

This article is bound to be published in *Frontiers in Neuromorphic Engineering* (<https://www.frontiersin.org/articles/10.3389/fnins.2019.00260>).
 Please cite it as *Wunderlich et al., Frontiers in Neuromorphic Engineering (2019)*, doi: [10.3389/fnins.2019.00260](https://doi.org/10.3389/fnins.2019.00260).

Demonstrating Advantages of Neuromorphic Computation: A Pilot Study

Timo Wunderlich^{1,*}, Akos F. Kungl^{1,**}, Eric Müller¹, Andreas Hartel¹, Yannik Stradmann¹, Syed Ahmed Aamir¹, Andreas Grübl¹, Arthur Heimbrecht¹, Korbinian Schreiber¹, David Stöckel¹, Christian Pehle¹, Sebastian Billaudelle¹, Gerd Kiene¹, Christian Mauch¹, Johannes Schemmel¹, Karlheinz Meier¹, and Mihai A. Petrovici^{1,2}

¹Kirchhoff Institute for Physics, Heidelberg University, Heidelberg, Germany

²Department of Physiology, University of Bern, Bern, Switzerland

*timo.wunderlich@kip.uni-heidelberg.de

**fkungl@kip.uni-heidelberg.de

Abstract

Neuromorphic devices represent an attempt to mimic aspects of the brain's architecture and dynamics with the aim of replicating its hallmark functional capabilities in terms of computational power, robust learning and energy efficiency. We employ a single-chip prototype of the BrainScaleS 2 neuromorphic system to implement a proof-of-concept demonstration of reward-modulated spike-timing-dependent plasticity in a spiking network that learns to play a simplified version of the Pong video game by smooth pursuit. This system combines an electronic mixed-signal substrate for emulating neuron and synapse dynamics with an embedded digital processor for on-chip learning, which in this work also serves to simulate the virtual environment and learning agent. The analog emulation of neuronal membrane dynamics enables a 1000-fold acceleration with respect to biological real-time, with the entire chip operating on a power budget of 57 mW. Compared to an equivalent simulation using state-of-the-art software, the on-chip emulation is at least one order of magnitude faster and three orders of magnitude more energy-efficient. We demonstrate how on-chip learning can mitigate the effects of fixed-pattern noise, which is unavoidable in analog substrates, while making use of temporal variability for action exploration. Learning compensates imperfections of the physical substrate, as manifested in neuronal parameter variability, by adapting synaptic weights to match respective excitability of individual neurons.

1 Introduction

Neuromorphic computing represents a novel paradigm for non-Turing computation that aims to reproduce aspects of the ongoing dynamics and computational functionality found in biological brains. This endeavor entails an abstraction of the brain’s neural architecture that retains an amount of biological fidelity sufficient to reproduce its functionality while disregarding unnecessary detail. Models of neurons, which are considered the computational unit of the brain, can be emulated using electronic circuits or simulated using specialized digital systems [1, 2].

BrainScaleS 2 (BSS2) is a neuromorphic architecture consisting of CMOS-based ASICs [3, 4] which implement physical models of neurons and synapses in analog electronic circuits while providing facilities for user-defined learning rules. A number of features distinguish BSS2 from other neuromorphic approaches, such as a speed-up factor of 10^3 compared to biological neuronal dynamics, correlation sensors for spike-timing-dependent plasticity in each synapse circuit and an embedded processor [4], which can use neural network observables to calculate synaptic weight updates for a broad range of plasticity rules. The flexibility enabled by the embedded processor is a particularly useful feature given the increasing effort invested in synaptic plasticity research, allowing future findings to be accommodated easily. The study at hand uses a single-chip prototype version of the full system, which allows the evaluation of the planned system design on a smaller scale.

Reinforcement learning has been prominently used as the learning paradigm of choice in machine learning systems which reproduced, or even surpassed, human performance in video and board games [5–7]. In reinforcement learning, an agent interacts with its environment and receives reward based on its behavior. This enables the agent to adapt its internal parameters so as to increase the potential for reward in the future [8]. In the last decades, research has found a link between reinforcement learning paradigms used in machine learning and reinforcement learning in the brain [for a review, see 9]. The neuromodulator dopamine was found to convey a reward prediction error, akin to the temporal difference error used in reinforcement learning methods [8, 10]. Neuromodulated plasticity can be modeled using three-factor learning rules [11, 12], where the synaptic weight update depends not only on the learning rate and the pre- and post-synaptic activity but also on a third factor, representing the neuromodulatory signal, which can be a function of reward, enabling reinforcement learning.

In this work, we demonstrate the advantages of neuromorphic computation by showing how an agent controlled by a spiking neural network (SNN) learns to solve a smooth pursuit task via reinforcement learning in a fully embedded perception-action loop that simulates the classic Pong video game on the BSS2 prototype. Measurements of time-to-convergence, power consumption and sensitivity to parameter noise demonstrate the advantages of our neuromorphic solution compared to classical simulation on a modern CPU that runs the NEST simulator [13]. The on-chip learning converges within seconds, which is equivalent to hours in biological terms, while the software simulation is at least an order of magnitude slower and three orders of magnitude less energy-efficient. We find that fixed-pattern noise on BSS2 can be compensated by the chosen learning paradigm, reducing the required calibration precision, and that the results of hyperparameter learning can be transferred between different BSS2 chips. The experiment takes place on the chip fully autonomously, i.e., both the environment and synaptic weight changes are computed using the embedded processor. As the number of neurons (32) and synapses (1024) on the prototype chip constrain the complexity of solvable learning tasks, the agent’s task in this

work is simple smooth pursuit without anticipation. The full system is expected to enable more sophisticated learning, akin to the learning of Pong from pixels that was previously demonstrated using an artificial neural network [5].

2 Materials and Methods

2.1 The BrainScaleS 2 Neuromorphic Prototype Chip

The BSS2 prototype is a neuromorphic chip and the predecessor of a large-scale accelerated network emulation platform with flexible plasticity rules [4]. It is manufactured using a 65 nm CMOS process and is designed for mixed-signal neuromorphic computation. All experiments in this work were performed on the second prototype version. Future chips will be integrated into a larger setup using wafer-scale technology [14, 15], thereby enabling the emulation of large plastic neural networks.

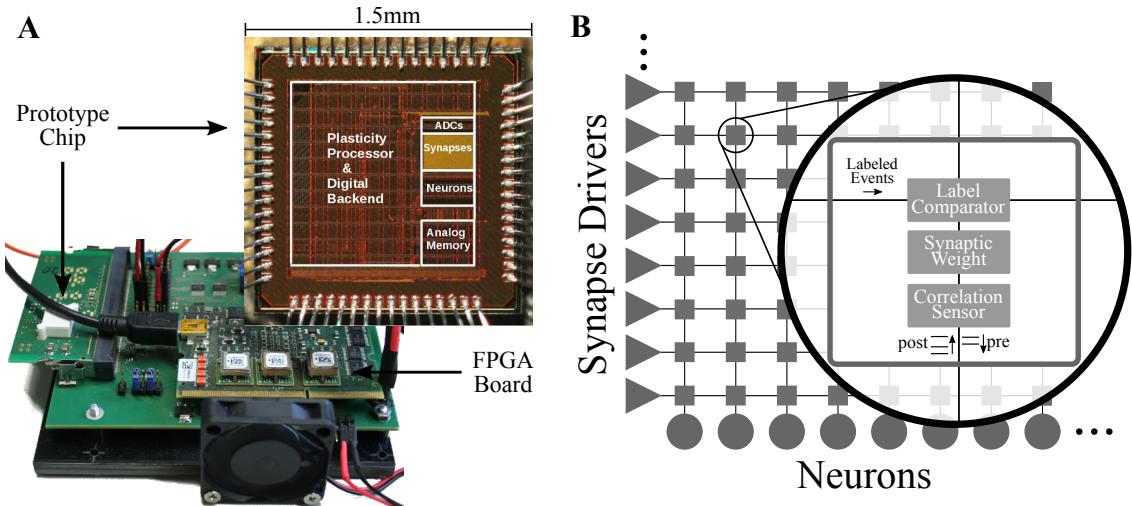


Figure 1: Physical setup and neural network schematic. **A:** In the foreground: BSS2 prototype chip with demarcation of different functional parts. In the background: the development board on which the chip is mounted. Adapted from [3]. **B:** Schematic of the on-chip neural infrastructure. Each of the 32 implemented neurons is connected to one column of the synapse array, where each column comprises 32 synapses. Synapse drivers allow row-wise injection of individually labeled (6-bit) spike events. Each synapse locally stores a 6-bit label and a 6-bit weight and converts spike events with a matching label to current pulses traveling down towards the neuron. Each synapse also contains an analog sensor measuring the temporal correlation of pre- and post-synaptic events (see Section 2.1.5).

2.1.1 Experimental Setup

The BSS2 prototype setup is shown in Figure 1 A and contains the neuromorphic chip mounted on a prototyping board. The chip and all of its functional units can be accessed and configured from either a Xilinx Spartan-6 FPGA or the embedded processor (see Section 2.1.4). The FPGA in turn can be accessed via a USB-2.0 connection between the prototype setup and the host computer. In addition to performing chip configuration, the FPGA can also provide hard real-time playback of input and recording of output data.

Experiments are described by the user through a container-based programming interface which provides access to all functional units such as individual neuron circuits or groups of synapses. The experiment configuration is transformed into a bitstream and uploaded to DRAM attached to the FPGA. Subsequently, the software starts the experiment and a sequencer logic in the FPGA begins to play back the experiment data (e.g., input spike trains) stored in the DRAM. At the same time, output from the chip is recorded to a different memory area in the DRAM. Upon completion of the experiment, the host computer downloads all recorded output from the FPGA memory.

2.1.2 Neurons & Synapses

Our approach to neuromorphic engineering follows the idea of “physical modeling”: the analog neuronal circuits are designed to have similar dynamics compared to their biological counterparts, making use of the physical characteristics of the underlying substrate. The BSS2 prototype chip contains 32 analog neurons based on the Leaky Integrate-and-Fire (LIF) model [3, 16]. Additionally, each neuron has an 8-bit spike counter, which can be accessed and reset by the the embedded processor ([4], see Section 2.1.4) for plasticity-related calculations.

In contrast to other neuromorphic approaches [17–21], this implementation uses the fast supra-threshold dynamics of CMOS transistors in circuits which mimic neuronal membrane dynamics. In the case of BSS2, this approach provides time constants that are smaller than their biological counterparts by three orders of magnitude, i.e., the hardware operates with a speed-up factor of 10^3 compared to biology, independent of the network size or plasticity model. Throughout the manuscript, we provide the true (wall-clock time) values, which are typically on the order of microseconds, compared to the millisecond-scale values usually found in biology.

The 32-by-32 array of synapses is arranged such that each neuron can receive input from a column of 32 synapses (see Figure 1 B). Each row consisting of 32 synapses can be individually configured as excitatory or inhibitory and receives input from a synapse driver that injects labeled digital pre-synaptic spike packets. Every synapse compares its label (a locally stored configurable address) with the label of a given spike packet and if they match, generates a current pulse with an amplitude proportional to its 6-bit weight that is sent down along the column towards the post-synaptic neuron. There, the neuron circuit converts it into an exponential post-synaptic current (PSC), which is injected into the neuronal membrane capacitor.

Post-synaptic spikes emitted by a neuron are signaled (back-propagated) to every synapse in its column, which allows the correlation sensor in each synapse to record the time elapsed between pre- and post-synaptic spikes. Thus, each synapse accumulates correlation measurements that can be read out by the embedded processor, to be used, among other observables, for calculating weight updates (see Section 2.1.5 for a detailed description).

2.1.3 Calibration and Configuration of the Analog Neurons

Neurons are configured using on-chip analog capacitive memory cells [22]. The ideal LIF model neuron with one synapse type and exponential PSCs can be characterized by six parameters: membrane time constant τ_{mem} , synaptic time constant τ_{syn} , refractory period τ_{ref} , resting potential v_{leak} , threshold potential v_{thresh} , reset potential v_{reset} . The neuromorphic implementation on the chip carries 18 tunable parameters per neuron and one global parameter [3]. Most of these hardware parameters are used to set the circuits to the proper point of operation and therefore have fixed values that are calibrated once for any given chip; for the experiments described here,

the six LIF model parameters mentioned above are fully controlled by setting only six of the hardware parameters per neuron.

Manufacturing variations cause fixed-pattern noise (see Section 2.2), therefore each neuron circuit behaves differently for any given set of hardware parameters. In particular, the time constants (τ_{mem} , τ_{syn} , τ_{ref}) display a high degree of variability. Therefore, in order to accurately map user-defined LIF time constants to hardware parameters, neuron circuits are calibrated individually. Using this calibration data reduces deviations from target values to less than 5% [3, see also Figure 2 B].

2.1.4 Plasticity Processing Unit

To allow for flexible implementation of plasticity algorithms, the chip uses a Plasticity Processing Unit (PPU), which is a general-purpose 32-bit processor implementing the PowerPC-ISA 2.06 instruction set and custom vector extensions [4]. In the used prototype chip, it is clocked at a frequency of 98 MHz and has access to 16 KiB of main memory. Vector registers are 128-bit wide and can be processed in slices of eight 16-bit or sixteen 8-bit units within one clock cycle. The vector extension unit is loosely coupled to the general-purpose part. When fetching vector instructions, the commands are inserted into a dedicated command queue which is read by the vector unit. Vector commands are decoded, distributed to the arithmetic units and executed as fast as possible.

The PPU has dedicated fully-parallel access ports to synapse rows, enabling row-wise readout and configuration of synaptic weights and labels. This enables efficient row-wise vectorized plasticity processing. Modifications of connectivity, neuron and synapse parameters are supported during neural network operation. The PPU can be programmed using assembly and higher-level languages such as C or C++ to compute a wide range of plasticity rules. Compiler support for the PPU is provided by a customized gcc [23, 24]. The software used in this work is written in C, with vectorized plasticity processing implemented using inline assembly instructions.

2.1.5 Correlation Measurement at the Synapses

Every synapse in the synapse array contains two analog units that record the temporal correlation between nearest-neighbor pairs of pre- and post-synaptic spikes. For each such pair, a dedicated circuit measures the value of either the causal (pre before post) or anti-causal (post before pre) correlation, which is modeled as an exponentially decaying function of the spike time difference [4]. The values thus measured are accumulated onto two separate storage capacitors per synapse. In an idealized model, the voltages across the causal and anti-causal storage capacitor are

$$a_+ = \sum_{\text{pre-post}} \eta_+ \exp \left(-\frac{t_{\text{post}} - t_{\text{pre}}}{\tau_+} \right) \quad (1)$$

and

$$a_- = \sum_{\text{post-pre}} \eta_- \exp \left(-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_-} \right), \quad (2)$$

respectively, with decay time constants τ_+ and τ_- and scaling factors η_+ and η_- . These accumulated voltages represent non-decaying eligibility traces that can be read out by the PPU using column-wise 8-bit Analog-to-Digital Converters (ADCs), allowing row-wise parallel readout.

Fixed-pattern noise introduces variability among the correlation units of different synapses, as visible in Figure 2 A. The experiments described here only use the causal traces a_+ to calculate weight updates.

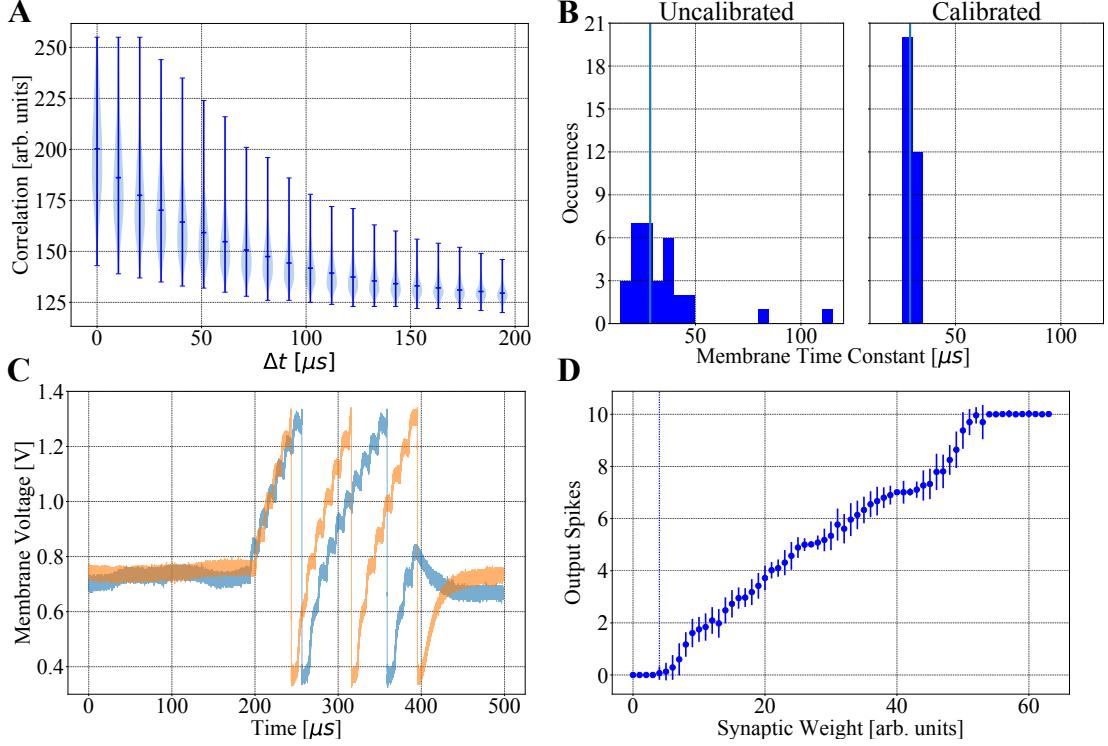


Figure 2: BSS2 is subject to fixed-pattern noise and temporal variability. **A:** Violin plot of the digitized output of the 1024 causal correlation sensors (a_+ , see Equation (1)) on a sample chip (chip #1) as a function of the time interval between a single pre-post spike pair. **B:** Distribution of membrane time constants τ_m over all 32 neurons with and without calibration. The target value is $28.5\ \mu\text{s}$ (vertical blue lines). **C:** Effects of temporal variability. A regular input spike train containing twenty spikes spaced by $10\ \mu\text{s}$, as used in the learning task, transmitted via one synapse, elicits different membrane responses in two trials. **D:** Mean and variance of the output spike count as a function of synaptic weight, averaged over 100 trials, for a single exemplary neuron receiving the input spike train from (C). The spiking threshold weight (the smallest weight with a higher than 5 % probability of eliciting an output spike under the given stimulation paradigm) is indicated by the dotted blue line. Trial-to-trial variation of the number of output spikes at fixed synaptic weight is due to temporal variability and mediates action exploration.

2.2 Types of Noise on BSS2

The BSS2 prototype has several sources of parameter variability and noise, as does any analog hardware. We distinguish between fixed-pattern noise and temporal variability.

Fixed-pattern noise refers to the systematic deviation of parameters (e.g., transistor parameters) from the values targeted during chip design. This type of noise is caused by the inaccuracies of the manufacturing process and unavoidable stochastic variations of process parameters. Fixed-pattern noise is constant in time and induces heterogeneity between neurons and synapses, but calibration can reduce it to some degree. The effects of the calibration on the distribution of the membrane time constant τ_{mem} are shown in Figure 2 B.

Temporal variability continually influences circuits during their operation, leading to fluctua-

tions of important dynamical variables such as membrane potentials. Typical sources of temporal variability are crosstalk, thermal noise and the limited stability of the analog parameter storage. These effects can cover multiple timescales and lead to variable neuron spike responses, even when the input spike train remains unchanged between trials. A concrete example of trial-to-trial variability of a neuron's membrane potential evolution, output spike timing and firing rate caused by temporal variability is shown in Figure 2 C, D for two trials of the same experiment, using the same input spike train and parameters, with no chip reconfiguration between trials.

2.3 Reinforcement Learning with Reward-Modulated STDP

In reinforcement learning, a behaving agent interacts with its environment and tries to maximize the expected future reward it receives from the environment as a consequence of this interaction [8]. The techniques developed to solve problems of reinforcement learning generally do not involve spiking neurons and are not designed to be biologically plausible. Yet reinforcement learning evidently takes place in biological SNNs, e.g., in basic operant conditioning [25–27]. The investigation of spike-based implementations with biologically inspired plasticity rules is therefore an interesting subject of research with evident applications for neuromorphic devices. The learning rule used in this work, Reward-modulated Spike-Timing Dependent Plasticity (R-STDP) [28–30], represents one possible implementation.

R-STDP is a three-factor learning rule that modulates the effect of unsupervised STDP using a reward signal. Recent work has used R-STDP to reproduce Pavlovian conditioning as in [28] on a specialized neuromorphic digital simulator, achieving real-time simulation speed [31]. While not yet directly applied to an analog neuromorphic substrate, aspects of this learning paradigm have already been studied in software simulations, under constraints imposed by the BSS2 system, in particular concerning the effect of discretized weights, with promising results [32]. Furthermore, it was previously suggested that trial-to-trial variations of neuronal firing rates as observed in cortical neurons can benefit learning in a reinforcement learning paradigm, rather than being a nuisance [33–35]. Our experiments corroborate this hypothesis by explicitly using trial-to-trial variations due to temporal variability for action exploration.

The reward mechanism in R-STDP is biologically inspired: the phasic activity of dopamine neurons in the brain was found to encode expected reward [10, 36, 37] and dopamine concentration modulates STDP [38–40]. R-STDP and similar reward-modulated Hebbian learning rules have been used to solve a variety of learning tasks in simulations, such as reproducing temporal spike patterns and spatio-temporal trajectories [29, 30, 41], reproducing the results of classical conditioning [28], making a recurrent neural network exhibit specific periodic activity and working-memory properties [42] and reproducing the seminal biofeedback experiment by Fetz and Baker [27, 33]. Compared to classic unsupervised STDP, using R-STDP was shown to improve the performance of a spiking convolutional neural network tasked with visual categorization [43, 44].

In contrast to other learning rules in reinforcement learning, R-STDP is not derived using gradient descent on a loss function; rather, it is motivated heuristically [11], the idea being to multiplicatively modulate STDP using a reward term.

We employ the following form of discrete weight updates using R-STDP:

$$\Delta w_{ij} = \beta \cdot (R - b) \cdot e_{ij}, \quad (3)$$

where β is the learning rate, R is the reward, b is a baseline and e_{ij} is the STDP eligibility trace which is a function of the pre- and post-synaptic spikes of the synapse connecting neurons i and j . The choice of the baseline reward b is critical: a nonzero offset introduces an admixture of unsupervised learning via the unmodulated STDP term, and choosing b to be the task-specific expected reward $b = \langle R \rangle_{\text{task}}$ leads to weight updates that capture the covariance of reward and synaptic activity [11]:

$$\langle \Delta w_{ij} \rangle_{\text{task}} = \langle R \cdot e_{ij} \rangle_{\text{task}} - \langle R \rangle_{\text{task}} \cdot \langle e_{ij} \rangle_{\text{task}} = \text{Cov}(R, e_{ij}) . \quad (4)$$

This setting, which we also employ in our experiments, makes R-STDP a statistical learning rule in the sense that it captures correlations of joint pre- and post-synaptic activity and reward; this information is collected over many trials of any single learning task. The expected reward may be estimated as a moving average of the reward over the last trials of that specific task; task specificity of the expected reward is required when multiple tasks need to be learned in parallel [30].

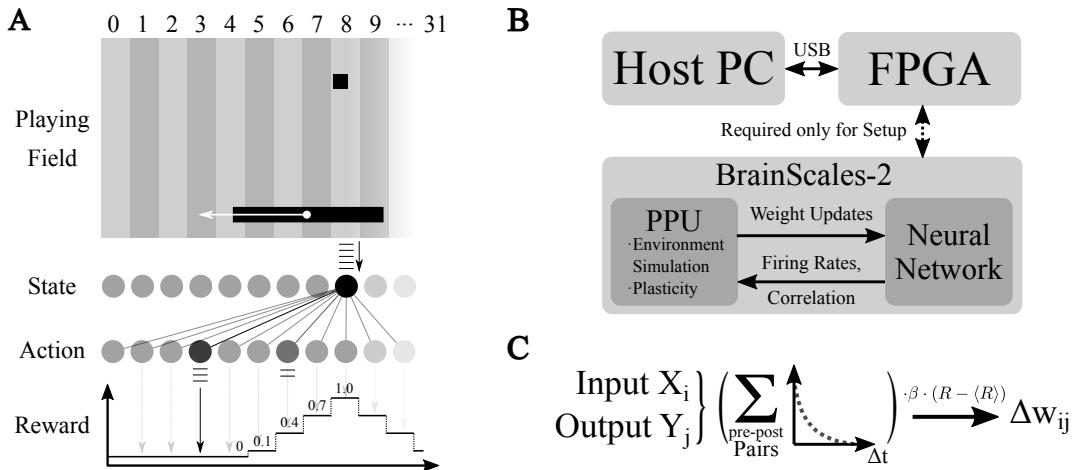


Figure 3: Overview of the experimental setup. **A:** The components of the environment are the playing field with three reflective walls (top, left, right), the ball and the paddle. In the depicted situation, the ball is in column 8 and therefore a uniform spike train is sent to output neurons via the synapses of input unit 8. Output unit 3 fires the most spikes and the paddle therefore moves toward column 3. As the target column is too far away from the ball column, it lies outside of the graded reward window and the reward received by the network is zero (see Equation (5)). **B:** The chip performs the experiment completely autonomously: the environment simulation, spiking network emulation and synaptic plasticity via the PPU are all handled on-chip. The FPGA, which is only used for the initial configuration of the chip, is controlled via USB from the host PC. **C:** The plasticity rule calculated by the PPU. Pre-before-post spike pairs of input unit X_i and output unit Y_j are exponentially weighted with the corresponding temporal distance and added up. This correlation measure is then multiplied with the learning rate and the difference of instantaneous and expected reward to yield the weight change for synapse (i, j) .

2.4 Learning Task and Simulated Environment

Using the PPU, we simulate a simple virtual environment inspired by the Pong video game. The components of the game are a two-dimensional square playing field, a ball and the player's paddle (see Figure 3 A). The paddle is controlled by the chip and the goal of the learning task is to trace the ball. Three of the four playing field sides are solid walls, with the paddle moving along

the open side. The experiment proceeds iteratively and fully on-chip (see Figure 3 B). A single experiment iteration consists of neural network emulation, weight modification and environment simulation. We visualize one iteration as a flowchart in Figure 4 and provide a detailed account in the following.

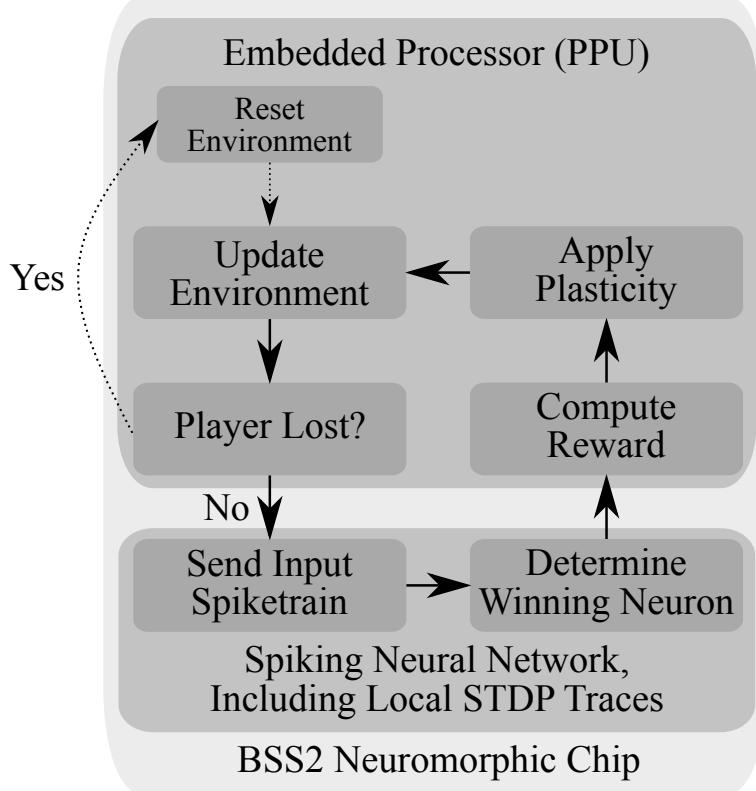


Figure 4: Flowchart of the experiment loop running autonomously on BSS2, using both the analog Spiking Neural Network (SNN) and the embedded processor. The environment is reset by positioning the ball in the middle of the playing field with a random direction of movement at the start of the experiment or upon the agent’s failure to reflect the ball. In the main loop, the (virtual) state unit corresponding to the current ball position transmits a spike train to all neurons in the action layer (see Figure 3). Afterwards, the winning action neuron, i.e., the action neuron that had the highest output spike count, is determined. Then, the reward is determined based on the difference between the ball position and the target paddle position as dictated by the winning neuron (Equation (5)). Using the reward, a stored running average of the reward and the STDP correlation traces computed locally at each synapse during SNN emulation, the weight updates of all synapses are computed (Equation (7)) and applied. The environment, i.e., the position of ball and paddle, are then updated, and the loop starts over.

The game dynamics consist of the ball starting in the middle of the playing field in a random direction with velocity \vec{v} and the paddle which moves with velocity v_p along its baseline. Surfaces elastically reflect the ball. If the paddle misses the ball, the game is reset, i.e., the ball starts from the middle of the field in a random direction. Specific parameter values are given in Table 1.

The paddle is controlled by the neural network emulated on the neuromorphic chip. The network receives the ball position along the paddle’s baseline as input and determines the movement of its paddle via the resulting network activity. The neural network consists of two 32-unit layers which are initialized with all-to-all feed-forward connections (see Table 1), where the individual weights are drawn from a Gaussian distribution and where the first layer represents input/state units u_i and the second layer represents output/action units v_i . The first layer is a

virtual layer for providing input spikes and the second layer consists of the chip's LIF neurons. All synaptic connections are excitatory. Discretizing the playing field into 32 columns along the paddle baseline, we assign a state unit u_i to each column i where $i \in [0, 31]$ and provide input to this unit if the ball is in the corresponding column via a uniform spike train (see Table 1 for parameters).

The action neurons' spike counts ρ_i are used to determine the paddle movement: the unit with the highest number of output spikes $j = \text{argmax}_i(\rho_i)$ determines the paddle's target column j , towards which it moves with constant velocity v_p (see Table 1). If the target column and center position of the paddle match, no movement is performed. Spike counts and in consequence, the target row j are determined after the input spike train has been delivered. If several output units have the same spike count, the winner is chosen randomly among them. Afterwards, the reward R is calculated based on the distance between the target column j and the current column of the ball, k :

$$R = \begin{cases} 1 - |j - k| \cdot 0.3 & \text{if } |j - k| \leq 3 , \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Learning success is therefore graded, i.e., the network obtains reduced reward for less than optimal aiming. The size of the reward window defined by Equation (5) is chosen to match the paddle length.

For every possible task (corresponding to a ball column k), the PPU holds task-specific expected rewards \bar{R}_k , $k \in [0, 31]$ in its memory, which it subtracts from the instantaneous reward R to yield the neuromodulating factor $R - \bar{R}_k$, which is also used to update the task-specific expected reward as an exponentially weighted moving average:

$$\bar{R}_k \leftarrow \bar{R}_k + \gamma (R - \bar{R}_k) , \quad (6)$$

where γ controls the influence of previous iterations. The expected reward of any state is initialized as the first reward received in that state. All 1024 synapse weights w_{mn} from input unit m to output unit n are then updated according to the three-factor rule (see Figure 3 C)

$$\Delta w_{mn} = \beta \cdot (R - \bar{R}_k) \cdot A_{mn}^+ , \quad (7)$$

where β is a learning rate and A_{mn}^+ is a modified version of a_{mn}^+ (see Equation (1)) which has been corrected for offset, digitized to 8 bit and right-shifted by one bit in order to reduce noise. After the weights have been modified, the Pong environment is updated according to the described dynamics and the next iteration begins.

The *mean expected reward*

$$\langle \bar{R} \rangle = \frac{1}{32} \sum_{i=0}^{31} \bar{R}_i , \quad (8)$$

i.e., the average of the expected rewards over all states, represents a measure of the progress of learning in any given iteration. Due to the paddle width and correspondingly graded reward scheme, the agent is able to catch the ball even when it is not perfectly centered below the ball. Therefore, the *performance* in playing Pong can be quantified by

$$P = \frac{1}{32} \sum_{i=0}^{31} [R_i] , \quad (9)$$

where R_i is the last reward received in state i . This provides the percentage of states in which the agent has aimed the paddle such that it is able to catch the ball.

In order to find suitable hyperparameters for the chip configuration, we used a Bayesian parameter optimization based on sequential optimization using decision trees to explore the neuronal and synaptic parameter space while keeping parameters such as game dynamics, input spike train and initial weight distribution fixed. The software package used was scikit-optimize [45]. Initially, 30 random data points were taken, followed by 300 iterations with the next evaluated parameter set being determined by the optimization algorithm FOREST_MINIMIZE with default parameters (maximizing expected improvement, extra trees regressor model, 10000 acquisition function samples, target improvement of 0.01). All neuron and synapse parameters were subject to the optimization, i.e., all neuronal time constants and voltages as well as the time constant and amplitude of the synapse correlation sensors. The results are a common set of parameters for all neurons and synapses (see Table 1).

2.5 Software Simulation with NEST

In order to compare the learning performance and speed of the chip to a network of deterministic, perfectly identical LIF neurons, we ran a software simulation of the experiment using the NEST v2.14.0 SNN simulator [13], using the same target LIF parameters as in our experiments using the chip, with time constants scaled by a factor of 10^3 . We did not include fixed-pattern noise of neuron parameters in the simulation, i.e., all neurons had identical parameters. We used the *iaf_psc_exp* integrate-and-fire neuron model available in NEST, with exponential PSC kernels and current-based synapses. Using NEST’s *noise_generator*, we are able to investigate the effect of injecting Gaussian current noise into each neuron. The scaling factors η_+ and η_- , as well as the time constants τ_+ and τ_- of the correlation sensors were chosen to match the mean values on BSS2. The correlation factor a_+ was calculated within the Python script controlling the experiment using Equation (1) and the spike times provided by the NEST simulator. Hyperparameters such as learning rate and game dynamics (e.g. the reward window defined in Equation (5)) were set to be equivalent to BSS2 and weights were scaled to a dimensionless quantity and discretized to match the neuromorphic emulation.

The synaptic weight updates in each iteration were restricted to those synapses which transmitted spikes, i.e., the synapses from the active input unit to all output units (32 out of the 1024 synapses), as the correlation a_+ of all other synapses is zero in a perfect simulation without fixed-pattern noise. This has the effect of reducing the overall time required to simulate one iteration and is in contrast to the implementation on BSS2, where all synapses are updated in each iteration as there is no guarantee that correlation traces are zero and we excluded this kind of “expert knowledge” from the implementation.

The source code of the simulation is publicly available [46].

3 Results

3.1 Learning Performance

The progress of learning over 10^5 iterations is shown in Figure 5 for both BSS2 (subplot A) and an ideal software simulation with and without injected noise (subplot B). We use both measures

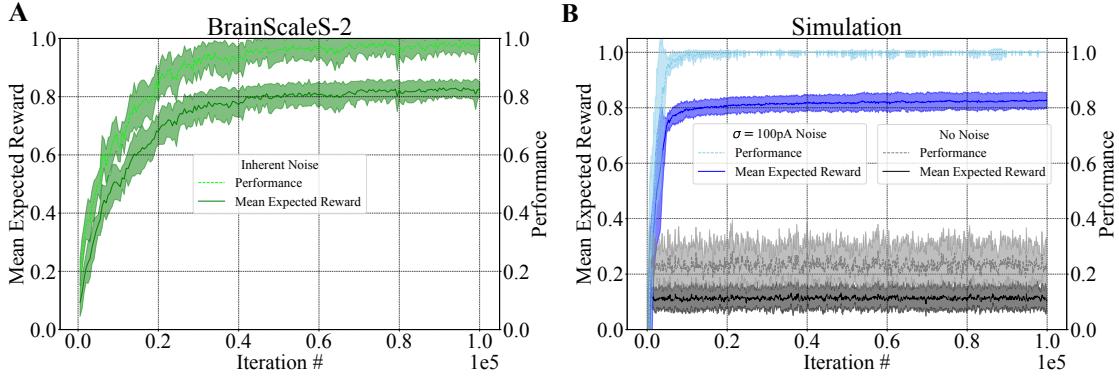


Figure 5: Learning results for BSS2 and the software simulation using NEST, in terms of Mean expected reward (Equation (8)) and Pong performance (Equation (9)). In both cases, we plot the mean and standard deviation (shaded area) of 10 experiments. **A:** BSS2 uses its intrinsic noise as an action exploration mechanism that drives learning. **B:** The software simulation without noise is unable to learn and does not progress beyond chance level. Adding Gaussian zero-mean current noise with $\sigma = 100 \text{ pA}$ to each neuron allows the network to explore actions and enables learning. The simulation converges faster due to the idealized simulated scenario where no fixed-pattern noise is present.

described above to quantify the agent’s success: the mean expected reward (Equation (8)) reflects the agent’s aiming accuracy and the Pong performance (Equation (9)) represents the ability of the agent to catch the ball using its elongated paddle. By repeating the procedure with ten randomly initialized weight matrices, we show that learning is reproducible, with little variation in the overall progress and outcome.

The optimal solution of the learning task is a one-to-one mapping of states to actions that place the center of the paddle directly below the ball at all times. In terms of the neural network, this means that the randomly initialized weight matrix should be dominated by its diagonal elements. We show the weight matrix on BSS2 after 10^5 learning iterations, averaged over the ten different trials depicted in Figure 5, in Figure 6 A. As expected, the diagonals of the matrix are dominant. Note that also slightly off-diagonal synapses are also strengthened, as dictated by the graded reward scheme. The visibly distinguishable vertical lines stem from neuronal fixed-pattern noise, as learning adapts weights to compensate for neuronal variability and one column in the weight matrix corresponds to one neuron.

A screen recording of a live demonstration of the experiment is available at <https://www.youtube.com/watch?v=LW0Y5SSIQU4> and allows the viewer to follow the learning progress in a single experiment.

3.1.1 Temporal Variability on BSS2 Causes Exploration

On BSS2, action exploration and thereby learning is driven by trial-to-trial variations of neuronal spike counts that are due to temporal variability (see Figure 2 C, D).

In contrast, we find that the software simulation without injected noise (see Section 2.5) is unable to progress beyond the mean expected reward received by an agent choosing random actions, which is around $\langle \bar{R} \rangle = 0.1$ (the randomness in the weight matrix initialization leads to some variation in the mean expected reward after learning). The only non-deterministic mechanism in the software simulation without noise is due to the fact that if several action neurons elicit the same number of spikes, the winner is chosen randomly among them, but its effects are negligible. Injecting Gaussian current noise with zero mean and a standard deviation

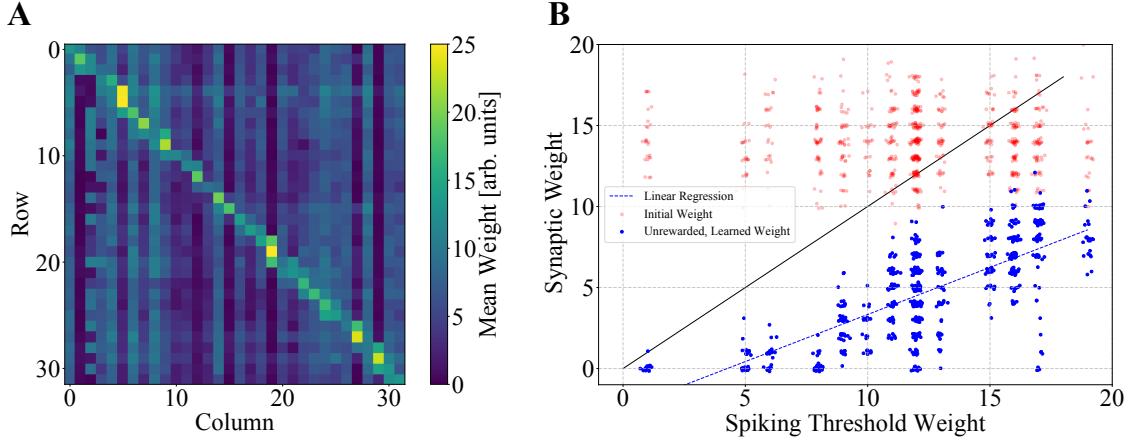


Figure 6: **A:** Synaptic weight matrix after learning, averaged over the 10 trials depicted in Figure 5. Weights on and near the diagonal dominate, corresponding to the goal of the learning task. The noticeable vertical stripes are a direct consequence of learning, which implicitly compensates neuronal variability (each column represents all input synapses of an action neuron). **B:** Learning compensates circuit variability. Weights corresponding to unrewarded actions ($R = 0$) are systematically pushed below the threshold weight of the respective neuron, i.e., below the main diagonal. This leads to a correlation of learned weight and threshold (Pearson's $r = 0.76$). Weights are plotted with slight jitter along both axes for better visibility.

of $\sigma = 100$ pA into each neuron independently enables enough action exploration to converge to similar performance as BSS2. Compared to BSS2, the simulation with injected noise converges faster and to a higher level of Pong performance; this is due to the fact that the simulation contains no fixed-pattern noise, the network starts from an unbiased, perfectly balanced state and that Gaussian current noise is not an exact model of the temporal variability found in BSS2.

We can therefore conclude that under an appropriate learning paradigm, analog-hardware-specific temporal variability that is generally regarded as a nuisance can become a useful feature. Adding an action exploration mechanism similar to ϵ -greedy action selection would enable the software simulation to learn with guaranteed convergence to optimal performance, but would come at the cost of additional computational resources required for emulating such a mechanism.

3.2 Learning Is Calibration

The learning process adjusts synaptic weights such that individual differences in neuronal excitability on BSS2 are compensated. We correlated learned weights to neuronal properties and found that learning shapes a weight matrix that is adapted to a specific pattern of neuronal variability.

Each neuron is the target of 32 synapses. A subset of these are systematically potentiated, as they correspond to actions yielding a reward higher than the current expected reward, while the remainder is depressed, as these synapses correspond to actions yielding less reward. This leads to the diagonally-dominant matrix depicted in Figure 6 A. At the same time, neuronal variability leads to different spiking-threshold weights, i.e., the smallest synaptic weight for which the neuron elicits one spike with a probability higher than 5 %, given the fixed input spike train used in the experiment (see Figure 2 D). The learning process pushes the weights of unrewarded ($R = 0$) synapses below the spiking threshold of the respective neuron, thereby compensating variations of neuronal excitability, leading to a correlation of both quantities. Using the weights depicted in

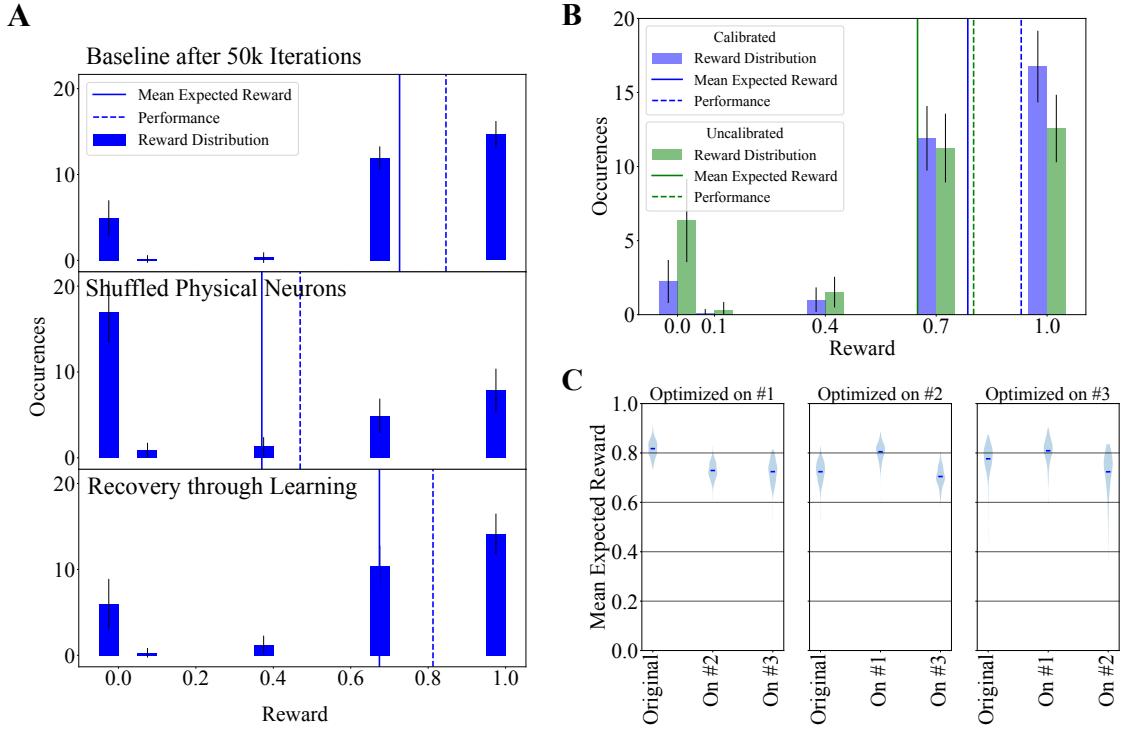


Figure 7: Learning can largely supplant individual calibration of neuron parameters and adapts synaptic weights to compensate for neuronal variability. **A:** Top: Reward distribution over 100 experiments measured with a previously learned, fixed weight matrix. Middle: Same as above, with randomly permuted neurons in each of the 100 experiments. The agent’s performance and therefore its received reward decline due to the weight matrix being adapted to a specific pattern of fixed-pattern noise. Bottom: Allowing the agent to learn for 50000 additional iterations after having randomly shuffled its neurons leads to weight re-adaptation, increasing its performance and received reward. In these experiments, LIF parameters were not calibrated individually per neuron. **B:** Reward distribution after 50000 learning iterations in 100 experiments for a calibrated and an uncalibrated system, with learning being largely able to compensate for the difference. **C:** Results can be reproduced on different chips. Violin plot of mean expected reward after hyperparameter optimization on chips #1, #2 and #3. Results are shown for the calibrated case. All other results in this manuscript were obtained using Chip #1.

(A) and empirically determined spiking thresholds, we found that for each neuron the weights of synapses which correspond to unrewarded actions were correlated with the spiking-threshold with a correlation coefficient of $r = 0.76$ (see Figure 6 B).

The learned adaptation of the synaptic weights to neuronal variability can be disturbed by randomly shuffling the assignment of logical action units to physical neurons. This is equivalent to the thought experiment of physically shuffling neurons on the chip and leads to synaptic weights which are maladapted to their physical substrate, i.e., efferent neuronal properties. In the following, we demonstrate the detrimental effects of such neuronal permutations and that the system can recover performance by subsequent learning.

We considered a weight matrix after 50000 learning iterations and measured the resulting reward distribution over 100 experiments with learning turned off. Here, “reward distribution” refers to the distribution of the most recent reward over all 32 states. The top panel of Figure 7 A shows the reward distribution for this baseline measurement. Mean expected reward and performance for this measurement are $\langle \bar{R} \rangle = 0.73 \pm 0.05$ and $P = 0.85 \pm 0.06$, respectively.

The same weight matrix was applied to 100 systems with randomly shuffled physical neuron assignment and the reward distribution measured as before with learning switched off, yielding the distribution depicted in the middle panel of the same plot, with mean expected reward and performance of $\langle \bar{R} \rangle = 0.37 \pm 0.09$ and $P = 0.47 \pm 0.11$. Each of the 100 randomly shuffled systems was then subjected to 50000 learning iterations, starting from the maladapted weight matrix, leading to the distribution shown in the bottom panel, with mean expected reward and performance of $\langle \bar{R} \rangle = 0.67 \pm 0.07$ and $P = 0.81 \pm 0.09$.

This demonstrates that our learning paradigm implicitly adapts synaptic weights to a specific pattern of neuronal variability and compensates for fixed-pattern noise. In a more general context, these findings support the idea that for neuromorphic systems endowed with synaptic plasticity, time-consuming and resource-intensive calibration of individual components can be, to a large extent, supplanted by on-chip learning.

3.3 Learning Robustness

One of the major concerns when using analog electronics is the control of fixed-pattern noise. We can partly compensate for fixed-pattern noise using a calibration routine [3], but this procedure is subject to a trade-off between accuracy and the time and computational resources required for obtaining the calibration data. Highly precise calibration is costly because it requires an exhaustive mapping of a high-dimensional parameter space, which has to be done for each chip individually. A faster calibration routine, on the other hand, necessarily involves taking shortcuts, such as assuming independence between the influence of hardware parameters, thereby potentially leading to systematic deviations from the target behavior. Furthermore, remaining variations can affect the transfer of networks between chips and therefore potentially impact learning success when using a given set of (hyper-)parameters. We discuss these issues in the following. All results in this section were obtained after using 50000 training iterations, which take around 25 s of wall-clock time on our BSS2 prototypes.

3.3.1 Impact of Time Constant Calibration

The neuronal calibration (see Section 2.1.3) adjusts hardware parameters controlling the LIF time constants (τ_{mem} , τ_{ref} and τ_{syn}) on a per-neuron basis to optimally match target time constants (Table 1), compensating neuronal variability. Depending on the target parameter value, it is possible to reduce the standard deviations of the LIF time constants across a chip by up to an order of magnitude [3].

We investigated the effect of uncalibrated neuronal time constants on learning. The uncalibrated state is defined by using the same value for a given hardware parameter for all neurons on a chip. To set up a reasonable working point, we chose this to be the average of the calibrated values of all neurons on the chip. A histogram of the membrane time constants of all neurons on the chip in the calibrated and uncalibrated state is given in Figure 2 B. In both cases, voltages (v_{leak} , v_{thresh} , v_{reset}) were uncalibrated.

We measured the reward distribution after learning in both the calibrated and the uncalibrated state, performing 100 experiments in both cases (Figure 7 B). Even in the uncalibrated state, learning was possible using only the inherent hardware noise for action exploration, albeit with some loss (around 17 %) in mean expected reward. The mean expected reward and performance

in the calibrated state are $\langle \bar{R} \rangle = 0.79 \pm 0.05$ and $P = 0.93 \pm 0.05$, respectively. In the uncalibrated state, the values are $\langle \bar{R} \rangle = 0.65 \pm 0.08$ and $P = 0.80 \pm 0.09$.

As a corollary, the reward distributions depicted in Figure 7 B suggest that narrowing the reward window (defined in Equation (5)) to half the original size (i.e., removing the $R \leq 0.4$ part) would only have a small effect on converged Pong performance.

3.3.2 Transferability of Results Between Chips

All results presented thus far were obtained using one specific chip (henceforth called chip #1) and parameter set (given in Table 1 as set #1). These parameters were found using a hyper-parameter optimization procedure (see Section 2) that was performed on this chip. In addition, we performed the same optimization procedure on two other chips, using the original optimization results as a starting point, which yielded three parameter sets in total. The two additional parameter sets, sets #2 and #3 corresponding to chips #2 and #3, are also given in Table 1.

We then investigated the effects of transferring each parameter set to the other chips, testing all six possible transitions of learned parameters between the chips. To this end, we conducted 200 trials consisting of 50000 learning iterations on every chip for every parameter set and compare the resulting mean expected reward in Figure 7 C. Learning results were similar across all nine experiments. As expected due to process variations, there were small systematic deviations between the chips, with chip #1 slightly outperforming the other two for all three hyperparameter sets, but in all scenarios the agent successfully learned to play the game. These results suggest that chips can be used as drop-in replacements for other chips and that the hyperparameter optimization does not have to be specific to a particular substrate to yield good results.

3.4 Speed and Power Consumption

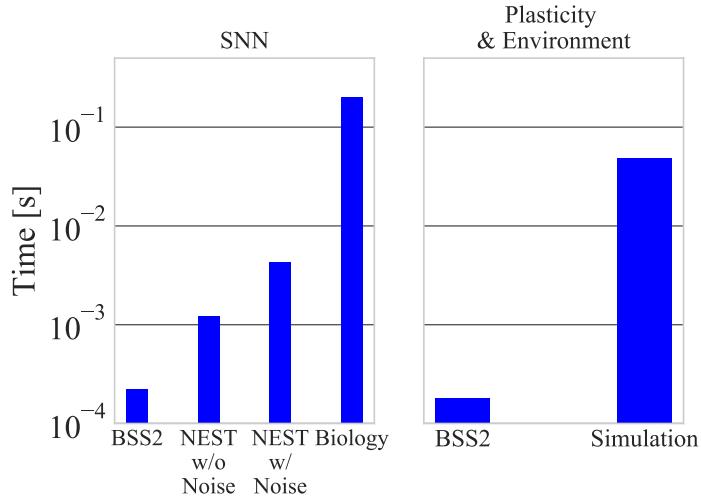


Figure 8: A single experiment iteration on BSS2 takes around $400\mu\text{s}$, with $220\mu\text{s}$ devoted to network emulation and $180\mu\text{s}$ to plasticity calculation. The spiking network activity corresponds to 200 biological ms. In the software simulation, the neural network with and without noise is simulated in 4.3 ms and 1.2 ms, respectively, while plasticity calculations take around 50 ms.

3.4.1 Speed

An essential feature of BSS2 is its 10^3 speed-up factor compared to biological real-time. To put this in perspective, we compared the emulation on our BSS2 prototypes to a software simulation with NEST running on a conventional CPU (see Section 2.5). We found that a single experiment iteration in the simulation takes approximately 50 ms when running it on a single core of an Intel i7-4771 CPU (utilizing more cores does not lead to a faster simulation due to the small size of the network). For our speed comparison, we differentiate the case of no injected noise (where the network's time evolution is fully deterministic) and the case where we use NEST's *noise_generator* module to inject current noise into each neuron. When noise is injected, the 200 ms of neural network activity is simulated in 4.3 ms; without noise, the activity is simulated in 1.2 ms. This is the time that is spent in NEST's state propagation routine (the *Simulate* routine). The remainder is spent calculating the plasticity rule and environment in the additional Python script, which we consider separately as it was external to NEST.

In contrast, BSS2 takes 0.4 ms per iteration as measured using the time stamp register of the PPU. This time is approximately equally divided between neural network emulation and other calculations, including the updates of the synaptic weights and the environment. The total time duration of an experiment with 50000 learning iterations is 25 s for BSS2 and 40 min for the software simulation. A constant overhead of around 5 s when using the chip is spent on calibration, connection setup and configuring the chip.

The comparison between BSS2 and the software simulation is visualized in Figure 8. Note that while the calculation of eligibility traces in software was not optimized for speed, it incurs a significant computational cost, especially as it scales linearly with the number of synapses and therefore approximately quadratically with the network size. This is in contrast to the emulation on BSS2, where the eligibility trace is measured locally with analog circuitry at each synapse during network emulation.

In both cases, the time taken for environment simulation is negligible compared to plasticity calculation. We have confirmed that these measurements are independent on learning progress (i.e., measurements during the first iterations and with a diagonal weight matrix are equal).

3.4.2 Power Consumption

We measured the current drawn by the CPU during the software simulation of the neural network, i.e., during NEST's numerical simulation routine as well as when idling using the EPS 12 V power supply cable that supplies only the CPU. Again, we differentiate the cases of no noise and injected current noise. Based on these measurements, we determined a lower bound of 24 W for the CPU power consumption during SNN simulation without noise and a lower bound of 25 W when injecting noise. A single simulation iteration in software without noise, taking 1.2 ms and excluding plasticity and environment simulation, therefore consumes at least 29 mJ. When we inject current noise, the simulation takes 4.3 ms which corresponds to an energy consumption of 106 mJ per iteration.

By measuring the current drawn by BSS2 during the experiment, we calculated the power dissipated by it to be 57 mW, consistent with the measurement of another network in [3] (this is not considering the power drawn by the FPGA, which is only used for initial configuration, and other prototype board components). This implies a per-iteration energy consumption, including plasticity and environment simulation, of around $23\mu\text{J}$ by the chip.

These measurements imply that, in a conservative comparison, the emulation using BSS2 is at least three orders of magnitude more energy-efficient than the software simulation.

4 Discussion and Outlook

We demonstrated key advantages of our approach to neuromorphic computing in terms of speed and energy efficiency compared to a conventional approach using dedicated software. The already observable order-of-magnitude speed-up of our BSS2 prototypes compared to software for our small-scale test case is expected to increase significantly for larger networks. At the same time, this performance can be achieved with a three-orders-of-magnitude advantage in terms of power consumption, as our conservative estimate shows.

Furthermore, we showed how substrate imperfections, which inevitably lead to parameter variations in analog neuro-synaptic circuits, can be compensated using an implementation of reinforcement learning via neuromodulated plasticity based on R-STDP. Meta-learning can be done efficiently despite substrate variability, as results of hyperparameter optimization can be transferred across chips. We further find that learning is not only robust against analog temporal variability, but that trial-to-trial variations are in fact used as a computational resource for action exploration.

In this context, it is important to mention that temporal variability is generally undesirable due to its uncontrollable nature and chips are designed with the intention of keeping it to a minimum. Still, learning paradigms that handle such variability gracefully are a natural fit for analog neuromorphic hardware, where noise inevitably plays a role.

Due to the limited size of the chips used in this pilot study, the neural networks discussed here are constrained in size. However, the final system will represent a significant scale-up of this prototype. The next hardware revision will be the first full-size BSS2 chip and will provide 512 neurons, 130k synapses and two embedded processors with appropriately scaled vector units. Neurons on the chip will emulate the Adaptive-Exponential (AdEx) Integrate-and-Fire model with multiple compartments, while synapses will contain additional features for Short-Term Plasticity (STP). Poisson-like input stimuli with rates in the order of MHz, i.e., biological kHz, will be realized using pseudo-random number generators to provide the possibility for stochastic spike input to neurons, yielding controllable noise which can be used for emulation of in-vivo activity and large-scale functional neural networks [47–49]. The full BSS2 neuromorphic system will comprise hundreds of such chips on a single wafer which itself will be interconnected to other wafers, similar to its predecessor BrainScaleS 1 [14]. The study at hand lays the groundwork for future experiments with reinforcement learning in neuromorphic SNNs, where an expanded hardware real-estate will allow the emulation of more complex agents learning to navigate more difficult environments.

The advantages in speed and energy consumption will become even more significant when moving to large networks. In our experiments, the relative speed of the software simulation was due to the small size of the network. State-of-the-art software simulations of large LIF neural networks take minutes to simulate a biological second [50] and even digital simulations on specialized neuromorphic hardware typically achieve real-time operation [31, 51]. On BSS2, the speed-up factor of 10^3 is independent of network size, which, combined with the two embedded processors per chip and the dedicated synapse circuits containing local correlation sensors, enables

the accelerated emulation of large-scale plastic neural networks.

The speed-up factor of BSS2 is both an opportunity and a challenge. In general, rate-based or sampling-based codes would profit from longer integration times enabled by the acceleration. On the other hand, interfacing BSS2 to the real world (e.g. using robotics) requires fast sensors; the speed-up could enable fast sensor-motor loops with possible applications in, for example, radar beam shaping. In general, however, the main advantage of an accelerated system becomes most evident when learning is involved: long-term learning processes lasting several years in biological spiking networks can be emulated in mere hours on BSS2, whereas real-time simulations are unfeasible.

The implemented learning paradigm (R-STDP) and simulated environment (Pong) were kept simple in order to focus our study on the properties of the prototype hardware. R-STDP is a well-studied model that lends itself well to hardware implementation, while the simplified Pong game is a suitable learning task that can be realized on the prototype chip and provides an accessible, intuitive interpretation. Still, the PPU's computational power limits the complexity of environment simulations that can be realized on-chip, especially when simulations have hard real-time constraints. However, such simulations could take place on a separate system that merely provides spike input, collects spike output and provides reward to the neuromorphic system. Alternatively, simulations could be horizontally distributed across PPUs.

Solving more complex learning tasks demands more complex network models. We expect that the future large-scale BSS2 system will be able to instantiate not only larger, but also more complex network models, by offering more flexibility in the choice of spiking neuron dynamics (e.g. AdEx, LIF), short-term synaptic plasticity and enhanced PPU capabilities. However, further theoretical work is required for mapping certain state-of-the-art reinforcement learning models, such as DQN [5] and AlphaGo (Zero) [6, 7] to this substrate. On the other hand, learning paradigms like TD-STDP [12], which implements actor-critic reinforcement learning using a spiking neural network, already match the capabilities of a large-scale version of the BSS2 system and therefore represent suitable candidates for learning in more complex environments with sparse rewards and agent-dependent state transitions.

Conflict of Interest Statement

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Author Contributions

Andreas Hartel (AH), Akos F. Kungl (AFK), Timo Wunderlich (TW) and Mihai A. Petrovici (MAP) designed the study. TW conducted the experiments and the evaluations. TW and AFK wrote the initial manuscript. Eric Müller (ECM) and Christian Mauch (CM) supported experiment realization; ECM coordinated the software development for the neuromorphic systems. Yannik Stradmann (YS) contributed with characterization, calibration testing and debugging of the prototype system. Syed Ahmed Aamir (SAA) designed the neuron circuits. Andreas Grübl (AG) was responsible for chip assembly and did the digital front- and backend implementation. Arthur Heimbrecht (AWH) extended the GNU Compiler Collection backend support for the embedded

processor. David Stöckel (DS) contributed to the host control software development and supported chip commissioning. Korbinian Schreiber (KS) designed and assembled the development board. Christian Pehle (CP) provided FPGA firmware and supported chip commissioning. Gerd Kiene (GK) contributed to the verification of the synaptic input circuits. Johannes Schemmel (JS) is the architect and lead designer of the neuromorphic platform. MAP, Karlheinz Meier (KM), JS, Sebastian Billaudelle (SB), and ECM provided conceptual and scientific advice. All authors contributed to the final manuscript.

Funding

This research was supported by the EU 7th Framework Program under grant agreements 269921 (BrainScaleS), 243914 (Brain-i-Nets), 604102 (Human Brain Project), the Horizon 2020 Framework Program under grant agreement 720270 (Human Brain Project), the Manfred Stärk Foundation and the Deutsche Forschungsgemeinschaft within the funding programme Open Access Publishing, by the Baden-Württemberg Ministry of Science, Research and the Arts and by Ruprecht-Karls-Universität Heidelberg.

Acknowledgments

The authors wish to thank Simon Friedmann, Matthias Hock and Paul Müller. They contributed to developing the hardware and software that enabled this experiment.

References

- [1] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbrück, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic Silicon Neuron Circuits. *Frontiers in Neuroscience*, 5:73, 5 2011. ISSN 1662-4548. doi:10.3389/fnins.2011.00073. URL <http://journal.frontiersin.org/article/10.3389/fnins.2011.00073/abstract>.
- [2] Steve Furber. Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, 13(5):051001, 2016. doi:10.1088/1741-2560/13/5/051001. URL <http://stacks.iop.org/1741-2552/13/i=5/a=051001>.
- [3] Syed Ahmed Aamir, Yannik Stradmann, Paul Müller, Christian Pehle, Andreas Hartel, Andreas Grübl, Johannes Schemmel, and Karlheinz Meier. An Accelerated LIF Neuronal Network Array for a Large Scale Mixed-Signal Neuromorphic Architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–14, 2018. doi:10.1109/TCSI.2018.2840718. URL <https://ieeexplore.ieee.org/document/8398542/>.
- [4] Simon Friedmann, Johannes Schemmel, Andreas Grübl, Andreas Hartel, Matthias Hock, and Karlheinz Meier. Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System. *IEEE Transactions on Biomedical Circuits and Systems*, 11(1):128–142, 2 2017. doi:10.1109/TBCAS.2016.2579164. URL <http://ieeexplore.ieee.org/document/7563782/>.
- [5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2 2015. ISSN 0028-0836. doi:10.1038/nature14236. URL <http://www.nature.com/articles/nature14236>.

- [6] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016. doi:10.1038/nature16961. URL <https://pdfs.semanticscholar.org/1740/eb993cc8ca81f1e46ddaadce1f917e8000b5.pdf>.
- [7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354, 10 2017. URL <https://doi.org/10.1038/nature24270><https://www.nature.com/articles/nature24270#supplementary-information>.
- [8] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. MIT Press, 1998. ISBN 9780262193986. URL <https://mitpress.mit.edu/books/reinforcement-learning>.
- [9] Yael Niv. Reinforcement learning in the brain. *The Journal of Mathematical Psychology*, 53:139–154, 2009. doi:10.1016/j.jmp.2008.12.005. URL <https://www.princeton.edu/~yael/Publications/Niv2009.pdf>.
- [10] W Schultz, P Dayan, and P R Montague. A neural substrate of prediction and reward. *Science (New York, N.Y.)*, 275(5306):1593–9, 3 1997. ISSN 0036-8075. doi:10.1126/SCIENCE.275.5306.1593. URL <http://www.ncbi.nlm.nih.gov/pubmed/9054347>.
- [11] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules. *Frontiers in Neural Circuits*, 9:85, 2015. ISSN 1662-5110. doi:10.3389/fncir.2015.00085. URL <http://www.ncbi.nlm.nih.gov/pubmed/26834568><http://www.ncbi.nlm.nih.gov/article/PMC4717313>.
- [12] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons. *PLoS Computational Biology*, 9(4):e1003024, 4 2013. ISSN 1553-7358. doi:10.1371/journal.pcbi.1003024. URL <http://dx.plos.org/10.1371/journal.pcbi.1003024>.
- [13] Alexander Peyser, Ankur Sinha, Stine Brekke Vennemo, Tammo Ippen, Jakob Jordan, Steffen Gruber, Abigail Morrison, Guido Trensch, Tanguy Fardet, Håkon Mørk, Jan Hahne, Jannis Schuecker, Maximilian Schmidt, Susanne Kunkel, David Dahmen, Jochen Martin Eppler, Sandra Diaz, Dennis Terhorst, Rajalekshmi Deepu, Philipp Weidel, Itaru Kitayama, Sepehr Mahmoudian, David Kappel, Martin Schulze, Shailesh Appukuttan, Till Schumann, Hünkar Can Tunç, Jessica Mitchell, Michael Hoff, Eric Müller, Milena Menezes Carvalho, Barna Zajzon, and Hans Ekkehard Plesser. NEST 2.14.0. Zenodo, 10 2017. doi:10.5281/ZENODO.882971. URL <https://zenodo.org/record/882971>.
- [14] J Schemmel, D Brüderle, A Grübl, M Hock, K Meier, and S Millner. A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling. *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS’10)*, pages 1947–1950, 2010. doi:10.1109/ISCAS.2010.5536970.
- [15] Kai Zoschke, Maurice Güttler, Lars Böttcher, Andreas Grübl, Dan Husmann, Johannes Schemmel, Karlheinz Meier, and Oswin Ehrmann. Full Wafer Redistribution and Wafer Embedding as Key Technologies for a Multi-Scale Neuromorphic Hardware Cluster. *EPTC 2017*, 2017. doi:10.1109/EPTC.2017.8277579.
- [16] Syed Ahmed Aamir, Paul Müller, Andreas Hartel, Johannes Schemmel, and Karlheinz Meier. A highly tunable 65-nm CMOS LIF neuron for a large scale neuromorphic system. In *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 71–74. IEEE, 9 2016. ISBN 978-1-5090-2972-3. doi:10.1109/ESSCIRC.2016.7598245. URL <http://ieeexplore.ieee.org/document/7598245/>.
- [17] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience*, 9:141, 4 2015. ISSN 1662-453X. doi:10.3389/fnins.2015.00141. URL <http://journal.frontiersin.org/article/10.3389/fnins.2015.00141/abstract>.

- [18] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker Project. *Proceedings of the IEEE*, 102(5):652–665, 5 2014. ISSN 0018-9219. doi:10.1109/JPROC.2014.2304638. URL <http://ieeexplore.ieee.org/document/6750072/>.
- [19] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R. Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V. Arthur, Paul A. Merolla, and Kwabena Boahen. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE*, 102(5):699–716, 5 2014. ISSN 0018-9219. doi:10.1109/JPROC.2014.2313565. URL <http://ieeexplore.ieee.org/document/6805187/>.
- [20] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra S Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668, 8 2014. doi:10.1126/science.1254642. URL <http://science.sciencemag.org/content/345/6197/668.abstract>.
- [21] M Davies, N Srinivasa, T Lin, G Chinnya, Y Cao, S H Choday, G Dimou, P Joshi, N Imam, S Jain, Y Liao, C Lin, A Lines, R Liu, D Mathaiikutty, S McCoy, A Paul, J Tse, G Venkataraman, Y Weng, A Wild, Y Yang, and H Wang. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, 2018. ISSN 0272-1732. doi:10.1109/MM.2018.112130359.
- [22] Matthias Hock, Andreas Hartel, Johannes Schemmel, and Karlheinz Meier. An analog dynamic memory array for neuromorphic hardware. In *2013 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4. IEEE, 9 2013. ISBN 978-3-00-043785-4. doi:10.1109/ECCTD.2013.6662229. URL <http://ieeexplore.ieee.org/document/6662229/>.
- [23] Electronic Vision(s). <https://github.com/electronicvisions/gcc>, 2017.
- [24] Richard M. Stallman and GCC Developer Community. *GCC 8.0 GNU Compiler Collection Internals*. 12th Media Services, 2018. ISBN 9781680921878.
- [25] Norman Guttman. Operant conditioning, extinction, and periodic reinforcement in relation to concentration of sucrose used as reinforcing agent. *Journal of Experimental Psychology*, 46(4):213–224, 1953. ISSN 0022-1015. doi:10.1037/h0061893. URL <http://doi.apa.org/getdoi.cfm?doi=10.1037/h0061893>.
- [26] Chet T Moritz and Eberhard E Fetz. Volitional control of single cortical neurons in a brain-machine interface. *Journal of neural engineering*, 8(2):025017, 4 2011. ISSN 1741-2552. doi:10.1088/1741-2560/8/2/025017. URL <http://www.ncbi.nlm.nih.gov/pubmed/21436531><http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3156089/>.
- [27] E E Fetz and M A Baker. Operantly conditioned patterns on precentral unit activity and correlated responses in adjacent cells and contralateral muscles. *Journal of Neurophysiology*, 36(2):179–204, 3 1973. ISSN 0022-3077. doi:10.1152/jn.1973.36.2.179. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4196269/><http://www.physiology.org/doi/10.1152/jn.1973.36.2.179>.
- [28] E. M. Izhikevich. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex*, 17(10):2443–2452, 10 2007. ISSN 1047-3211. doi:10.1093/cercor/bhl152. URL <http://www.ncbi.nlm.nih.gov/pubmed/17220510><https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhl152>.
- [29] Michael A. Farries and Adrienne L. Fairhall. Reinforcement Learning With Modulated Spike Timing-Dependent Synaptic Plasticity. *Journal of Neurophysiology*, 98(6):3648–3665, 12 2007. ISSN 0022-3077. doi:10.1152/jn.00364.2007. URL <http://www.physiology.org/doi/10.1152/jn.00364.2007>.
- [30] N. Frémaux, H. Sprikeler, and W. Gerstner. Functional Requirements for Reward-Modulated Spike-Timing-Dependent Plasticity. *Journal of Neuroscience*, 30(40):13326–13337, 10 2010. ISSN 0270-6474. doi:10.1523/JNEUROSCI.6249-09.2010. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC20926659/><http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.6249-09.2010>.
- [31] Mantas Mikaitis, Garibaldi Pineda García, James C. Knight, and Steve B. Furber. Neuromodulated synaptic plasticity on the spinnaker neuromorphic system. *Frontiers in Neuroscience*, 12:105, 2018. ISSN 1662-453X. doi:10.3389/fnins.2018.00105. URL <https://www.frontiersin.org/article/10.3389/fnins.2018.00105>.

- [32] Simon Friedmann, Nicolas Frémaux, Johannes Schemmel, Wulfram Gerstner, and Karlheinz Meier. Reward-based learning under hardware constraints - Using a RISC processor embedded in a neuromorphic substrate. *Frontiers in Neuroscience*, pages 160. , 3 2013. doi:10.3389/fnins.2013.00160. URL <https://www.frontiersin.org/articles/10.3389/fnins.2013.00160/full>.
- [33] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLoS Computational Biology*, 4 (10):e1000180, 10 2008. ISSN 1553-7358. doi:10.1371/journal.pcbi.1000180. URL <http://dx.plos.org/10.1371/journal.pcbi.1000180>.
- [34] Wolfgang Maass. Noise as a Resource for Computation and Learning in Networks of Spiking Neurons. *Proceedings of the IEEE*, 102(5):860–880, 5 2014. doi:10.1109/JPROC.2014.2310593. URL <http://ieeexplore.ieee.org/document/6797856/>.
- [35] Xiaohui Xie and H. Sebastian Seung. Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, 69(4):041909, 4 2004. ISSN 1539-3755. doi:10.1103/PhysRevE.69.041909. URL <https://link.aps.org/doi/10.1103/PhysRevE.69.041909>.
- [36] Jeffrey R. Hollerman and Wolfram Schultz. Dopamine neurons report an error in the temporal prediction of reward during learning. *Nature Neuroscience*, 1(4):304–309, 8 1998. ISSN 1097-6256. doi:10.1038/1124. URL http://www.nature.com/articles/nn0898_304.
- [37] Hannah M. Bayer and Paul W. Glimcher. Midbrain Dopamine Neurons Encode a Quantitative Reward Prediction Error Signal. *Neuron*, 47(1):129–141, 7 2005. ISSN 0896-6273. doi:10.1016/J.NEURON.2005.05.020. URL <https://www.sciencedirect.com/science/article/pii/S0896627305004678>.
- [38] Verena Pawlak and Jason N D Kerr. Cellular/Molecular Dopamine Receptor Activation Is Required for Corticostriatal Spike-Timing-Dependent Plasticity. *The Journal of Neuroscience*, 2008. doi:10.1523/JNEUROSCI.4402-07.2008. URL <https://pdfs.semanticscholar.org/f072/62a58aa607abfeff0a040d98ad561342c5d1.pdf>.
- [39] Zuzanna Brzosko, Wolfram Schultz, and Ole Paulsen. Retroactive modulation of spike timing-dependent plasticity by dopamine. *eLife*, 4:e09685, 10 2015. ISSN 2050-084X. doi:10.7554/eLife.09685. URL <https://elifesciences.org/articles/09685>.
- [40] Elke Edelmann and Volkmar Lessmann. Dopamine Modulates Spike Timing-Dependent Plasticity and Action Potential Properties in CA1 Pyramidal Neurons of Acute Rat Hippocampal Slices. *Frontiers in Synaptic Neuroscience*, 3:6, 11 2011. ISSN 1663-3563. doi:10.3389/fnsyn.2011.00006. URL <http://journal.frontiersin.org/article/10.3389/fnsyn.2011.00006/abstract>.
- [41] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-Based Reinforcement Learning in Continuous State and Action Space: When Policy Gradient Methods Fail. *PLoS Computational Biology*, 5(12):e1000586, 12 2009. ISSN 1553-7358. doi:10.1371/journal.pcbi.1000586. URL <http://www.ncbi.nlm.nih.gov/pubmed/19997492http://www.ncbi.nlm.nih.gov/articlerender.fcgi?artid=PMC2778872http://dx.plos.org/10.1371/journal.pcbi.1000586>.
- [42] Gregor M. Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of Complex Computational Structures From Chaotic Neural Networks Through Reward-Modulated Hebbian Learning. *Cerebral Cortex*, 24(3):677–690, 3 2014. ISSN 1460-2199. doi:10.1093/cercor/bhs348. URL <http://www.ncbi.nlm.nih.gov/pubmed/23146969https://academic.oup.com/cercor/article-lookup/doi/10.1093/cercor/bhs348>.
- [43] M Mozafari, S R Kheradpisheh, T Masquelier, A Nowzari-Dalini, and M Ganjtabesh. First-Spike-Based Visual Categorization Using Reward-Modulated STDP. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 6 2018. ISSN 2162-237X. doi:10.1109/TNNLS.2018.2826721.
- [44] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J. Thorpe, and Timothée Masquelier. Combining STDP and Reward-Modulated STDP in Deep Convolutional Spiking Neural Networks for Digit Recognition. 3 2018. URL <http://arxiv.org/abs/1804.00227>.

- [45] Tim Head, MechCoder, Gilles Louuppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene-rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. scikit-optimize/scikit-optimize: v0.5.2. 3 2018. doi:10.5281/ZENODO.1207017. URL <https://zenodo.org/record/1207017>.
- [46] Timo Wunderlich. Neuromorphic R-STDP Experiment Simulation. <https://github.com/electronicvisions/model-sw-pong>, 2019.
- [47] Alain Destexhe, Michael Rudolph, and Denis Paré. The high-conductance state of neocortical neurons in vivo. *Nature Reviews Neuroscience*, 4(9):739–751, 9 2003. ISSN 1471-003X. doi:10.1038/nrn1198. URL <http://www.nature.com/articles/nrn1198>.
- [48] Mihai A Petrovici, Bernhard Vogginger, Paul Müller, Oliver Breitwieser, Mikael Lundqvist, Lyle Muller, Matthias Ehrlich, Alain Destexhe, Anders Lansner, René Schüffny, et al. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PloS one*, 9(10):e108590, 2014. doi:10.1371/journal.pone.0108590.
- [49] Mihai A. Petrovici, Johannes Bill, Ilja Bytschok, Johannes Schemmel, and Karlheinz Meier. Stochastic inference with spiking neurons in the high-conductance state. *Physical Review E*, 94(4):042312, 10 2016. ISSN 2470-0045. doi:10.1103/PhysRevE.94.042312. URL <https://link.aps.org/doi/10.1103/PhysRevE.94.042312>.
- [50] Jakob Jordan, Tammo Ippen, Moritz Helias, Itaru Kitayama, Mitsuhisa Sato, Jun Igarashi, Markus Diesmann, and Susanne Kunkel. Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics*, 12:2, 2 2018. ISSN 1662-5196. doi:10.3389/fninf.2018.00002. URL <https://www.frontiersin.org/article/10.3389/fninf.2018.00002/full>.
- [51] Sacha J. van Albada, Andrew G. Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B. Stokes, David R. Lester, Markus Diesmann, and Steve B. Furber. Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model. *Frontiers in Neuroscience*, 12:291, 5 2018. ISSN 1662-453X. doi:10.3389/fnins.2018.00291. URL <https://www.frontiersin.org/article/10.3389/fnins.2018.00291/full>.

Appendix

Symbol	Description	Value		
N	Neuromorphic hardware	BrainScaleS 2 (2nd prototype version)		
N_S	Number of action/output neurons (LIF)	32		
N_{syn}	Number of state/input units	32		
N_{spikes}	Number of synapses	$32 \cdot 32 = 1024$		
T_{ISI}	Number of spikes from input unit	20		
w	ISI of spikes from input unit	$10 \mu\text{s}$		
σ_w	Mean of distribution of initial weights (digital value)	14		
L	Standard deviation of distribution of initial weights	2		
$\ \vec{v}\ _1$	Length and width of quadratic playing field	1		
v_p	L1-norm of ball velocity	0.025 per iteration		
r_b	Velocity of paddle controlled by BSS2	0.05 per iteration		
r_p	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 μs	18.4 μs	24.8 μs
τ_{ref}	LIF refractory time constant	4 μs	14.3 μs	13.8 μs
τ_{syn}	LIF excitatory synaptic time constant	1.8 μs	2.4 μs	1.4 μs
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 μs	80 μs	60 μs

Table 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.