the agent uses samples of Equation (3.19) to calculate weight updates. The baseline C can reduce the variance of these gradient estimates, with an empirical reward average $C = \bar{R}$ that estimates $\mathbb{E}[R]$ being a reasonable approximation of the optimal choice (Fremaux et al., 2010; Greensmith et al., 2004). This allows the gradient estimates to capture the fluctuations of reward around the average reward, decreasing gradient variance and thereby the magnitude of individual gradient updates.

The derivative of the log-likelihood with respect to synaptic weight, $\frac{\partial \log P(y_i \mid \mathbf{x})}{\partial w_{ij}}$, can be seen as an eligibility trace e_{ij} that captures deviations from the average spiking behavior:

$$\begin{aligned} e_{ij} &:= \frac{\partial \log P(y_i \mid \mathbf{x})}{\partial w_{ij}} \\ &= \int_0^T \frac{\mathrm{d} \log \rho_i(s \mid \mathbf{x}, y_i)}{\mathrm{d}u_i} \left[\gamma(s) - \rho_i(s \mid \mathbf{x}, y_i) \right] \sum_{t_f^{\text{pre}} \in x_j} \epsilon(s - t_f^{\text{pre}}) \mathrm{d}s \,, \end{aligned} \tag{3.20}$$

with

$$\mathbb{E}[e_{ij}] = \sum_{y_i} P(y_i \mid \mathbf{x}) \frac{\partial \log P(y_i \mid \mathbf{x})}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{y_i} P(y_i \mid \mathbf{x}) = 0\,. \tag{3.21}$$

The quantity e_{ij} is called an eligibility trace because it captures the role of synapse ij in producing non-average spiking behavior and thereby quantifies the eligibility of synapse ij for reward-based changes. In our formulation of stochastic gradient ascent, we then use samples of episodes of length T to calculate weight updates:

$$\Delta w_{ij} = \alpha(R - \bar{R})e_{ij} \tag{3.22}$$

with learning rate α. Taking the expectation value of Δw_{ij} reveals that these weight updates capture the covariance of reward and eligibility trace:

$$\begin{aligned}\mathbb{E}[\Delta w_{ij}] &\propto \mathbb{E}[(R - \bar{R})e_{ij}] \\ &= \mathbb{E}[Re_{ij}] - \bar{R}\mathbb{E}[e_{ij}] \\ &\approx \mathbb{E}[Re_{ij}] - \mathbb{E}[R]\mathbb{E}[e_{ij}] \\ &= \mathrm{Cov}(R, e_{ij})\,. \end{aligned} \tag{3.23}$$

This learning rule can therefore be understood as correlating reward R and the eligibility trace e_{ij}, which in turn quantifies the participation of a given synapse in producing the post-synaptic spike train and deviations from expected post-synaptic activity. To calculate the eligibility trace, we require the post-synaptic neuron's membrane potential and its "activation function" $\rho_i(t)$, which are two quantities that are presumably not available at the synaptic site.

3.4.2 The R-STDP Learning Rule

In the following section, R-STDP is introduced: a learning rule that uses only synapse-local information and can be understood as correlating reward and synaptic activity broadly similar to R-MAX. R-STDP is a heuristically motivated learning rule of the form

$$\Delta w_{ij} = \alpha(R - C)e_{ij}^{\mathrm{STDP}} \tag{3.24}$$

where e_{ij}^{STDP} is a STDP-based eligibility trace, $\alpha \in \mathbb{R}$ is a learning rate and $C \in \mathbb{R}$ a reward baseline (Frémaux and Wulfram Gerstner, 2015). The eligibility trace is a low-pass filter of the outcome of classic, unsupervised STDP:

$$\frac{\mathrm{d}}{\mathrm{d}t} e_{ij}^{\mathrm{STDP}} = -\frac{e_{ij}^{\mathrm{STDP}}}{\tau_e} + \mathrm{STDP}\,, \tag{3.25}$$

where STDP denotes the addition of spike time coincidences with a positive contribution of pre-before-post coincidences and a negative contribution of post-before-pre coincidences. The idea of R-STDP is therefore to modulate the outcome of unsupervised STDP with a reward-dependent term.

This learning rule has been used in a number of publications, for example, to reproduce spatio-temporal spike patterns for a given input (Farries and Fairhall, 2007; Florian, 2007; Legenstein, Pecevski, et al., 2008; Fremaux et al., 2010), to solve a water-maze navigation task (Vasilaki et al., 2009), to reproduce classical conditioning (Izhikevich, 2007), to reproduce the seminal biofeedback experiment by Fetz & Baker (Legenstein, Pecevski, et al., 2008) and to improve the classification performance of a convolutional spiking network (Mozafari et al., 2018).

Using the empirical reward average as baseline, $C = \bar{R}$, R-STDP becomes sensitive to the covariance of reward and synaptic activity:

$$\begin{aligned}\mathbb{E}[\Delta w_{ij}] &\propto \mathbb{E}[R e_{ij}^{\mathrm{STDP}}] - \bar{R}\mathbb{E}[e_{ij}^{\mathrm{STDP}}] \\ &\approx \mathbb{E}[R e_{ij}^{\mathrm{STDP}}] - \mathbb{E}[R]\mathbb{E}[e_{ij}^{\mathrm{STDP}}] \\ &= \mathsf{Cov}(R, e_{ij}^{\mathrm{STDP}})\,. \end{aligned} \tag{3.26}$$

The balance of the depressing and potentiating parts of the STDP window were found to be largely irrelevant for learning (Fremaux et al., 2010). The choice of the reward baseline C, however, is critical. This is laid out in the following.

Relating R-STDP and R-MAX With the choice $C = \bar{R}$, R-STDP and R-MAX can be related in the sense that they are both covariance-based learning rules which capture the correlation of reward and a Hebbian measure that depends on the pre- and post-synaptic activity at a given synapse. A notable difference is that the R-MAX eligibility trace averages to zero by definition, $\mathbb{E}[e_{ij}] = 0$, which ensures that the average weight changes are insensitive to baselines $C \neq \bar{R}$ and $\mathbb{E}[\Delta w_{ij}] = \mathsf{Cov}(R, e_{ij})$ independent of C. This is not the case for R-STDP, where $\mathbb{E}[e_{ij}^{\mathrm{STDP}}] \neq 0$ in general, which means that the choice of C is critical: adding a constant offset $C = \bar{R} + C_0$, $C_0 \in \mathbb{R}$ causes a reward-independent term in the expected weight change:

$$\mathbb{E}[\Delta w_{ij}] \propto \mathsf{Cov}(R, e_{ij}^{\mathrm{STDP}}) + C_0 \mathbb{E}[e_{ij}^{\mathrm{STDP}}]\,. \tag{3.27}$$

Therefore, in a reward-based learning task, offsets $C_0 \neq 0$ generally prevent learning in the case of R-STDP, but not in R-MAX, as empirically demonstrated in Fremaux et al., 2010. This is because a constant offset causes an unsupervised reward-independent bias in the weight changes, leading to divergent behavior.

This observation has ramifications in the case of several learning tasks that should be learned in parallel (e.g., if there exist multiple input patterns which should elicit different network responses). In this case, R-STDP requires a task-specific reward-average $C = \bar{R}_{\text{task}}$ in order to learn while R-MAX does not, for the same reasons as stated above. If the reward-average were task-unspecific, Equation (3.26) would not hold and a unsupervised bias would be introduced. Failure of R-STDP to learn under this circumstance was empirically demonstrated in Fremaux et al., 2010. Therefore, R-STDP requires task-specific rewards in order to remain a unbiased covariance-based learning rule.

For our experiment on HICANN-DSLv2, we use the R-STDP learning rule because it requires quantities that are readily available at each synapse (i.e. the correlation measures (see Section 2.1.2)) and the update rule can be efficiently calculated using the PPU.

Experimental Results 4

Ever tried. Ever failed.
No matter. Try again.
Fail again. Fail better.

— **Samuel Beckett**

HICANN-DLSv2, as described in Section 2.2, provides an accelerated neuromorphic substrate for the implementation of spiking neural networks and is especially well suited for learning experiments. This section describes one of the first functional learning experiments on the chip. In the experiment, the emulated network learns via reinforcement learning in a virtual environment that is simulated on the PPU. This allows the experiment to run on the chip fully autonomously, i.e., without external off-chip communication. The aim of this section is to describe the experimental setup, provide the measurement results as well as to analyze and interpret the obtained data.

The virtual environment resembles the classic Pong arcade video game. Using all of its 32 neurons and 1024 synapses, the chip learns to map ball positions to target paddle positions. Because it is rewarded for proper aiming, the chip learns to trace the ball by smooth pursuit. It does so by trial-and-error learning: trial-to-trial variations of neuronal firing rates mediated by noise in the analog neurons lead to action exploration and drive learning.

The experiment demonstrates that all chip components can function in concert: it uses all neurons and synapses, all spike counters, all correlation sensors and the PPU. By comparing it to a software simulation of the involved neural network, we demonstrate that the neuromorphic emulation is at least an order of magnitude faster and three orders of magnitude more energy-efficient. Besides this, we find that temporal variability, usually regarded as a nuisance, takes on a functional role: it mediates action exploration and drives learning. The deterministic software simulation is unable to learn properly without injected noise, as it lacks the trial-to-trial variability of spike rates required for trial-and-error learning.

Crucially, we find that fixed-pattern noise, which expresses itself for example in variability of neuronal parameters, is implicitly compensated by the learning process.

This is important as parameter variability is unavoidable when using analog circuits. Our experiment demonstrates that calibration efforts can be largely supplanted by learning. We show that the learned weight matrix is adapted to the characteristics of the physical substrate: the learning process adapts synaptic weights in such a way as to compensate for the difference in excitability of the post-synaptic neuron.

4.1 Experimental Setup

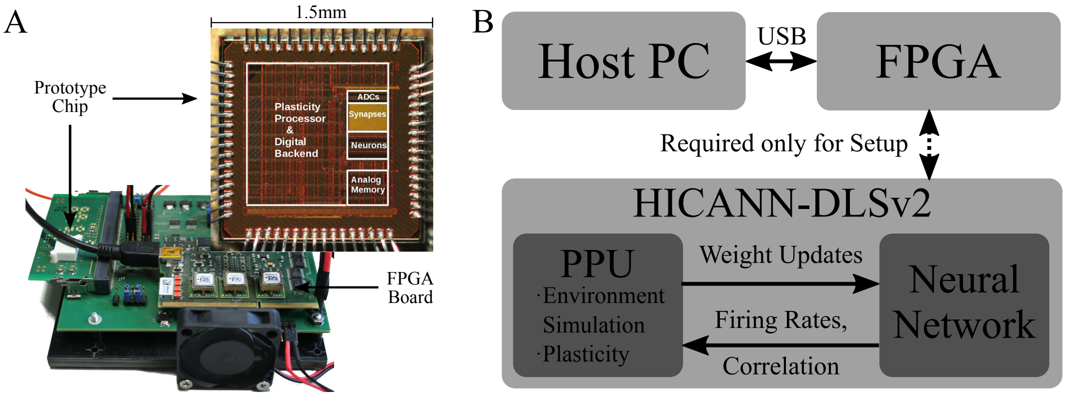


Fig. 4.1: **A**: HICANN-DLSv2 (foreground) and prototyping board (background). **B**: The host computer communicates with the FPGA via USB. The FPGA in turn configures the HICANN-DLSv2, which executes the experiment fully autonomously.

The physical experimental setup consists of a HICANN-DLSv2 chip mounted on a prototyping board (see Figure 4.1 A). The chip is accessed and configured via the FPGA, which in turn is accessed by the host computer using a USB connection. In the experiment described here, the FPGA has no function beyond chip configuration and reading out results from the chip's memory, as the experiment runs on the chip fully autonomously once it is configured. The PPU simulates the virtual environment and calculates weight updates based on firing rates and synaptic correlation of the on-chip neural network (see Figure 4.1 B).

4.1.1 Learning Task

Virtual Environment The virtual environment is a simplified version of the Pong video game. It consists of a two-dimensional playing field with three closed (reflecting) walls, a ball and a paddle positioned along the open side. Upon experiment initialization, the ball is positioned in the middle of the playing field and proceeds to move with a fixed velocity $\vec{v}$ toward a random direction.

The paddle is controlled by the neural network, i.e., the network output is interpreted as a target paddle position along the open side. It moves toward its target position with constant velocity v_p. If the ball hits the paddle, it is reflected; if the paddle does

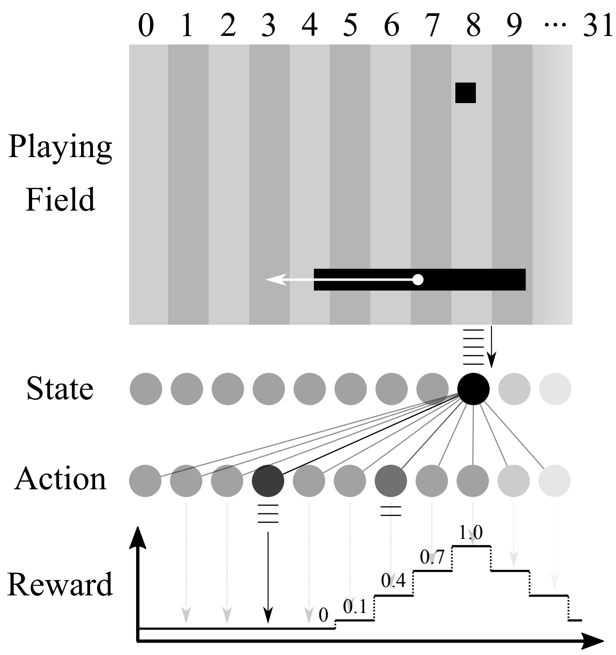


Fig. 4.2: **A**: The playing field is discretized into 32 columns. The two-layer neural network transmits the ball position to the action units (the chip's physical neurons) via a uniform spike train, where the transmitting input unit identifies the column of the ball. The action unit firing the most spikes dictates the column towards which the paddle moves. Reward is provided depending on how well ball column and target column match (see reward window depicted at the bottom). In the depicted situation the ball is in column 8 and therefore, input unit 8 transmits a spike train. As the winning output neuron, i.e., target paddle column (3) and ball column (8) don't match at all, reward is zero.

not catch the ball, the game is reset (i.e., ball and paddle positions are set to the starting position and a new random direction is determined). Reward is provided at each time step to the network depending on its aiming accuracy.

Learning Task Mechanics The goal of the learning task is to map ball positions to target paddle positions. The playing field is discretized into 32 columns along the axis on which the paddle moves (see Figure 4.2). A two-layer neural network consisting of an input and an output layer receives the ball position as input and defines the target paddle position using the activity of its output neurons. The output neurons are the actual 32 hardware neurons, while the "virtual" input neurons provide spike input to the output neurons. Each column corresponds to an input unit: if the ball is in column k, input unit k transmits a uniform spike train towards the output units. Initially, all input units are connected to all output units; initial weights are drawn from a Gaussian distribution (see Table 4.1).

The spikes from the input unit will cause some pattern of spiking activity in the output units. We denote the number of spikes of the output units as ρ_i where $i \in \{0..31\}$. The output unit with the highest amount of elicited spikes defines the target column of the paddle, $j = \mathrm{argmax}_i \, \rho_i$. The environment then provides reward $R \in [0, 1]$ depending on how well the ball column k and the paddle column j match:

$$R = \begin{cases} 1 - |j-k| \cdot 0.3 & \text{if } |j-k| \leq 3\,, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

See Figure 4.2 for a visualization of the reward window defined by this equation. This means that the network receives zero reward if its aim is off by more than three columns and non-zero reward otherwise, with a maximum reward of one if the columns perfectly match. The non-zero reward window matches the paddle length, i.e., the network receives reward only when it aims the paddle such that it would be able to catch the ball, with off-center aiming being rewarded less.

In terms of reinforcement learning, the learning problem consists of 32 32-armed bandits (see Section 3.2), as each state defines a 32-armed bandit.

4.1.2 Learning Rule

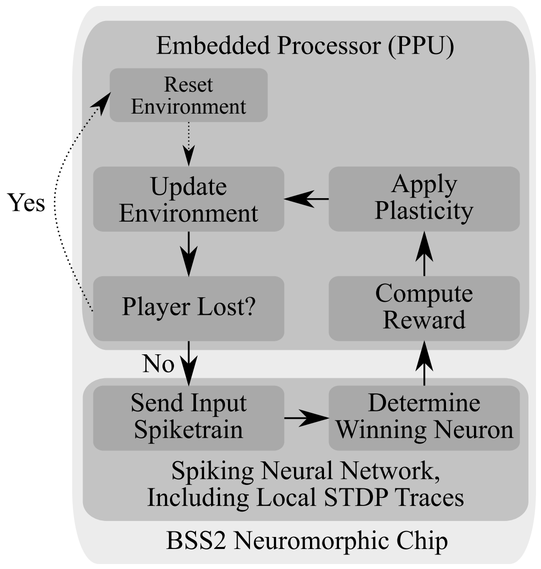


Fig. 4.3: Flowchart of the experiment loop running fully autonomously on HICANN-DLSv2.

The PPU calculates weight updates using a specific learning rule, namely R-STDP as described in Section 3.4.2. It maintains an expected reward $\bar{R}_k$ (an exponentially weighted moving average) for every possible state, i.e., every possible discretized ball position k with $k \in \{0..31\}$. This expected reward is subtracted from the instantaneous reward R to yield the neuromodulating factor $R - \bar{R}_k$. The expected reward needs to be state-specific, as indicated by the index k, in order to avoid inappropriately biasing the weight changes (Fremaux et al., 2010, see Section 3.4.2). For a given synaptic weight w_{mn}, this factor is used together with the digitized causal correlation value A^+_{mn} (see Section 2.1.2) and the learning rate β to calculate the weight update:

$$\Delta w_{mn} = \beta \cdot \left(R - \bar{R}_k\right) \cdot A^+_{mn}\,. \tag{4.2}$$

The digitized correlation values have been offset-corrected and right shifted by one bit in order to reduce noise. Upon receiving a reward, this weight update is calculated and applied for all synapses.

The expected reward $\bar{R}_k$ is updated to provide an exponentially weighted moving average over the last rewards:

$$\bar{R}_k \leftarrow \bar{R}_k + \gamma \left(R - \bar{R}_k \right) \ , \tag{4.3}$$

where the parameter γ controls the influence of previous iterations.

An entire experiment iteration consists of environment simulation and calculating and applying weight updates: see Figure 4.3 for a visualization and Figure 4.4 for pseudo-code.

```
Result: One iteration (time step) of simulated pong with one paddle action
        controlled by BSS2.
begin
    UpdateEnvironment() ;
    if PlayerHasLost() then
        ResetEnvironmentSimulation();
    end
    $j \leftarrow$GetColumnContainingBall();
    SendSpikeTrainFromInputNeuron(j);
    $i \leftarrow$ GetWinningOutputNeuron();
    switch $|i - j|$ do
        case 0 do $R \leftarrow 1$;
        case 1 do $R \leftarrow 0.7$;
        case 2 do $R \leftarrow 0.4$;
        case 3 do $R \leftarrow 0.1$;
        otherwise do
            $R \leftarrow 0$;
        end
    end
    if IsTheFirstIteration() then
        $\langle R \rangle_j \leftarrow R$;
    end
    $S \leftarrow R - \langle R \rangle_j$;
    $\langle R \rangle_j \leftarrow \bar{R} + 0.5 \cdot S$;
    for $m \leftarrow 0$ to 31 do
        for $n \leftarrow 0$ to 31 do
            $w_{mn} \leftarrow w_{mn} + \beta \cdot S \cdot A^{+}_{mn}$;
        end
    end
end
```

Fig. 4.4: Pseudo-code of single experiment run (visually depicted in Figure 4.3).

4.1.3 Learning Progress

In order to quantify learning progress, we use two quantities which are both based on reward but differ in their interpretation. The *mean expected reward*

$$\langle \bar{R} \rangle = \frac{1}{32} \sum_{i=0}^{31} \bar{R}_i \tag{4.4}$$

is the average over all 32 expected rewards. It can be interpreted as an average aiming accuracy, quantifying how well the network centers the paddle below the ball.

As the paddle spatially extends over the whole reward window, it is able to catch the ball in a given state even with an expected reward of less than one. Therefore, we also consider the *performance* in playing pong:

$$P = \frac{1}{32} \sum_{i=0}^{31} \lceil R_i \rceil \, , \tag{4.5}$$

where R_i is the last reward received in state i and $\lceil \cdot \rceil$ denotes the ceiling function. This quantity can be interpreted as the percentage of states where the agent is able to catch the ball.

4.1.4 Hyperparameter Optimization

Prior to obtaining the experimental results given in this thesis, we used a hyperparameter optimization procedure to find proper parameters for the neurons and synapses. During this optimization, we kept parameters concerning the game dynamics, the initial weight distribution and the input spike train fixed. We used a Bayesian parameter optimizer within the SCIKIT-OPTIMIZE software package (Head et al., 2018). We took 30 random, initial data points and proceeded to perform 300 iterations with each evaluated parameter set being determined by the FOREST_MINIMIZE optimization algorithm (default parameters: maximizing expected improvement, extra trees regressor model, 10000 acquisition function samples, target improvement of 0.01). Each data point consisted of the mean expected reward obtained after 50000 experiment iterations. All neuronal and synaptic parameters were globally (not individually) subject to the optimization, i.e., all neuronal time constants and voltages (τ_{mem}, τ_{ref}, τ_{syn}, v_{leak}, v_{reset}, v_{thresh}) as well as the time constant and amplitude of the causal correlation sensors (η_+, τ_+).

The result of this hyperparameter optimization was a common set of parameters for all neurons and synapses (see Table 4.1). These parameters were given in the pre-calibration domain, i.e., in terms of microseconds. The calibration translated these values to hardware parameters.

We conducted this procedure on three different chips in order to test the transferability of results. All results given in the following section have been acquired using the first parameter set resulting from an optimization on chip #1, if not specified otherwise.

Used Hyperparameters

Symbol	Description	Value
	Neuromorphic hardware	BrainScaleS 2 (2nd prototype version)
N	Number of action/output neurons (LIF)	32
N_S	Number of state/input units	32
N_syn	Number of synapses	$32 \cdot 32 = 1024$
N_spikes	Number of spikes from input unit	20
T_ISI	ISI of spikes from input unit	$10\,\mu\text{s}$
w	Mean of distribution of initial weights (digital value)	14
σ_w	Standard deviation of distribution of initial weights	2
L	Length and width of quadratic playing field	1
$\|\vec{v}\|_1$	L1-norm of ball velocity	0.025 per iteration
v_p	Velocity of paddle controlled by BSS2	0.05 per iteration
r_b	Radius of ball	0.02
r_p	Length of paddle	0.20
γ	Decay constant of reward	0.5
β	Learning rate	0.125
	NEST version (software simulation)	2.14.0
	NEST timestep	0.1 ms
	CPU (software simulation, one core used)	Intel i7-4771

		Set #1 (standard)	Set #2	Set #3
τ_mem	LIF membrane time constant	$28.5\,\mu\text{s}$	$18.4\,\mu\text{s}$	$24.8\,\mu\text{s}$
τ_ref	LIF refractory time constant	$4\,\mu\text{s}$	$14.3\,\mu\text{s}$	$13.8\,\mu\text{s}$
τ_syn	LIF excitatory synaptic time constant	$1.8\,\mu\text{s}$	$2.4\,\mu\text{s}$	$1.4\,\mu\text{s}$
v_leak	LIF leak voltage	0.62 V	0.56 V	0.87 V
v_reset	LIF reset voltage	0.36 V	0.36 V	0.30 V
v_thresh	LIF threshold voltage	1.28 V	1.31 V	1.21 V
η_+	Amplitude of correlation function a_+ (digital value)	72	114	70
τ_+	Time constant of correlation function a_+	$64\,\mu\text{s}$	$80\,\mu\text{s}$	$60\,\mu\text{s}$

Tab. 4.1: Parameters used in the experiment. The different parameter sets are the result of optimizing parameters on three different chips. If not mentioned otherwise, results were obtained using set #1. Abbreviations used in table: LIF: Leaky Integrate-and-Fire; ISI: Inter-Spike Interval.

4.1.5 Software Simulation

Using the commonly used NEST v2.14.0 SNN simulator (Peyser et al., 2017) and Python, we implemented a software simulation of the experiment. This allowed us to compare the speed and energy efficiency of the neuromorphic emulation to this particular software simulation. We set the LIF parameters of the simulated neurons to be the target parameters of the emulated neurons. In contrast to the neuromorphic emulation, these simulated neurons are completely identical and behave in a deterministic way.

We used the IAF_PSC_EXP integrate-and-fire neuron model, which uses exponential post-synaptic currents and current-based synapses. Using the NOISE_GENERATOR module in NEST, we could optionally inject Gaussian current noise into each neuron. The causal correlation values were calculated in Python using the spike times provided by NEST, with the amplitude and time constant chosen to match the mean values on HICANN-DLSv2. All hyperparameters (game dynamics, learning rate) were chosen to match the emulation. Synaptic weights were scaled to a dimensionless quantity and discretized so as to provide a neuronal activation function (output spike rate vs. synaptic weight) similar to the ones observed on HICANN-DLSv2. The last step is necessary as there is no calibration routine that translates LIF model weights to hardware weights. In contrast to the neuromorphic emulation, where all synapses are updated in every iteration, the software simulation calculated weight updates only for those synapses which actually transmitted spikes. This is because the correlation values for all other synapses are zero in a perfect software simulation.

The source code has been made publicly available (Wunderlich, 2019).

4.2 Experimental Results

4.2.1 Fixed-Pattern Noise and Temporal Variability

This section characterizes the impact of fixed-pattern noise and temporal variability on different components of the experiment. See Section 2.2.1 for a definition of fixed-pattern noise and temporal variability.

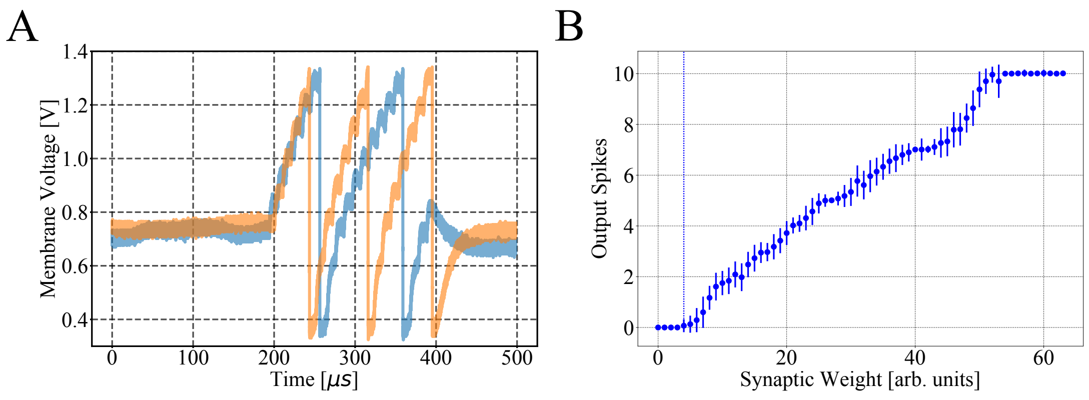


Fig. 4.5: **A**: Membrane potential of a single neuron on HICANN-DLSv2 in two trials, where the same spike train as used in the experiment is delivered in each trial. One trial elicits two spikes, the other three. **B**: Activation function of a single neuron, i.e., number of output spikes (mean and standard deviation) as function of synaptic weight when transmitting the spike train used in the experiment. The dashed vertical line denotes the firing threshold weight.

Spike Rates The input spike train transmitted to the output neurons is a fixed, uniform spike train and spikes arrive with little to no timing jitter. Despite this fact, the number of spikes elicited by the post-synaptic neuron varies between repetitions of the same experiment. This is due to temporal variability in neurons. See Figure 4.5 A for membrane traces where an identical input spike train causes two spikes in one trial and three in another. This variability leads to noisy activation functions (number of output spikes as a function of synaptic weight for the given spike train). One exemplary activation function is shown in Figure 4.5 B. It is precisely this variability of firing rates that causes action exploration in the experiment, as the population of output neurons exhibits different firing patterns over many trials.

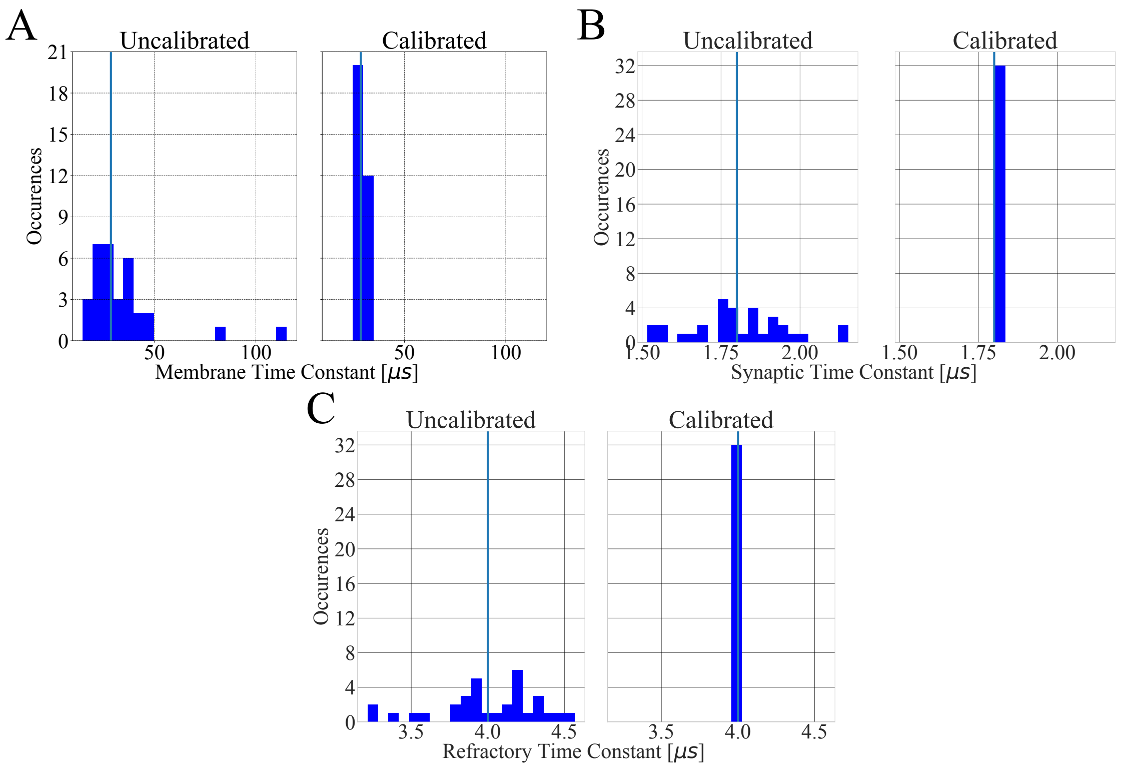
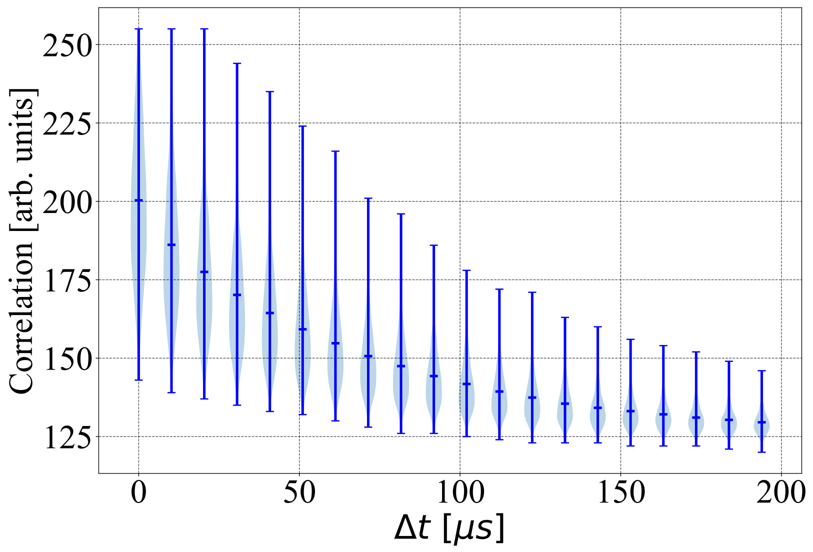


Fig. 4.6: Calibrated and uncalibrated parameter distributions over all 32 neurons for the membrane time constant (**A**), synaptic time constant (**B**) and refractory time period (**C**). The blue vertical line shows the target values.

Neuron Parameters In addition to the temporal variability of a single neuron, the population of all neurons exhibits a fixed pattern of variability in neuron parameters. The neuronal calibration (Stradmann, 2016) is able to compensate this variability to some degree by setting hardware parameters on a per-neuron basis and can be used to translate target LIF time constants to hardware parameters. No calibration for LIF voltages was available, but the variability among these parameters is rather minor compared to the time constants (Stradmann, 2016). The impact of the calibration on the time constant distribution over all neurons is shown in Figure 4.6, where the uncalibrated state is defined using the average calibrated hardware parameter for all neurons. The target values are the same as in the main experiment (see Table 4.1). Variability in neuron parameters leads to variable neuronal activation functions in terms of slope and offset (i.e. excitability, an example activation function is depicted in Figure 4.5 B).

The impact of the calibration is strongest in case of the membrane time constant. This is because the target membrane time constant is at the edge of the specification where neurons are especially sensitive. The uncalibrated state of neuron parameters as defined here will be used in the following to investigate the effects of an imperfect calibration on learning performance.



H

Fig. 4.7: Violin plot of the characteristic curves of all 1024 causal correlation sensors.

Correlation Sensors For a given set of global parameters, each of the 1024 correlation sensors provides a different correlation curve due to fixed-pattern noise (see Figure 4.7). As mentioned in Section 2.1.2, each individual correlation sensor has four calibration bits which can be used to offset the effect of fixed-pattern noise to some degree. However, we did not find a calibration of the correlation sensors to be necessary for learning in this experiment.

4.2.2 Learning Performance

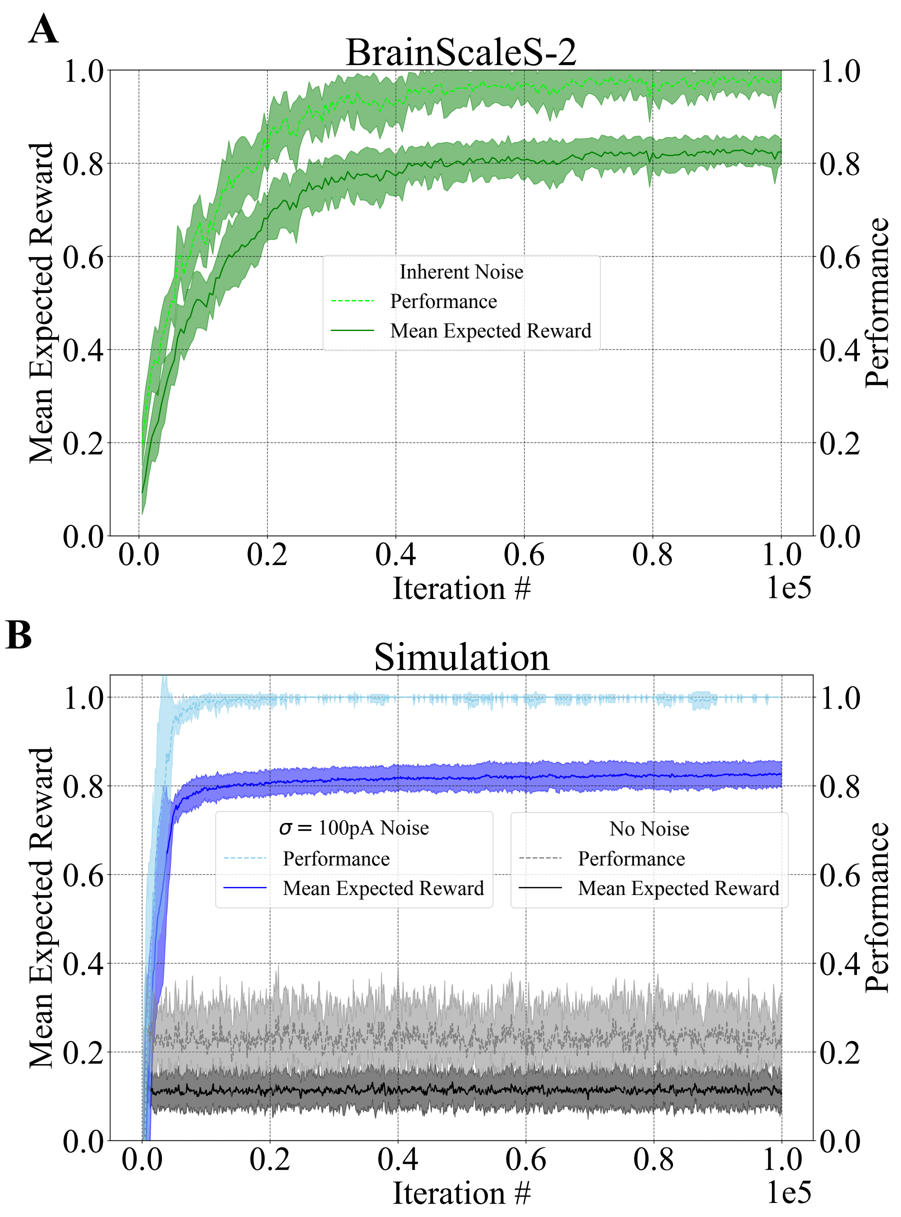


Fig. 4.8: Mean expected reward and Pong performance as learning progresses for both HICANN-DLSv2 and the software simulation (with and without injected noise). We plot mean and standard deviation for ten different randomly initialized weight matrices. **A**: HICANN-DLSv2 uses its intrinsic hardware noise to learn. **B**: The software simulation without noise is unable to learn beyond chance level. Injecting Gaussian current noise into each neuron with zero mean and $\sigma = 100\,\mathrm{pA}$ enables action-exploration and thereby learning. The noisy software simulation converges faster than HICANN-DLSv2 because the idealized simulated scenario contains no fixed-pattern noise and Gaussian current noise does not reflect the temporal variability present on HICANN-DLSv2.

This section demonstrates the capability of HICANN-DLSv2 to solve the learning task using hardware noise for action exploration. Figure 4.8 shows the progress of learning over 10^5 iterations for both HICANN-DLSv2 and the software simulation with and without injected noise. Both measures, the mean expected reward and the Pong performance, are used to quantify learning. We performed ten experiments with a different randomly initialized weight matrix in each case, the results are given in Figure 4.8. In case of the software simulation, we conducted separate measurements without noise and with injected Gaussian current noise with zero mean and a standard deviation of $\sigma = 100\,\mathrm{pA}$ (injected independently into each neuron).

Temporal Variability on HICANN-DLSv2 Drives Learning On HICANN-DLSv2, the trial-to-trial variations of neuronal firing rates for the given parameters are sufficient to drive learning and to solve the learning task (see Figure 4.8 A). This is in contrast to the deterministic software simulation without noise (see Figure 4.8 B), where the absence of an action exploration mechanism prevents learning performance beyond the mean expected reward expected for a uniformly random agent which is $\langle \bar{R} \rangle_{\text{rand}} \approx 0.10$. The small, ongoing fluctuations in case of the software simulation without noise are due to the fact that the winning neuron is selected randomly if several neurons fire the same number of spikes. Injecting current noise into each neuron enables action exploration and thereby learning, leading to a converged performance level roughly similar to HICANN-DLSv2. The simulation with injected noise converges faster and to a higher level of Pong performance. This is due to the fact that the simulation starts from a balanced, completely unbiased state, contains no fixed-pattern noise and that the Gaussian current noise is not a good model of the temporal variability present on HICANN-DLSv2.

In Figure 4.8 A , we show that learning on HICANN-DLSv2 is reproducible, with little variation in convergence. As might be expected, the converged Pong performance is larger than the mean expected reward. This means that the agent aims slightly off-center on average but is still able to catch the ball.

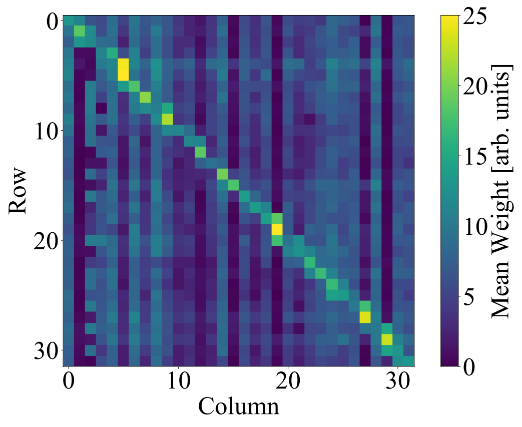


Fig. 4.9: Weight matrix after learning, averaged over the ten trials depicted in Figure 4.8. Elements on and near the diagonal dominate, which is the goal of the learning task.

Learned Weight Matrix The goal of the learning task is to map ball positions to target paddle positions as good as possible. In terms of the neural weight matrix, this means the initial all-to-all connectivity should be ideally reduced to a one-to-one mapping of states to actions, which corresponds to a diagonal matrix in our case. Indeed, Figure 4.9 shows that learning on HICANN-DLSv2 leads to a diagonally dominant weight matrix. Weights on the first off-diagonals are also strengthened, because these correspond to rewarded actions, albeit with a reward of less than one. The noticeable pattern of vertical lines in the weight matrix is caused by neuronal fixed pattern noise: one column in the weight matrix corresponds to one neuron and the learning process adapts weights so as to compensate for the excitability of the respective neuron. This leads to different weight patterns for each neuron and suggests that synaptic fixed-pattern noise plays a lesser role.

Temperature Dependence We found a notable temperature dependence of learning performance at the given parameters when continuously performing experiments while controlling the ambient temperature using a Binder model KT 53 temperature cabinet (see Figure 4.10). This is likely due to the fact that the neurons are operated at a point in parameter space where the membrane time constant is most sensitive to variations in the corresponding hardware parameter. In other words, changes as small as a single LSB in the hardware parameter controlling the membrane time constant will have a significant impact on neuron behavior (see the calibration curves in Stradmann, 2016). Changing the temperature while holding all parameters fixed likely causes a systematic variation of the DAC ramp used by the capacitive memory to convert digital hardware parameters to analog voltages. This change is then amplified by the supra-linear calibration curves which provide the LIF time constants as a function of digital hardware parameters (see Stradmann, 2016).

The digital hardware neuron parameters are converted to analog values using a chip-internal DAC (see Section 2.2). The DAC functions using a voltage ramp which is controlled by a current generated on the prototype board, which is in turn parameterized by three parameters for DACs on the prototype board (CAPMEM_IBUF, CAPMEM_IBIAS, CAPMEM_IREF). We found that temperature-induced performance

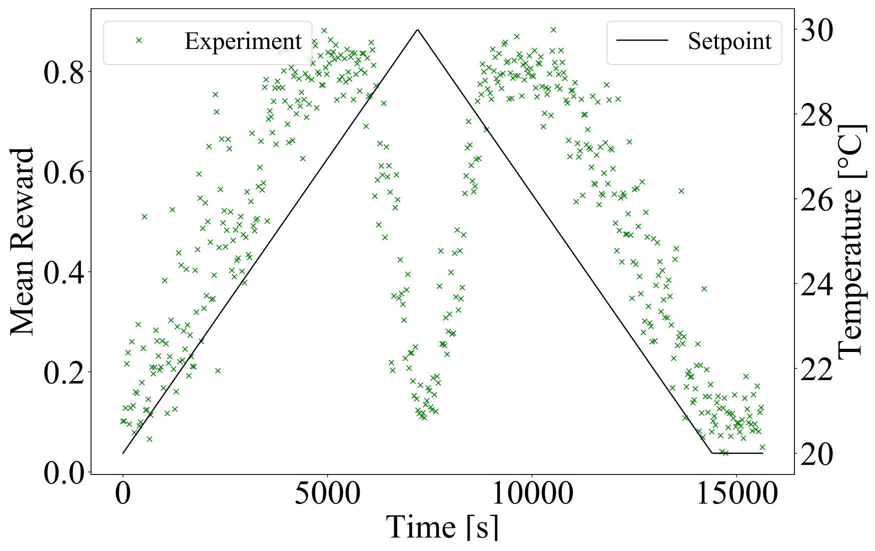


Fig. 4.10: Mean reward obtained after 50000 iterations while sweeping ambient temperature using a temperature cabinet. We plot the setpoint of the temperature cabinet over time because direct temperature measurements were not available.

loss can be compensated by changing the CAPMEM_IREF parameter on a scale of a few LSB which supports the above hypothesis. Most of the following experimental results were obtained under controlled temperature conditions using the Binder model KT 53 temperature cabinet set to 26 °C. Future experiments could mitigate this issue by moving into less sensitive areas of the neuronal parameter space.

4.2.3 Learning Is Calibration

In this section we show that the learning process implicitly compensates variability in neuronal parameters by correlating neuronal excitability with learned synaptic weights. Furthermore, we show that shuffling the assignment of output unit to physical neuron after learning leads to a loss in performance, as this represents an undoing of the learned calibration. We demonstrate that continued learning can recover performance from this state.

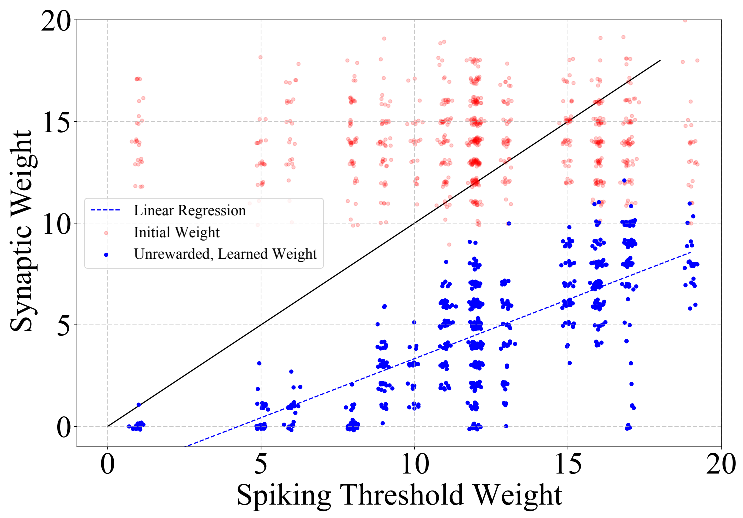


Fig. 4.11: Learning adapts weights in such a way as to compensate for variations in neuronal excitability, leading to a correlation of learned weights (corresponding to unrewarded actions, i.e., the far off-diagonal weights) and neuronal firing threshold weights. Weights are plotted with slight jitter for better visibility.

Compensation of Neuronal Excitability Fixed-pattern noise leads to variability in neuron behavior, even when the calibration is used. At a high level, this means that the spiking threshold weight (see Figure 4.5 B) is different across neurons. Each neuron in the experiment has at least 25 synapses which correspond to unrewarded actions and should therefore ideally be pushed below the spiking threshold weight. We determined the spiking threshold weight for each neuron, when using parameters as in the main experiment. This enabled us to find that those weights depicted in Figure 4.9 which correspond to unrewarded actions and the respective firing threshold weights are correlated with a Pearson's r of $r = 0.76$ (see Figure 4.11), which demonstrates the ability of the learning process to implicitly compensate differences in neuronal excitability.

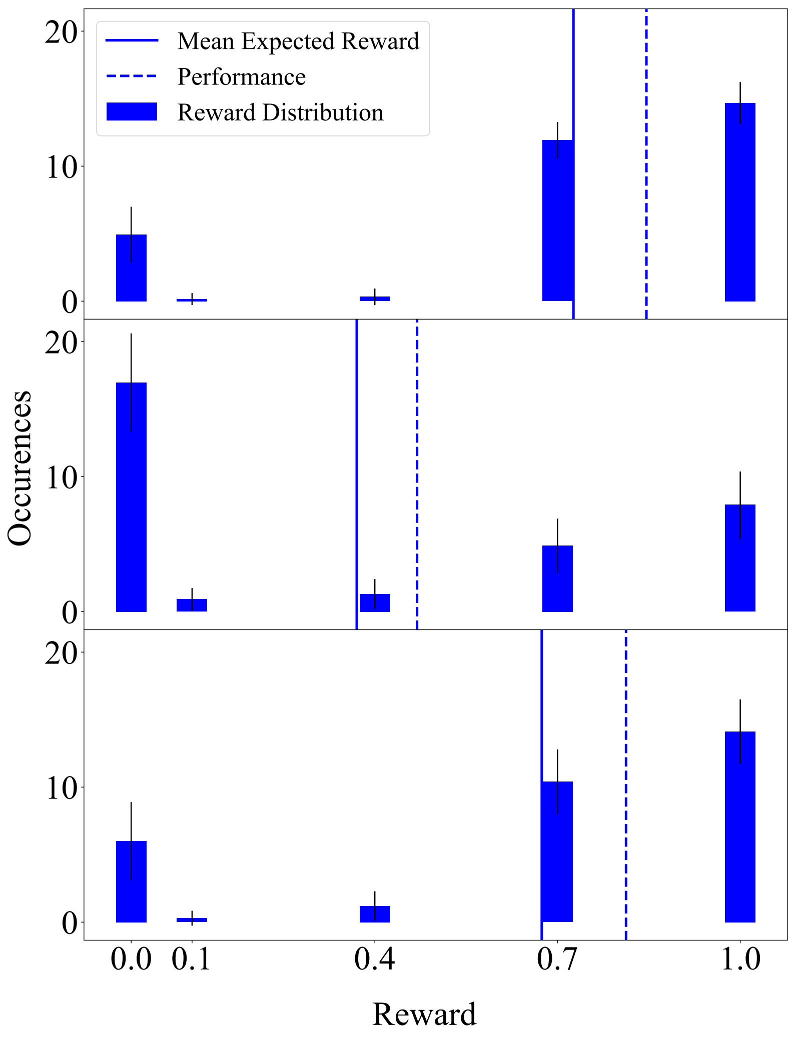


Fig. 4.12: **Top**: Baseline reward distribution for a single weight matrix obtained after 50000 experiment iterations (mean and standard deviation for each reward). **Middle**: Shuffling physical neuron assignment leads to heavy loss in reward. **Bottom**: Starting from the shuffled state, learning can recover performance.

Undoing and Recovering Learned Calibration As shown above, learning leads to a weight matrix which is adapted to the variability of the physical substrate. This point is emphasized by the following experiment: we randomly shuffle the assignment of physical neurons to logical output units, while keeping the learned logical weight matrix fixed. This leads to a state where the weight matrix is maladapted to the physical substrate and can be likened to the thought experiment of physically exchanging neurons on the chip. We can then evaluate the previously learned weight matrix, without learning, and proceed with learning to see if we can recover from this maladapted state.

We first consider a weight matrix after 50000 learning iterations, without using neuronal calibration. We then evaluate this weight matrix 100 times by determining the reward distribution after performing 1000 iterations with learning switched off, where the reward distribution refers to the set of the most recent reward received in each of the 32 states. This leads to the reward distribution shown in the top panel of Figure 4.12, with a mean expected reward and Pong performance of $\langle R \rangle = 0.73 \pm 0.05$ and $P = 0.85 \pm 0.06$, respectively.

Then, we randomly shuffle the assignment of physical and logical neurons 100 times and measure the resulting reward distribution as before, i.e., with learning switched off. This leads to the reward distribution shown in the middle panel of Figure 4.12, with a mean expected reward and Pong performance of $\langle R \rangle = 0.37 \pm 0.09$ and $P = 0.47 \pm 0.11$. The expectation of a significant drop in performance subsequent to the random shuffling is therefore confirmed.

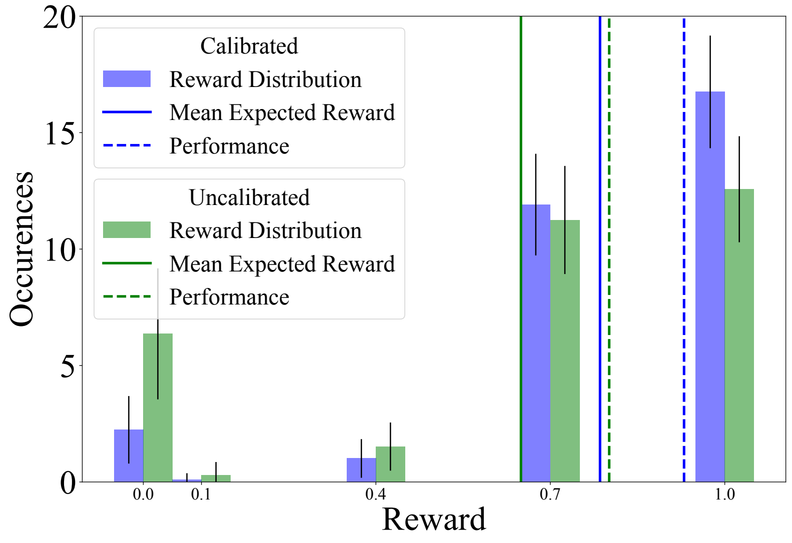
Finally, we switch learning on for each of the 100 random permutations, proceed with 50000 experiment iterations and measure the resulting reward distribution after learning. The resulting reward distribution is shown in the bottom panel, the mean expected reward and Pong performance are $\langle R \rangle = 0.67 \pm 0.07$ and $P = 0.81 \pm 0.09$, respectively. Evidently, the learning process is able to recover performance starting from the maladapted state represented by the middle plot.

These findings demonstrate that learning compensates for fixed-pattern noise and support the notion that learning itself constitutes a form of calibration.

4.2.4 Learning Is Robust

A major concern when using analog neuromorphic neurons is fixed-pattern noise and its effect on the robustness of learning processes. A commonly used method is to calibrate neurons individually toward target values, so as to compensate for variations. This approach inevitably makes a trade-off between the accuracy of the calibration data and the computational resources invested into obtaining them and has to be done for each chip individually. Therefore, it is of interest to test the impact of an absent calibration on learning performance in the experiment, which we investigate in this section.

Impact of the Calibration of Time Constants The neuronal calibration (Stradmann, 2016) adjusts the hardware parameters determining the LIF time constants (τ_{mem},



H

Fig. 4.13: Learning performance in the calibrated and uncalibrated state. We performed 100 experiments with 50000 iterations each and plot the reward distribution (mean and standard deviation) as well as mean expected reward and Pong performance.

τ_{syn}, τ_{ref}) individually per neuron in order to achieve as little variability as possible, while providing the targeted mean value. The impact of this calibration on the distribution of the empirical parameter distribution, when using the target values of the main experiment, is shown in Figure 4.6.

We define the uncalibrated state as the state where the average of the calibrated hardware parameters is used for all neurons and the calibrated state as the state where the individual values are used. Using this definition, we compare learning performance in both cases. The results are given in Figure 4.13: learning is possible in both scenarios, albeit with a drop in mean expected reward when omitting the calibration (around $17\,\%$). In the calibrated state, mean expected reward and Pong performance are $\langle \bar{R} \rangle = 0.79 \pm 0.05$ and $P = 0.93 \pm 0.05$. In the uncalibrated state, they are $\langle \bar{R} \rangle = 0.65 \pm 0.08$ and $P = 0.80 \pm 0.09$.

This shows that the experiment does profit from calibration but substantial learning success can be reached without it.

4.2.5 Learning Is Transferable

This section is concerned with the transferability of hyperparameters (neuronal and synaptic parameters) across different chips. We show that parameters can indeed be transferred, with little variation in learning performance. This implies that the used setup (hardware and learning rule) is robust enough such that the costly hyperparameter optimization has to be executed only once for several realizations of the chip.

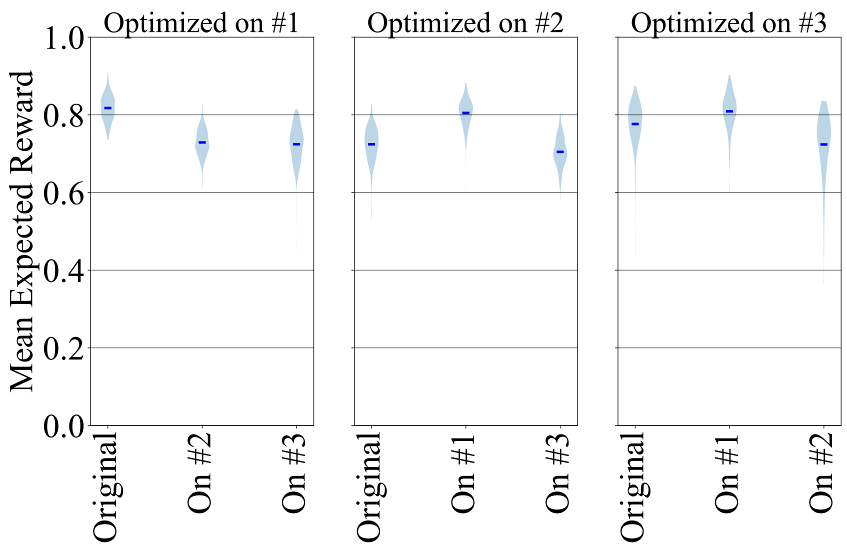


Fig. 4.14: Violin plot of mean expected reward on different chips, using different sets of parameters which were obtained using the hyperparameter optimization on different chips. These results suggest that hyperparameters can be transferred across chips.

Transferring Parameters Across Chips All of the results presented so far were obtained using chip #1 and one specific parameter set, which is given as set #1 in Table 4.1. These parameters are the result of a hyperparameter optimization procedure (see Section 4.1.4). We proceeded to perform the same optimization procedure using two further chips, chip #2 and chip #3, yielding three parameter sets in total. Then, we tested each of the three parameter sets on every chip by conducting 200 experiments in each of the nine cases. We used the individual calibration of each of the chips to transfer neuronal parameters. The results are shown in Figure 4.14: learning results are similar in all cases, slight deviations due to process variations were to be expected.

This suggests that other chips can be used as drop-in replacements for each other and that it generally suffices to perform hyperparameter optimization on a single chip.

4.2.6 Speed and Energy Efficiency

HICANN-DLSv2 operates with a speed-up factor of 10^3 compared to biological real-time and is specialized for emulating spiking neural networks (SNNs), in contrast to general-purpose processors. We compared the speed and energy efficiency of the experiment running on HICANN-DLSv2 to the software simulation running on an off-the-shelf CPU.

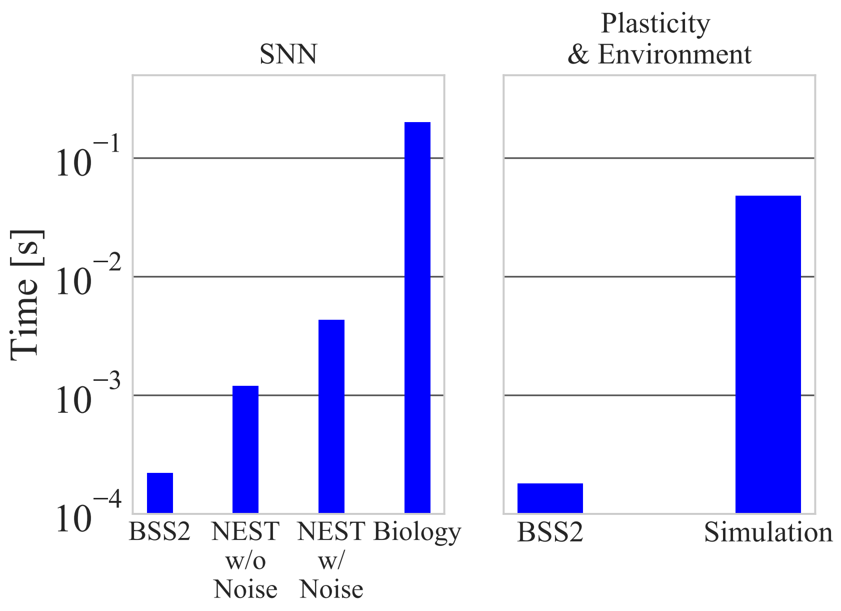


Fig. 4.15: On HICANN-DLSv2, a single experiment iteration takes around $400\,\mu s$, where network emulation takes around $220\,\mu s$ and plasticity calculations take $180\,\mu s$. In contrast, the software simulation needs $1.2\,ms$ to simulate the SNN if no noise is injected ($4.3\,ms$ if noise is injected) and around $50\,ms$ to calculate the weight changes.

Speed The software simulation (see Section 4.1.5) ran on a single core of a Intel i7-4771 CPU and used NEST v2.14.0 (Peyser et al., 2017) to simulate the SNN, while using Python to calculate weight updates and the virtual environment. Due to the small size of the network, distributing the simulation across more cores does not lead to a faster SNN simulation. In order to provide a conservative comparison, we consider the time spent in NEST's state propagating simulation routine (the *Simulate* routine) and the time spent in the Python code separately. The NEST software simulation of the $200\,ms$ of SNN activity takes $4.3\,ms$ if the NOISE_GENERATOR module is used to inject current noise into each neuron; if no noise is injected, the simulation takes $1.2\,ms$. The other calculations take around $50\,ms$. In contrast, one experiment iteration on HICANN-DLSv2 takes around $0.4\,ms$ (see Figure 4.15).

This time is approximately equally divided among neural network emulation and plasticity calculations.

An experiment with 50000 iterations takes $25\,\mathrm{s}$ on HICANN-DLSv2 and $40\,\mathrm{min}$ in the software simulation. This includes a constant overhead of $5\,\mathrm{s}$ for applying the calibration, configuration and chip setup on the neuromorphic hardware.

On HICANN-DLSv2, causal correlation traces are calculated locally at each synapse in an analog fashion. This means that adding more synapses does not incur additional computational cost – in contrast to the software simulation, where eligibility traces have to be digitally calculated using spike times. In our case, the eligibility traces were calculated manually outside of NEST, i.e., the above comparison does not even include the time required to obtain these traces.

For both HICANN-DLSv2 and the software simulation, the time required to calculate the virtual environment updates are negligible compared to plasticity calculations.

Energy Consumption We determined a lower bound of $24\,\mathrm{W}$ (no noise) and $25\,\mathrm{W}$ (with noise) for the power consumption of the SNN software simulation by measuring the current drawn by the CPU using its EPS 12V power supply cable both during NEST's numerical simulation and when idling. As the SNN simulation contained in a single iteration takes $1.2\,\mathrm{ms}$ (no noise) and $4.3\,\mathrm{ms}$ (with noise), the energy consumption for a single iteration is at least $29\,\mathrm{mJ}$ (no noise) and $106\,\mathrm{mJ}$ (with noise).

The power drawn by HICANN-DLSv2 was measured to be $57\,\mathrm{mW}$ by measuring the current drawn by the $1.2\,\mathrm{V}$ and $2.5\,\mathrm{V}$ lines supplying the chip on the prototype board, which is consistent with the measurement found in Aamir, Stradmann, et al., 2018. This does not consider the power drawn by the FPGA, as the FPGA is only used for the initial configuration and has no functional role during the experiment. Therefore, an entire iteration taking $0.4\,\mathrm{ms}$ consumes around $23\,\mu\mathrm{J}$. This implies that HICANN-DLSv2 is at least three orders of magnitude more energy-efficient than the software simulation.

We confirmed that both the speed and energy measurements do not depend on learning progress by performing measurements at the beginning of the experiment and when using a diagonal weight matrix.

4.3 Web-based Live Demo

The described experiment is highly amenable to live demonstration, as learning converges in a matter of tens of seconds of wall-clock time. Framing the learning task using the Pong arcade game provides an intuitive and visually appealing aspect. In order to provide a convenient and portable interface to demonstrate the experiment, we developed a web-based live demonstration system.

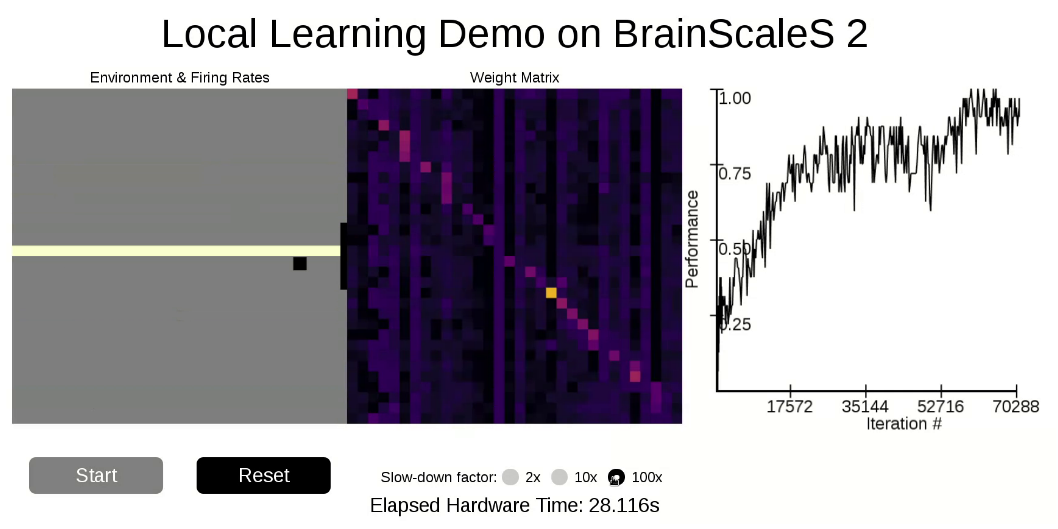

Local Learning Demo on BrainScaleS 2
Environment & Firing Rates
Weight Matrix
Performance
1.00
0.75
0.50
0.25
17572
35144
52716
70288
Iteration #
Start
Reset
Slow-down factor: 2x 10x 100x
Elapsed Hardware Time: 28.116s

Fig. 4.16: Screenshot of the web-based demo interface.

The experiment backend is written in Python using the Flask web framework and uses the web server integrated in Flask. We use the socket.io real-time engine to provide event-based communication between the web server and the browser. When an experiment is running, the experiment backend continually sends data points to all connected clients. Visualization is done via the PixiJS graphics engine and we show the firing rates, Pong playing field, weight matrix, a learning progress plot (mean expected reward or Pong performance vs. number of iterations) and the elapsed hardware time.

The graphical user interface (shown in Figure 4.16) provides the user with the possibility to start and reset the experiment, as well as to artificially slow down the experiment. The slow-down factor (2x, 10x, 100x) is implemented using a simple waiting period after each iteration. The fastest possible setting (2x) omits the waiting period, i.e., iterations are performed as possible. In this case, the speed is around 1200 iterations per second and every 200th iteration is plotted. Full real-time live streaming of the experiment is not possible, as asynchronous access of the FPGA to the PPU memory was found to be unreliable. Switching between slow-down factors is possible during the experiment, allowing the user to follow the game dynamics or to speed up learning.

An “expert menu” is accessible to the user by pressing the E key. This menu allows the user to choose the color theme, to set a technical parameter (CAPMEM_IREF) to allow for compensation of temperature variations, to choose between plotting modes (mean expected reward or Pong performance) and to toggle an automatic reset after a user-specified number of iterations.

5 Software Development

Within the scope of this thesis, the software around HICANN-DSLv2 (see Section 2.2.4) was significantly extended in order to provide access to chip functionality from the PPU, as used in the main experiment. Apart from this, we developed a debugging interface to the PPU that can be used with the GNU debugger.

The main experiment code is contained in the MODEL-HW-PONG repository maintained on the group's OpenProject project management server.

5.1 Extending PPU Functionality

In the main experiment, the PPU requires access to synapse correlation sensors, synapse weights and neuron rate counters and needs to trigger the synapse drivers to send spikes. Therefore, the PPU software library LIBNUX was extended to provide these features:

- The files CORRELATION.H and CORRELATION.C provide functions to reset or read out synapse correlation sensors of a given row into vector registers.

  Listing 5.1: Function Signatures in correlation.h:

```c
void reset_all_correlations();
void reset_correlation(uint8_t row);
void get_correlation(
    vector uint8_t* first_causal_half,
    vector uint8_t* second_causal_half,
    vector uint8_t* first_acausal_half,
    vector uint8_t* second_acausal_half,
    uint8_t row);
void get_causal_correlation(
    vector uint8_t* first_half,
    vector uint8_t* second_half,
    uint8_t row);
```

- The files COUNTER.H and COUNTER.C provide functions to reset, configure (clear on read, fire interrupt) or read out neuron rate counters.

Listing 5.2: Neuron Counter Configuration Struct:

```
typedef struct {
    bool fire_interrupt;
    bool clear_on_read;
} neuron_counter_config;
```

Listing 5.3: Function Signatures in counter.h:

```
uint32_t get_neuron_counter(uint8_t neuron);
void reset_neuron_counter(uint8_t neuron);
void reset_all_neuron_counters();
void enable_neuron_counters(uint32_t enable_mask);
uint32_t get_enabled_neuron_counters();
void configure_neuron_counter(neuron_counter_config config);
neuron_counter_config get_neuron_counter_configuration();
void clear_neuron_counters_on_read(bool value);
void fire_interrupt(bool value);
```

- The file SYN.H provides functions to set and get weights of a given synapse row from/into vector registers.

Listing 5.4: Function Signatures in syn.h:

```
void get_weights(vector uint8_t* first_half,
    vector uint8_t* second_half,
    uint8_t row);
void set_weights(vector uint8_t* first_half,
    vector uint8_t* second_half,
    uint8_t row);
```

- The files SPIKES.H and SPIKES.C provide functions to send single spike packets or a uniform spike train. Spike packets can trigger several synapse drivers at once using a 32-bit row mask and contain a 6-bit synapse label.

Listing 5.5: Neuron Counter Configuration Struct:

```
typedef struct {
    uint32_t row_mask;
    uint8_t addr;
} spike_t;
```

Listing 5.6: Function Signatures in spikes.h:

```
void send_spike(spike_t* sp);
void send_uniform_spiketrain(spike_t* single_spike,
    uint32_t number,
    uint32_t isi_usec);
```

- The files RANDOM.H and RANDOM.C provide 32-bit random numbers from an XOR-shift based pseudo-random number generator.

  Listing 5.7: Function Signatures in random.h:

```c
uint32_t xorshift32(uint32_t* seed);
```

- The files TIME.H and TIME.C provide a function to sleep a given number of clock cycles.

  Listing 5.8: Function Signatures in time.h:

```c
void sleep_cycles(uint32_t cycles);
```

5.2 GNU Debugger Interface

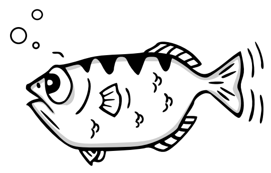


Fig. 5.1: The GNU debugger mascot.

In software development, debuggers are a vital tool to investigate program behavior at runtime: the regular program flow can be interrupted using break points and as soon as a break point is hit, the program control is handed over to the debugger. The developer can then examine and manipulate the contents of the memory and processor registers, look at the stack trace as well as step through the code. This is a critical component of the software development process, especially when dealing with fully custom hardware such as the PPU.

We developed an interface to the GNU debugger (Free Software Foundation, 2018a, GDB) that allows the user to debug PPU programs. The PPU represents a remote, embedded system. GDB contains a remote serial protocol that can be used to debug remote systems (Free Software Foundation, 2018b). In this case, GDB connects to a server that implements this protocol in order to send user input and receive state information. When debugging an embedded system without an operating system, the program must be linked together with a so-called *debugging stub* that includes an interrupt handler (called when interrupts of a specific type occur) which performs the operations required for debugging (i.e., which processes the user-specified commands).

Therefore, interfacing the PPU to GBD requires two components:

1. The debugging stub that is linked together with PPU programs and

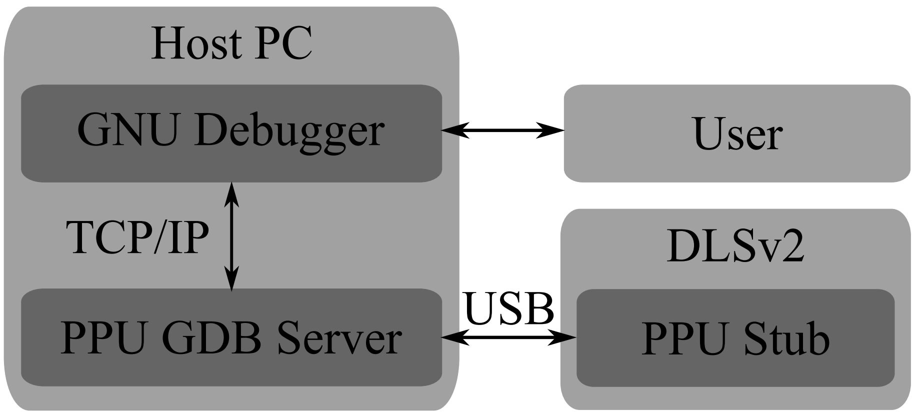


Fig. 5.2: Communication diagram of the GDB PPU debugger. The newly developed components are the PPU GDB server and the PPU stub.

2. a server on the host computer that mediates communication between the interrupt handler on the PPU and GDB (see Figure 5.2).

The first component was developed by the author of this thesis while the second component was developed by Philipp Spilger who also documented the server as well as the debugging work flow (Philipp Spilger, 2018). Therefore, we will focus on the debugging stub in the following.

PPU Debugging Stub We describe the header-only PPU debugging stub that is part of Changeset No. 3955 (internal code review). After including it into a program, the user can simply call the `breakpoint` method to set breakpoints. This method executes the trap instruction (an instruction causing a software-based interrupt), with a nop (no operation) before so as to work around a hardware bug where the instruction after trap is executed otherwise:

Listing 5.9: Breakpoint method in ppu-stub.h

```
static inline __attribute__((always_inline)) void breakpoint(void)
{
    // clang-format off
    asm volatile (
            "nop\n" // needed, because otherwise, the instruction
               after trap is executed, when trap.
            "trap"
        ); // Always trap
    // clang-format on
}
```

The trap execution will cause the PPU to save the Program Counter (PC) and the Machine State Register (MSR) to the Status Restore Registers (SRRs) 0 and 1, respectively and then jump into the program interrupt handler:

Listing 5.10: Program interrupt handler in ppu-stub.h

```c
void __attribute__((naked)) isr_program(void)
{
    save_registers();
    handle_exception();
    load_registers();
    asm volatile ("rfi"); // Return From Interrupt
}
```

The `naked` attribute was introduced to gcc in changeset 3959 (internal code review) and simply prevents the compiler from creating a function prologue and epilogue which would normally serve to save and restore registers. In our case, we require all registers to be exactly in the state they were upon trapping and therefore use this attribute to prevent compiler generated instructions from tampering with register contents. The `save_registers` function saves all General and Special Purpose Registers (GPRs/SPRs) to memory in order to expose the contents to the user.

After saving the registers to memory, the interrupt handler calls `handle_exception`. This is the main part of the stub and communicates with the PPU GDB server using two memory buffers for input and output. Different commands from the server can trigger the stub to

- continue execution, either at the instruction after the trap or at an arbitrary location,
- to get all or single registers (SPRs, GPRs, VRs),
- to set single registers (SPRs, GPRs, VRs),
- to step through the code and
- to flush the instruction cache for newly set memory.

The main memory can be read and set by the PPU GDB server directly and in principle this does not require assistance from the PPU stub. However, a notable workaround concerns the instruction cache of 4 KiB. By specification, the PPU instruction cache should be invalidated if the corresponding location in memory is altered (Friedmann, 2013). However, this was found not to be the case. Additionally, the data cache block invalidate instruction that is part of the PowerPC ISA v2.06 is not available on the PPU. Therefore, in order to flush the instruction cache corresponding to a given memory location, the stub replaces the instruction at the given location plus 4 kB by a branch instruction that jumps right back into the stub. The stub then branches to the modified instruction, which immediately returns to the stub that then restores the proper memory content. In this way, the direct-mapped instruction cache is forced to reload the instruction at the given location. This workaround is critical when stepping through the code and when setting break points using the PPU GDB server as instructions have to be replaced at runtime.

The fact that instructions have to be replaced at runtime can itself be seen as a workaround for the absence of a hardware break point register, which would normally enable the user to load addresses which trigger an interrupt. In order to allow for convenient debugging, the following hardware generations should include a properly invalidated instruction cache and ideally, also hardware breakpoints.

The stub restores all SPRs and GPRs each time it is exited (i.e., when stepping or continuing) in `load_registers` and executes the return from interrupt (rfi) instruction which restores the PC from SRR0 and MSR from SRR1. Note that the described debugger is useful to debug PPU programs. Entire experiments, which generally include a playback program on the FPGA that interacts with the chip, cannot be debugged because the FPGA is required for the PPU GDB server in our case. Such an integration could be the subject of future work. Another currently present limitation is that branch instructions of the debugged program cannot be stepped over. This would require detecting (and possibly predicting) branches, determining the branched-to address and replacing the instruction at that address with a trap instruction. Implementing this functionality will be subject of future work as well.

Discussion

6

This work demonstrates that HICANN-DLSv2 is a functional neuromorphic substrate by concertedly using all components of the chip, short of anti-causal correlation traces and inhibitory synapses, in a reinforcement learning experiment. We thereby show the advantages of the neuromorphic approach that HICANN-DLSv2 represents — specifically in terms of speed, energy efficiency, flexible synaptic plasticity and the robustness of learning.

Speed and Energy Efficiency of HICANN-DLSv2 We determined HICANN-DLSv2 to be at least an order of magnitude faster and three orders of magnitude more energy-efficient than an equivalent software simulation with the standard NEST v2.14.0 simulator (see Section 4.2.6). These are conservative comparisons by construction, as we compare complete iterations on HICANN-DLSv2 to only the numerical SNN simulating part of the software simulation (native in NEST), which includes neither the calculation of causal correlation traces nor the calculation and application of weight changes which is done in our custom Python script. Due to the small size of the simulated network, the software simulation operates around 100 times faster than real-time. This is in contrast to the case of large SNNs, where software simulations and even specialized digital-neuromorphic systems generally achieve only real-time operation (Jordan et al., 2018; Albada et al., 2018; Mikaitis et al., 2018). The neuromorphic approach behind HICANN-DLSv2, however, affords a 1000-fold speed-up factor that is independent of network size. Eligibility traces are calculated in an analog fashion and locally at each synapse, which means that scaling up network size does not incur additional computational cost with regards to obtaining correlation values. Therefore, we expect its advantages in terms of speed and energy efficiency to become even more apparent when moving to larger networks.

Learning Robustness We found that the emulated model does not require calibration of neuronal time constants to learn (see Section 4.2.2). The mean expected reward drops from $\langle \bar{R} \rangle = 0.79 \pm 0.05$ to $\langle \bar{R} \rangle = 0.65 \pm 0.08$ in the uncalibrated state. This drop in learning performance comes as no surprise, as an initially balanced neuronal activity is desirable in order to provide even chances for action exploration and the absence of the calibration has a strong influence on the distribution of neuronal time constants (see Section 4.2.1). It is remarkable that no compensation of neuronal fixed-pattern noise is necessary to provide a good learning result when individual neuron activity is the main driver of learning. Future experiments could further alleviate the need for a calibration by using a population coding scheme, where neuronal variability could average out, as well as by using more systematic action exploration mechanisms that do not rely on the intrinsic noise of individual neurons.

Noise as Computational Resource Action exploration and therefore learning is driven by the trial-to-trial variability of neuronal firing rates (i.e., firing rates vary from trial to trial when keeping the uniform input spike train fixed, see Section 2.2.1). The propensity to this neuronal behavior likely stems from the fact that comparatively long membrane time constants are used where the LIF time constants are a supra-linear function of the hardware parameters, which amplifies parameter deviations (Stradmann, 2016). For the given Inter-Spike Interval (ISI) of the input spike train (see Table 4.1), long membrane time constants make sense to be able to integrate incoming PSPs. The trial-to-trial variability is generally not desirable and chips are designed with the aim of keeping such effects to a minimum. Still, it is important to note that the presented model can not only deal with this type of noise gracefully, indeed, it benefits from it by utilizing it for action exploration. Solving more complicated learning tasks will likely require more systematic mechanisms for action exploration. The action exploration mechanism used in this experiment is prone to be disturbed by unbalanced neuronal activity. It is precisely for this reason that some states are not perfectly learned (i.e., $\bar{R} < 1$) as in those cases the correct neuron is drowned out by the incorrect neurons: due to an imbalance in neuronal excitability the exploration mechanism can only slowly move away (if at all) from the suboptimal solution.

Transferability By performing the hyperparameter optimization procedure and testing its results among three different chips, we were able to verify that the obtained hyperparameters can be transferred across chips and that learning results can be reproduced (see Section 4.2.5). However, we found that chip #1 has a slight edge and performs better compared to chips #2 and #3 in all cases. This is due to process variations which led to the capacitive memory functioning better on the first chip. Being able to transfer hyperparameters across chips is an important point, as optimizing hyperparameters (also referred to as "learning to learn", Bellec et al., 2018) can be a computationally costly and time consuming procedure. Further, we conclude that chips can be drop-in replacements for each other.

Limitations of the study In this study, we used a prototype chip that is down-scaled with respect to the full future chips that will be part of BSS2. The limited number of neurons constrains the complexity of neural network models that can be implemented on HICANN-DLSv2. However, the results in terms of speed and energy efficiency will be carried over to a wafer-scale implementation. Because neurons and synapses are emulated in dedicated circuits the emulation speed is independent of the size of the network. As each chip will have its own PPUs in a wafer-scale setup, the plasticity calculation will not limit the emulation. For the same reason, the power consumption scales linearly with the number of used chips, i.e. with the size of the emulated network. The presented model contains a shallow two-layer neural network — in the future, deeper networks will allow for more complex input transformations. For example, reward-based Hebbian learning in recurrent neural networks has been demonstrated (Miconi, 2017; Hoerzer et al., 2014; Legenstein, Chase, et al., 2010) and could be implemented in future experiments. The simplicity of the network model is also a merit because it allowed us to focus on the neuromorphic hardware aspects of the experiment and to analyze the results without extra complications from the model. The virtual environment used in this experiment is computationally extremely simple and was therefore simulated using the embedded processor. Future experiments might require complex simulations; in this case, it would be appropriate to perform the simulation externally — for example on FPGAs, on a microcontroller or on the host — and provide reinforcement signals to the PPU. Note that the PPU is not designed as a virtual environment simulator but as a platform to implement flexible plasticity rules using the available observables.

Conclusion and Outlook We have demonstrated that HICANN-DLSv2 is a functional neuromorphic substrate and delivers on its promises of accelerated learning and power-efficient SNN emulation. This thesis lays the groundwork for future reinforcement learning experiments which will tackle more complex learning tasks using larger and more structured neural networks. The neural network models that can be realized on HICANN-DLSv2 are constrained in size. However, the next generation neuromorphic chip contains 512 neurons with 256 synapses each, enabling the implementation of larger networks [1]. It has two embedded processor with appropriately scaled vector units, a more flexible neuron model based on the multi-compartment Adaptive-Exponential (AdEx) Integrate-and-Fire model and synapses with features for Short-Term Plasticity (STP). Additionally, Poisson-like input stimuli can be realized using on-chip Pseudo-Random Number Generators (PRNGs), allowing for controllable noise which can be used for LIF-based neural sampling (Mihai A. Petrovici et al., 2016) or to implement an action exploration mechanism. On a larger time scale, future chips will be integrated on wafer-scale similar to the BrainScaleS 1 system (Schemmel, Brüderle, et al., 2010) which will enable the implementation of even larger networks.

[1]The next generation chip was in its design phase during the writing of this thesis.

The wafer-scale BSS2 system will allow the implementation of large, deep and biologically inspired neural networks that can interact with complex environments. Because of the acceleration factor, researchers will be able to evaluate learning processes that take years of biological time in a matter of hours. With this study, we have paved the way for biologically inspired artificial intelligence on a novel computational substrate that is a physical model of the human brain.

Bibliography

Aamir, Syed Ahmed, Paul Müller, Andreas Hartel, Johannes Schemmel, and Karlheinz Meier (2016). „A highly tunable 65-nm CMOS LIF neuron for a large scale neuromorphic system“. In: *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*. IEEE, pp. 71–74. DOI: 10.1109/ESSCIRC.2016.7598245. URL: http://ieeexplore.ieee.org/document/7598245/ (cit. on pp. 8, 11).

Aamir, Syed Ahmed, Yannik Stradmann, Paul Müller, et al. (2018). „An Accelerated LIF Neuronal Network Array for a Large-Scale Mixed-Signal Neuromorphic Architecture“. In: *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14. DOI: 10.1109/TCSI.2018.2840718. URL: https://ieeexplore.ieee.org/document/8398542/ (cit. on pp. 8, 11, 56).

Albada, Sacha J. van, Andrew G. Rowley, Johanna Senk, et al. (2018). „Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model“. In: *Frontiers in Neuroscience* 12, p. 291. DOI: 10.3389/fnins.2018.00291. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00291/full (cit. on pp. 2, 65).

Baras, Dorit and Ron Meir (2007). „Reinforcement Learning, Spike-Time-Dependent Plasticity, and the BCM Rule“. In: *Neural Computation* 19.8, pp. 2245–2279. DOI: 10.1162/neco.2007.19.8.2245. URL: https://doi.org/10.1162/neco.2007.19.8.2245 (cit. on p. 27).

Bayer, Hannah M. and Paul W. Glimcher (2005). „Midbrain Dopamine Neurons Encode a Quantitative Reward Prediction Error Signal“. In: *Neuron* 47.1, pp. 129–141. DOI: 10.1016/J.NEURON.2005.05.020. URL: https://www.sciencedirect.com/science/article/pii/S0896627305004678 (cit. on p. 26).

Bellec, Guillaume, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass (2018). „Long short-term memory and learning-to-learn in networks of spiking neurons“. In: URL: http://arxiv.org/abs/1803.09574 (cit. on p. 66).

Benjamin, Ben Varkey, Peiran Gao, Emmett McQuinn, et al. (2014). „Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations“. In: *Proceedings of the IEEE* 102.5, pp. 699–716. DOI: 10.1109/JPROC.2014.2313565. URL: http://ieeexplore.ieee.org/document/6805187/ (cit. on pp. 3, 8).

Bentham, Jeremy (1780). *An Introduction to the Principles of Morals and Legislation*. Vol. 45. n/a. Dover Publications (cit. on p. 17).

Bi, Guo-qiang and Mu-ming Poo (1998). „Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type“. In: *The Journal of Neuroscience* 18.24, p. 10464. DOI: `10.1523/JNEUROSCI.18-24-10464.1998`. URL: `http://www.jneurosci.org/content/18/24/10464.abstract` (cit. on p. 9).

Bliss, T V P and G L Collingridge (1993). „A synaptic model of memory: long-term potentiation in the hippocampus“. In: *Nature* 361, p. 31. URL: `https://doi.org/10.1038/361031a0%2010.1038/361031a0` (cit. on p. 9).

Brzosko, Zuzanna, Wolfram Schultz, and Ole Paulsen (2015). „Retroactive modulation of spike timing-dependent plasticity by dopamine“. In: *eLife* 4, e09685. DOI: `10.7554/eLife.09685`. URL: `https://elifesciences.org/articles/09685` (cit. on p. 26).

Chris73 (2015). *Figure of an action potential*. URL: `https://commons.wikimedia.org/wiki/File:Action_potential_basic_shape.svg` (cit. on p. 4).

Davies, M, N Srinivasa, T Lin, et al. (2018). „Loihi: A Neuromorphic Manycore Processor with On-Chip Learning“. In: *IEEE Micro* 38.1, pp. 82–99. DOI: `10.1109/MM.2018.112130359` (cit. on p. 3).

Dean, J, D Patterson, and C Young (2018). „A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution“. In: *IEEE Micro* 38.2, pp. 21–29. DOI: `10.1109/MM.2018.112130030` (cit. on p. 1).

Douence, Vincent, Arnaud Laflaquiere, Sylvie Le Masson, Thierry Bal, and Gwendal Le Masson (1999). „Analog electronic system for simulating biological neurons“. In: *Engineering Applications of Bio-Inspired Artificial Neural Networks*. Ed. by Jose Mira and Juan V Sanchez-Andres. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 188–197 (cit. on p. 8).

Dunnett, S.B., M. Bentivoglio, A. Björklund, and T. Hökfelt (2004). *Dopamine*. Elsevier Science, p. 418 (cit. on p. 25).

Edelmann, Elke and Volkmar Lessmann (2011). „Dopamine Modulates Spike Timing-Dependent Plasticity and Action Potential Properties in CA1 Pyramidal Neurons of Acute Rat Hippocampal Slices“. In: *Frontiers in Synaptic Neuroscience* 3, p. 6. DOI: `10.3389/fnsyn.2011.00006`. URL: `http://journal.frontiersin.org/article/10.3389/fnsyn.2011.00006/abstract` (cit. on p. 26).

Electronic Vision(s) (2017). *Electronic Vision(s) GCC*. URL: `https://github.com/electronicvisions/gcc` (cit. on p. 15).

Farries, Michael A. and Adrienne L. Fairhall (2007). „Reinforcement Learning With Modulated Spike Timing–Dependent Synaptic Plasticity“. In: *Journal of Neurophysiology* 98.6, pp. 3648–3665. DOI: `10.1152/jn.00364.2007`. URL: `http://www.physiology.org/doi/10.1152/jn.00364.2007` (cit. on p. 30).

Fetz, E E and M A Baker (1973). „Operantly conditioned patterns on precentral unit activity and correlated responses in adjacent cells and contralateral muscles.“ In: *Journal of Neurophysiology* 36.2, pp. 179–204. DOI: `10.1152/jn.1973.36.2.179`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/4196269%20http://www.physiology.org/doi/10.1152/jn.1973.36.2.179` (cit. on p. 24).

Feuillet, Lionel, Henry Dufour, and Jean Pelletier (2007). „Brain of a white-collar worker.“ In: *Lancet (London, England)* 370.9583, p. 262. DOI: 10.1016/S0140-6736(07)61127-1. URL: http://www.ncbi.nlm.nih.gov/pubmed/17658396 (cit. on p. 1).

Fiorillo, Christopher D, Philippe N Tobler, and Wolfram Schultz (2003). „Discrete Coding of Reward Probability and Uncertainty by Dopamine Neurons“. In: *Science* 299.5614, p. 1898. URL: http://science.sciencemag.org/content/299/5614/1898.abstract (cit. on p. 26).

– (2005). „Evidence that the delay-period activity of dopamine neurons corresponds to reward uncertainty rather than backpropagating TD errors“. In: *Behavioral and brain functions : BBF* 1, p. 7. DOI: 10.1186/1744-9081-1-7. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1182345/ (cit. on p. 26).

Florian, Răzvan V (2007). „Reinforcement Learning Through Modulation of Spike-Timing-Dependent Synaptic Plasticity“. In: *Neural Computation* 19.6, pp. 1468–1502. DOI: 10.1162/neco.2007.19.6.1468. URL: https://doi.org/10.1162/neco.2007.19.6.1468 (cit. on pp. 27, 30).

Free Software Foundation (2018a). *Debugging with GDB: the GNU Source-Level Debugger*. URL: https://sourceware.org/gdb/onlinedocs/gdb/ (cit. on p. 61).

– (2018b). *GDB Remote Serial Protocol*. URL: https://sourceware.org/gdb/onlinedocs/gdb/Remote-Protocol.html#Remote-Protocol (cit. on p. 61).

Fremaux, N., H. Sprekeler, and W. Gerstner (2010). „Functional Requirements for Reward-Modulated Spike-Timing-Dependent Plasticity“. In: *Journal of Neuroscience* 30.40, pp. 13326–13337. DOI: 10.1523/JNEUROSCI.6249-09.2010. URL: http://www.ncbi.nlm.nih.gov/pubmed/20926659%20http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.6249-09.2010 (cit. on pp. 27, 28, 30, 31, 36).

Frémaux, Nicolas and Wulfram Gerstner (2015). „Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules.“ In: *Frontiers in neural circuits* 9, p. 85. DOI: 10.3389/fncir.2015.00085. URL: http://www.ncbi.nlm.nih.gov/pubmed/26834568%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4717313 (cit. on pp. 27, 29).

Friedmann, Simon (2013). „A New Approach to Learning in Neuromorphic Hardware“. PhD thesis. University of Heidelberg. DOI: 10.11588/heidok.00015359 (cit. on p. 63).

Friedmann, Simon, Johannes Schemmel, Andreas Grübl, et al. (2017). „Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System“. In: *IEEE Transactions on Biomedical Circuits and Systems* 11.1, pp. 128–142. DOI: 10.1109/TBCAS.2016.2579164. URL: http://ieeexplore.ieee.org/document/7563782/ (cit. on pp. 10, 12–14).

Furber, Steve B., Francesco Galluppi, Steve Temple, and Luis A. Plana (2014). „The SpiNNaker Project“. In: *Proceedings of the IEEE* 102.5, pp. 652–665. DOI: 10.1109/JPROC.2014.2304638. URL: http://ieeexplore.ieee.org/document/6750072/ (cit. on p. 3).

Gerstner, Wulfram, Werner M Kistler, Richard Naud, and Liam Paninski (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press. DOI: DOI:10.1017/CBO9781107447615. URL: https://www.cambridge.org/core/books/neuronal-dynamics/75375090046733765596191E23B2959D (cit. on pp. 3–6, 27).

Gerstner, Wulfram, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea (2018). „Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules“. In: *Frontiers in Neural Circuits* 12, p. 53. URL: https://www.frontiersin.org/article/10.3389/fncir.2018.00053 (cit. on p. 10).

Greensmith, Evan, Peter L. Bartlett, and Jonathan Baxter (2004). „Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning“. In: *Journal of Machine Learning Research* 5.Nov, pp. 1471–1530. URL: http://www.jmlr.org/papers/v5/greensmith04a.html (cit. on pp. 23, 28).

Guttman, Norman (1953). „Operant conditioning, extinction, and periodic reinforcement in relation to concentration of sucrose used as reinforcing agent.“ In: *Journal of Experimental Psychology* 46.4, pp. 213–224. DOI: 10.1037/h0061893. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0061893 (cit. on p. 24).

Häfliger, Philipp, Misha Mahowald, and Lloyd Watts (1997). „A Spike Based Learning Neuron in Analog VLSI“. In: *Advances in Neural Information Processing Systems 9*. Ed. by M C Mozer, M I Jordan, and T Petsche. MIT Press, pp. 692–698. URL: http://papers.nips.cc/paper/1322-a-spike-based-learning-neuron-in-analog-vlsi.pdf (cit. on p. 8).

Hart, Andrew S, Jeremy J Clark, and Paul E M Phillips (2015). „Dynamic shaping of dopamine signals during probabilistic Pavlovian conditioning“. In: *Neurobiology of learning and memory* 117, pp. 84–92. DOI: 10.1016/j.nlm.2014.07.010. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4293327/ (cit. on p. 26).

Head, Tim, MechCoder, Gilles Louppe, et al. (2018). „scikit-optimize/scikit-optimize: v0.5.2“. In: DOI: 10.5281/ZENODO.1207017. URL: https://zenodo.org/record/1207017 (cit. on p. 39).

Hebb, Donald (1949). *The Organization of Behavior*. New York: Wiley & Sons (cit. on p. 9).

Hock, Matthias (2014). *Modern Semiconductor Technologies for Neuromorphic Hardware*. eng. URL: http://archiv.ub.uni-heidelberg.de/volltextserver/17129/1/Dissertation_Matthias_Hock.pdf (cit. on p. 11).

Hodgkin, A L and A F Huxley (1952). „A quantitative description of membrane current and its application to conduction and excitation in nerve“. In: *The Journal of physiology* 117.4, pp. 500–544. URL: https://www.ncbi.nlm.nih.gov/pubmed/12991237%20https://www.ncbi.nlm.nih.gov/pubmed/12991237 (cit. on p. 5).

Hoerzer, Gregor M., Robert Legenstein, and Wolfgang Maass (2014). „Emergence of Complex Computational Structures From Chaotic Neural Networks Through Reward-Modulated Hebbian Learning“. In: *Cerebral Cortex* 24.3, pp. 677–690. DOI: 10.1093/cercor/bhs348. URL: http://www.ncbi.nlm.nih.gov/pubmed/23146969%20https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhs348 (cit. on p. 67).

Hollerman, Jeffrey R. and Wolfram Schultz (1998). „Dopamine neurons report an error in the temporal prediction of reward during learning“. In: *Nature Neuroscience* 1.4, pp. 304–309. DOI: 10.1038/1124. URL: http://www.nature.com/articles/nn0898_304 (cit. on p. 26).

Indiveri, Giacomo, Bernabé Linares-Barranco, Tara Julia Hamilton, et al. (2011). „Neuromorphic Silicon Neuron Circuits“. In: *Frontiers in Neuroscience* 5, p. 73. DOI: 10.3389/fnins.2011.00073. URL: http://journal.frontiersin.org/article/10.3389/fnins.2011.00073/abstract (cit. on pp. 3, 8).

Izhikevich, E. M. (2007). „Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling“. In: *Cerebral Cortex* 17.10, pp. 2443–2452. DOI: 10.1093/cercor/bhl152. URL: http://www.ncbi.nlm.nih.gov/pubmed/17220510%20https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhl152 (cit. on p. 30).

Jordan, Jakob, Tammo Ippen, Moritz Helias, et al. (2018). „Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers“. In: *Frontiers in Neuroinformatics* 12, p. 2. DOI: 10.3389/fninf.2018.00002. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00002/full (cit. on pp. 2, 65).

Krishnavedala (2012). *Schematic of the Hodgkin-Huxley model*. URL: https://commons.wikimedia.org/wiki/File:Hodgkin-Huxley.svg (cit. on p. 5).

Legenstein, Robert, Steven M Chase, Andrew B Schwartz, and Wolfgang Maass (2010). „A Reward-Modulated Hebbian Learning Rule Can Explain Experimentally Observed Network Reorganization in a Brain Control Task“. In: *The Journal of Neuroscience* 30.25, p. 8400. DOI: 10.1523/JNEUROSCI.4284-09.2010. URL: http://www.jneurosci.org/content/30/25/8400.abstract (cit. on p. 67).

Legenstein, Robert, Dejan Pecevski, and Wolfgang Maass (2008). „A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback“. In: *PLoS Computational Biology* 4.10. Ed. by Lyle J. Graham, e1000180. DOI: 10.1371/journal.pcbi.1000180. URL: http://dx.plos.org/10.1371/journal.pcbi.1000180 (cit. on p. 30).

Lowel, S and W Singer (1992). „Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity“. In: *Science* 255.5041, p. 209. DOI: 10.1126/science.1372754. URL: http://science.sciencemag.org/content/255/5041/209.abstract (cit. on p. 9).

Markram, Henry, Wulfram Gerstner, and Per Jesper Sjöström (2011). „A History of Spike-Timing-Dependent Plasticity“. In: *Frontiers in Synaptic Neuroscience* 3, p. 4. URL: https://www.frontiersin.org/article/10.3389/fnsyn.2011.00004 (cit. on p. 10).

Mead, Carver (1989). *Analog VLSI and Neural Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (cit. on pp. 2, 3).

Merolla, Paul A, John V Arthur, Rodrigo Alvarez-Icaza, et al. (2014). „A million spiking-neuron integrated circuit with a scalable communication network and interface“. In: *Science* 345.6197, p. 668. URL: http://science.sciencemag.org/content/345/6197/668.abstract (cit. on p. 3).

Miconi, Thomas (2017). „Biologically plausible learning in recurrent neural networks reproduces neural dynamics observed during cognitive tasks“. In: *eLife* 6. DOI: 10.7554/eLife.20899. URL: https://elifesciences.org/articles/20899 (cit. on p. 67).

Mikaitis, Mantas, Garibaldi Pineda García, James C Knight, and Steve B Furber (2018). „Neuromodulated Synaptic Plasticity on the SpiNNaker Neuromorphic System“. In: *Frontiers in Neuroscience* 12, p. 105. DOI: 10.3389/fnins.2018.00105. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00105 (cit. on pp. 2, 65).

Montague, P Read, Steven E Hyman, and Jonathan D Cohen (2004). „Computational roles for dopamine in behavioural control“. In: *Nature* 431, p. 760. URL: `http://dx.doi.org/10.1038/nature03015%2010.1038/nature03015` (cit. on p. 25).

Moritz, Chet T and Eberhard E Fetz (2011). „Volitional control of single cortical neurons in a brain-machine interface.“ In: *Journal of neural engineering* 8.2, p. 025017. DOI: `10.1088/1741-2560/8/2/025017`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/21436531%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3156089` (cit. on p. 24).

Mozafari, M, S R Kheradpisheh, T Masquelier, A Nowzari-Dalini, and M Ganjtabesh (2018). „First-Spike-Based Visual Categorization Using Reward-Modulated STDP“. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13. DOI: `10.1109/TNNLS.2018.2826721` (cit. on p. 30).

NIDA and Quasihuman (2012). *Figure of dopamine pathways*. URL: `https://commons.wikimedia.org/wiki/File:Dopamine_pathways.svg` (cit. on p. 24).

Open Clipart (2013). *Clipart of a red slot machine*. URL: `https://openclipart.org/detail/185039/little-red-slot-machine` (cit. on p. 20).

Pavlov, P Ivan (2010). „Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex“. In: *Annals of neurosciences* 17.3, pp. 136–141. DOI: `10.5214/ans.0972-7531.1017309`. URL: `https://www.ncbi.nlm.nih.gov/pubmed/25205891%20https://www.ncbi.nlm.nih.gov/pmc/PMC4116985/` (cit. on p. 25).

Pawlak, Verena and Jason N D Kerr (2008). „Cellular/Molecular Dopamine Receptor Activation Is Required for Corticostriatal Spike-Timing-Dependent Plasticity“. In: *The Journal of Neuroscience*. DOI: `10.1523/JNEUROSCI.4402-07.2008`. URL: `https://pdfs.semanticscholar.org/f072/62a58aa607abfeff0a040d98ad561342c5d1.pdf` (cit. on p. 26).

Pawlak, Verena, Jeffery Wickens, Alfredo Kirkwood, and Jason Kerr (2010). „Timing is not Everything: Neuromodulation Opens the STDP Gate“. In: *Frontiers in Synaptic Neuroscience* 2, p. 146. URL: `https://www.frontiersin.org/article/10.3389/fnsyn.2010.00146` (cit. on p. 10).

Petrovici, Mihai A., Johannes Bill, Ilja Bytschok, Johannes Schemmel, and Karlheinz Meier (2016). „Stochastic inference with spiking neurons in the high-conductance state“. In: *Physical Review E* 94.4, p. 042312. DOI: `10.1103/PhysRevE.94.042312`. URL: `https://link.aps.org/doi/10.1103/PhysRevE.94.042312` (cit. on p. 67).

Petrovici, Mihai Alexandru (2016). *Form Versus Function: Theory and Models for Neuronal Substrates*. Springer Theses. Cham: Springer International Publishing. DOI: `10.1007/978-3-319-39552-4`. URL: `http://link.springer.com/10.1007/978-3-319-39552-4` (cit. on p. 7).

Peyser, Alexander, Ankur Sinha, Stine Brekke Vennemo, et al. (2017). „NEST 2.14.0“. In: *Zenodo*. DOI: `10.5281/ZENODO.882971`. URL: `https://zenodo.org/record/882971` (cit. on pp. 42, 55).

Pfister, Jean-Pascal, Taro Toyoizumi, David Barber, and Wulfram Gerstner (2006). „Optimal Spike-Timing-Dependent Plasticity for Precise Action Potential Firing in Supervised Learning“. In: *Neural Computation* 18.6, pp. 1318–1348. DOI: `10.1162/neco.2006.18.6.1318`. URL: `https://doi.org/10.1162/neco.2006.18.6.1318` (cit. on pp. 27, 28).

Philipp Spilger (2018). *On parameterization and debugging of PPU programs*. URL: https://www.kip.uni-heidelberg.de/vision/publications/reports/report_pspilger.pdf (cit. on p. 62).

Quasar, Jarosz (2009). *Figure of a neuron*. URL: https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg (cit. on p. 4).

Schemmel, J, D Brüderle, A Grübl, et al. (2010). „A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling“. In: *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS"10)*, pp. 1947–1950 (cit. on pp. 3, 10, 67).

Schemmel, J, K Meier, and E Mueller (2004). „A new VLSI model of neural microcircuits including spike time dependent plasticity“. In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. Vol. 3, pp. 1711–1716. DOI: 10.1109/IJCNN.2004.1380861 (cit. on p. 8).

Schultz, W, P Dayan, and P R Montague (1997). „A neural substrate of prediction and reward.“ In: *Science (New York, N.Y.)* 275.5306, pp. 1593–9. DOI: 10.1126/SCIENCE.275.5306.1593. URL: http://www.ncbi.nlm.nih.gov/pubmed/9054347 (cit. on p. 26).

Sjöström, J and W Gerstner (2010). „Spike-timing dependent plasticity“. In: *Scholarpedia* 5, p. 1362. URL: http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity (cit. on p. 9).

Sonic Homepage (2019). URL: http://www.sonici.com/ (cit. on p. 3).

Stallman, Richard M. and GCC Developer Community (2018). *GCC 8.0 GNU Compiler Collection Internals*. 12th Media Services (cit. on p. 15).

Stradmann, Yannik (2016). *Characterization and Calibration of a Mixed-Signal Leaky Integrate and Fire Neuron on HICANN-DLS*. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/download.php/5758/temp/3371-1.pdf (cit. on pp. 11–13, 44, 48, 52, 66).

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning : an introduction*. MIT Press, p. 322. URL: https://mitpress.mit.edu/books/reinforcement-learning (cit. on pp. 17–19, 21, 23).

Synaptics Homepage (2019). URL: https://www.synaptics.com/ (cit. on p. 3).

Thorndike, Edward Lee (1911). *Animal intelligence: Experimental studies*. New York: Macmillan Co. (cit. on pp. 17, 24).

Urbanczik, Robert and Walter Senn (2009). „Reinforcement learning in populations of spiking neurons“. In: *Nature Neuroscience* 12, p. 250. URL: http://dx.doi.org/10.1038/nn.2264%2010.1038/nn.2264%20https://www.nature.com/articles/nn.2264#supplementary-information (cit. on p. 27).

Vasilaki, Eleni, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner (2009). „Spike-Based Reinforcement Learning in Continuous State and Action Space: When Policy Gradient Methods Fail“. In: *PLoS Computational Biology* 5.12. Ed. by Karl J. Friston, e1000586. DOI: 10.1371/journal.pcbi.1000586. URL: http://www.ncbi.nlm.nih.gov/pubmed/19997492%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2778872%20http://dx.plos.org/10.1371/journal.pcbi.1000586 (cit. on pp. 27, 30).

Veendrick, Harry (2017). *Nanometer CMOS ICs*. 2nd ed. Heidelberg: Springer International Publishing. DOI: 10.1007/978-3-319-47597-4 (cit. on p. 2).

Wise, Roy A (2004). „Dopamine, learning and motivation“. In: *Nature Reviews Neuroscience* 5, p. 483. URL: https://doi.org/10.1038/nrn1406%2010.1038/nrn1406 (cit. on p. 25).

Wunderlich, Timo (2016). *Synaptic Calibration on the HICANN-DLS Neuromorphic Chip*. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/download.php/5766/temp/3380.pdf (cit. on p. 13).

– (2019). *Neuromorphic R-STDP Experiment Simulation*. Heidelberg. URL: https://github.com/electronicvisions/model-sw-pong (cit. on p. 42).

Wunderlich, Timo, Akos F. Kungl, Eric Müller, et al. (2018). „Demonstrating Advantages of Neuromorphic Computation: A Pilot Study“. In: URL: http://arxiv.org/abs/1811.03618 (cit. on p. vii).

Xie, Xiaohui and H. Sebastian Seung (2004). „Learning in neural networks by reinforcement of irregular spiking“. In: *Physical Review E* 69.4, p. 041909. DOI: 10.1103/PhysRevE.69.041909. URL: https://link.aps.org/doi/10.1103/PhysRevE.69.041909 (cit. on p. 27).

Yu, T and G Cauwenberghs (2010). „Analog VLSI Biophysical Neurons and Synapses With Programmable Membrane Channel Kinetics“. In: *IEEE Transactions on Biomedical Circuits and Systems* 4.3, pp. 139–148. DOI: 10.1109/TBCAS.2010.2048566 (cit. on p. 8).

List of Figures

List of Tables

Erklärung:

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

Heidelberg, den ....