Dissertation

submitted to the Combined Faculty of Natural Sciences and Mathematics
of Heidelberg University, Germany
for the degree of

Doctor of Natural Sciences

Put forward by

Ákos Ferenc Kungl

born in: Veszprém, Hungary

Oral examination: 2020 May 29th

Robust learning algorithms for spiking and rate-based neural networks

Referees:

Prof. Dr. Walter Senn (University of Bern)

Prof. Dr. Ulrich Schwarz (Heidelberg University)

Robuste Lernalgorithmen für spikende und ratenbasierte neuronale Netzwerke

Inspiriert von den herausragenden Eigenschaften des menschlichen Gehirns haben die Bereiche maschinelles Lernen, computergestützte Neurowissenschaften und neuromorphes Rechnen im letzten Jahrzehnt signifikante synergistische Fortschritte erzielt. Leistungsstarke neuronale Netzwerkmodelle, die auf maschinellem Lernen basieren, wurden als Modelle für die Neurowissenschaften und für die Anwendung in der neuromorphen Elektronik vorgeschlagen. Jedoch wird der Aspekt der Robustheit bei diesen Modellen häufig vernachlässigt. Sowohl biologische als auch technische Substrate weisen verschiedene Mängel auf, die die Leistung von Rechenmodellen beeinträchtigen oder deren Implementierung sogar verhindern. Diese Arbeit beschreibt drei Projekte, die darauf abzielen, robustes Lernen mit lokalen Lernregeln in neuronalen Netzen zu implementieren. Zuerst präsentieren wir eine Implementierung der spike-basierten[a] Bayes'schen Inferenz auf beschleunigter neuromorpher Hardware. Durch Lernen schafft das Model die störenden Auswirkungen der unpräzisen Hardware zu kompensieren, während es von der Beschleunigung der Hardware profitiert. Zweitens zeigen wir die Vorteile neuromorpher Berechnungen in einer Pilotstudie an einem Prototyp-Chip. Dabei quantifizieren wir die Geschwindigkeit und den Energieverbrauch des Systems im Vergleich zu einer Softwaresimulation und zeigen, wie On-Chip-Lernen zur Robustheit des Lernens beiträgt. Schließlich präsentieren wir ein robustes Modell für tiefes bestärkendes Lernen unter Verwendung lokaler Lernregeln. Es zeigt, wie Backpropagation[b] in Kombination mit Neuromodulation in einem biologisch plausiblen Rahmen implementiert werden kann. Die Ergebnisse tragen zur Entwicklung robuster und leistungsfähiger Lernnetzwerke für biologische und neuromorphe Substrate bei.

[a]Insbesondere in zusammengesetzten Ausdrücken werden Aktionspotentiale nach dem Englischen häufig Spikes genannt.

[b]Backpropagation wird selten als Fehlerrückführung übersetzt.

Robust learning algorithms for spiking and rate-based neural networks

Inspired by the remarkable properties of the human brain, the fields of machine learning, computational neuroscience and neuromorphic engineering have achieved significant synergistic progress in the last decade. Powerful neural network models rooted in machine learning have been proposed as models for neuroscience and for applications in neuromorphic engineering. However, the aspect of robustness is often neglected in these models. Both biological and engineered substrates show diverse imperfections that deteriorate the performance of computation models or even prohibit their implementation. This thesis describes three projects aiming at implementing robust learning with local plasticity rules in neural networks. First, we demonstrate the advantages of neuromorphic computations in a pilot study on a prototype chip. Thereby, we quantify the speed and energy consumption of the system compared to a software simulation and show how on-chip learning contributes to the robustness of learning. Second, we present an implementation of spike-based Bayesian inference on accelerated neuromorphic hardware. The model copes, via learning, with the disruptive effects of the imperfect substrate and benefits from the acceleration. Finally, we present a robust model of deep reinforcement learning using local learning rules. It shows how backpropagation combined with neuromodulation could be implemented in a biologically plausible framework. The results contribute to the pursuit of robust and powerful learning networks for biological and neuromorphic substrates.

Contents

1 Introduction: at the intersection between three worlds

The remarkable properties of the human brain — its highly parallel computation, its low power consumption, its learning capabilities, its ability to cope with noisy or incomplete observations, its ability to learn and generalize from a few examples and its robustness to injuries [Kety, 1957, Levin et al., 1987] — have inspired research in different fields. In the last few decades a tight synergy emerged between the fields of computational neuroscience — the study of computational principles governing the nervous system —, machine learning — the science to give a computer the ability of solving problems without explicitly programming them — and neuromorphic engineering — the endeavor to create novel fast and energy efficient hardware architectures inspired by the brain (figure 1.1). There is a continuous exchange of not only results and ideas but sometimes even researchers, which lead to prominent advancements in technology, such as today's deep learning boom [LeCun et al., 2015, Vinyals et al., 2019].

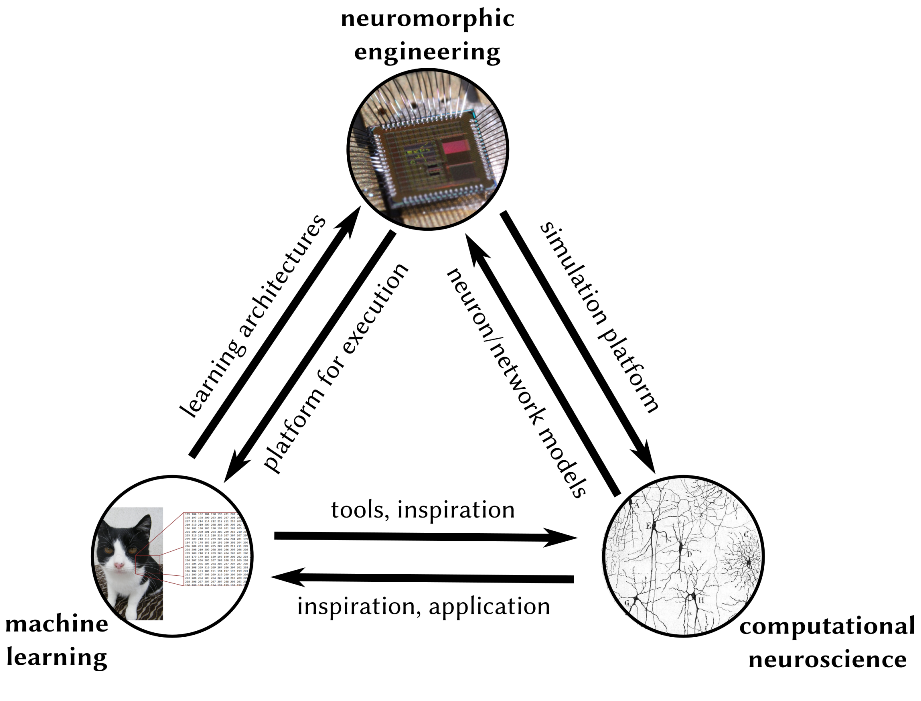


Figure 1.1: The synergy between machine learning, computational neuroscience and neuromorphic engineering. In the last decades, tight connections have been established between these three fields continuously influencing each-other.

The balanced relationship suggested in figure 1.1 among the three fields is less symmetric in reality. Neuromorphic engineering is dwarfed by the other two fields in terms of invested research, meaning people and funding, as well as probably in terms of immediate relevance for society. However neuromorphic engineering is a much younger field, with initial ideas from the 1980s [Mead, 1989]. It uses results and takes inspiration from the two other fields, but the backward direction is, at least for now, rather exploratory than actually established.

The relation between computational neuroscience and machine learning is the most apparent one. Some of the famous results in machine learning are based on ideas from neuroscience. For example, artificial neural networks are loosely based on the structure of the human visual cortex [McCulloch and Pitts, 1943, Rosenblatt, 1962, Rumelhart et al., 1986, LeCun et al., 2015]; and "experience replay" — a learning strategy that contributed to the recent super-human results in playing computer games [Vinyals et al., 2019, Rolnick et al., 2019] — was inspired by the memory-replay in the hippo-campus [Foster, 2017]. Conversely, machine learning provides tools for neuroscience research, such as reconstructing the connections between neurons from microscopy data or behavior tracking and segmentation [Hinton, 2011, Helmstaedter, 2015, Arganda-Carreras et al., 2017, Vu et al., 2018]. Furthermore, algorithms in machine learning inspire models on how the brain might implement information processing, with one particular example being the recent search for deep learning in the nervous system [Sacramento et al., 2018, Whittington and Bogacz, 2019, Richards et al., 2019, Senn et al., in preparation].

Most of the used neuron models, coding schemes and plasticity rules in neuromorphic engineering are rooted in computational neuroscience [Indiveri et al., 2011, Vanarse et al., 2016, Schuman et al., 2017]. The other way around, some of the neuromorphic systems aim to provide alternative simulation platforms for long time and large-scale neural network simulations, which would be infeasible or not affordable on conventional supercomputers [Schemmel et al., 2010, Furber et al., 2014]. Furthermore, neuromorphic engineering explores and tests ideas from computational neuroscience on the functional level, that is in application. This effort might inspire new ideas for modeling in neuroscience [Jordan et al., 2019, Dold et al., 2019].

The interaction between machine learning and neuromorphic engineering became stronger since the renaissance of artificial neural networks at around 2012 [Ciregan et al., 2012, Krizhevsky et al., 2012], but it is still asymmetric. On the one hand, neuromorphic realizations/implementations of network models and learning rules are are rooted in models from the field of machine learning [Esser et al., 2016, Schmitt et al., 2017, Kungl et al., 2019], or the implementations use machine learning as a tool to optimize the setup [Bohnstingl et al., 2019]. On the other hand, some neuromorphic hardware is developed with the aim to create fast and energy efficient substrates for the execution of machine learning models [Davies et al., 2018, Chen et al., 2019]. In the context of solving real-world tasks, two main approaches exist: neuromorphic engineering could create specialized hardware for existing machine learning models or it could follow an explorative development based on neuroscience-inspired solutions. At the moment, both approaches exist and are pursued actively [Thakur et al., 2018].

An often overlooked aspect in this synergistic advancement of science is the robustness of the proposed models both for neuromorphic engineering and for computational neuroscience. In the nervous system and on neuromorphic substrates, network and neuron models have to cope with constraints on the network structure and on the precision of the parameters as well as with the omnipresent temporal variations. For example, in biology each neuron is different to a certain extent due to variations during development and due to the stochastic nature of the processes, for example diffusion. Furthermore, especially on analog neuromorphic hardware, imperfections of the manufacturing process introduce differences between the single circuits, similar to differences between individual neurons in biology. Hence, studying the robustness of models is a central question if we want to verify the biological plausibility of models, or if we want to implement them on neuromorphic hardware. On the one hand, developing models that show robustness to the aforementioned effects would increase the biological plausibility of the models, on the other hand, robust models could mitigate the disrupting effects on neuromorphic hardware and increase the usability of these systems.

This thesis is centered around three projects contributing to this synergistic development. We focus on neuromorphic engineering and computational neuroscience, to which we aim to contribute; while machine learning serves as a tool and as a source of inspiration.

First, we present the implementation of sampling-based Bayesian computation with spiking neural networks (chapter 3) on the mixed-signal neuromorphic BrainScaleS-1 system (BSS-1). The project exemplifies, how an algorithm originally from machine learning is developed and adapted for neuromorphic hardware. We show that the probabilistic model combined with a local learning rule compensates for distorting effects on the neuromorphic substrate and benefits from the fast computation.

Second, we conduct a pilot-study characterizing the advantages of neuromorphic computation (chapter 4) using the example of a reinforcement learning algorithm — that is, learning from interaction with the environment — on a prototype chip of the BrainScaleS-2 system (BSS-2). The project quantifies these advantages in terms of execution speed and energy consumption in comparison to simulations on a conventional CPU. The study demonstrates that the learning rule can adapt to substrate imperfections and, furthermore, that the results of learning can be transferred between chips in spite of variations between realizations of the same chip version.

In the third project, we present a biologically plausible deep reinforcement learning rule derived in a top-down manner using the principle of least action (chapter 5). The implementation relies on mechanisms inspired by experimental observations, such as predictive firing of neurons, stereotypical local circuitry and winner-takes-all type interaction. The model turns out to be robust against time-delayed reward and imperfect parameters, which both increase its biological plausibility, and contributes to an envisioned neuromorphic implementation.

The structure of the thesis

The thesis is structured into these projects, in such a way that each project can be read individually as a stand-alone project in combination with the background chapter (chapter 2). We present the three projects such that they could be understood as stand-alone works or pieces of the introduced synergistic development. Accordingly, each of the three projects has its own introduction and each project is discussed in great details at the end of the respective chapter. The three chapters represent three different publications each.

In chapter 2, we introduce the necessary background for the thesis including more detailed background information on the current state and ideas of machine learning, computational neuroscience and neuromorphic engineering. In chapters 3 to 5, we describe and discuss the three aforementioned projects. Finally, in chapter 6, we discuss the potential significance of the three projects in relation to each other and in perspective of machine learning, computational neuroscience and neuromorphic engineering.

A list of publications and description of my contribution is given in appendix C.

2 Background

In this chapter, we give a brief introduction to the closely related fields of machine learning, computational neuroscience and neuromorphic engineering. The aim is to present the necessary basics for the three described projects in this thesis, and to sketch the place of this work on the broader scientific landscape. We put more emphasis on neuromorphic engineering, because this field, unlike the other two, lacks standard and established textbooks, and it is in general more explorative and less familiar to the general scientific public. For all three fields, we highlight recent review publications and textbooks when available.

2.1 Basics and terminology of machine learning

Machine learning is the field of research that attempts to give computers the ability to learn and solve tasks without explicitly programming them [Samuel, 1959]. In machine learning, we aim to create models that can solve practical tasks not by being told how to solve the given task, but by learning from lots of examples.[1] For instance, a machine learning model whose task is to recognize faces in images, would learn from a large dataset containing images with and without faces. Machine learning has a broad application field stretching — without claim to completeness — from automatic annotation of images [Karpathy and Fei-Fei, 2015] to providing tools for research in physics [Carleo et al., 2019a].

Each machine learning model consists of three main parts: the representation, the evaluation and the optimization [Domingos, 2012]. The representation is a mathematical model describing the mechanism of the machine learning algorithm. The mathematical model has to be both powerful enough to be able to represent the problem and at the same time accessible (mathematically tractable) for efficient optimization. Evaluation is a performance measure of the machine learning model. Often the learning (optimization) is not based on the same measure as the final performance evaluation (for a detailed example see section 2.1.1). Finally, optimization is how the algorithm extracts its optimal parameters from the available datasets. It is often referred to as the training or learning procedure.

A key aspect of machine learning is the requirement for generalization. The machine learning model is expected to perform well on not only the dataset used during optimization, but also on previously unseen examples. In practice, standard benchmark datasets consist of a separate set of training data for optimization and a test set for testing the model. In machine learning competitions — which

[1]There is a subfield of machine learning called one-shot learning, which studies learning from a few examples and from a few repetitions, see for example Fei-Fei et al. [2006].

are important drivers of machine learning research — , the test set is often hidden from the participants to avoid optimization for the test set.

In the following, we introduce the basic concepts and terminology of machine learning. We describe the three main frameworks of learning, each with an example. For a practical introduction to machine learning we refer to Mehta et al. [2019] and for a detailed standard textbook to Bishop [2006].

2.1.1 Supervised learning and artificial neural networks

The aim of supervised learning is to learn the input-output relation in a dataset $(\boldsymbol{X}, \boldsymbol{Y})$ consisting of pairs of input and output data $(\mathbf{x}_i, \mathbf{y}_i)$, with $\boldsymbol{x}_i \in \mathbb{R}^{N_\mathrm{x}} \forall i$ and $\boldsymbol{y}_i \in \mathbb{R}^{N_\mathrm{y}} \forall i$. We let the cardinality of the dataset be defined as $|\boldsymbol{X}| = |\boldsymbol{Y}| = N_\mathrm{d}$. Here, $\mathbf{x}_i$ is often referred to as the input data or feature vector and $\mathbf{y}_i$ as the output or label. In this regard, supervised learning is similar to function approximation, where we believe that the dataset implicitly defines the functional dependency we want to represent with the model. Let function F represent the machine learning model. We say that for an input data $\mathbf{x}_i$ it gives the predicted label $\mathbf{y}_i^\mathrm{pred} = F(\mathbf{x}_i)$.

A typical example of supervised learning is image classification. The algorithm should be able to recognize the type of the object seen in the picture. In multi-class classification, the label is coded in the one-hot coding scheme $\mathbf{y}_i \in \{(1,0,\ldots,0);(0,1,0,\ldots,0);(0,0,\ldots,0,1)\}$, where exactly the n-th element equals 1 when coding the n-th class. The prediction of the model is then the class with the highest value: argmax $(F(\mathbf{x}))$. The typical (but not only) loss function in classification is the Euclidean loss, also called the L^2 loss function,

$$L(\boldsymbol{X}, \boldsymbol{Y}; F) = \frac{1}{2} \sum_{i=1}^{N_\mathrm{d}} \|\mathbf{y}_i - F(\mathbf{x}_i)\|^2 \quad , \tag{2.1}$$

where we sum over the complete dataset and $\|\cdot\|$ means the standard Euclidean norm. The aim of the optimization procedure is to find a set of parameters that minimizes the loss function. The difference between the internal and the external evaluation immediately becomes apparent. Internally, we use the $L(\boldsymbol{X}, \boldsymbol{Y}; F)$ loss function to learn the parameters of the model, but externally, the performance of the model is assessed as the classification ratio on the test set.

Supervised learning is also a good example to understand underfitting and overfitting in machine learning (figure 2.1). Underfitting — also called high-bias problem — happens when the underlying model is unable to represent the relation between the input data and the label. Performance is equally low both on the training set and the test set. Further refinements on the optimization procedure do not mitigate the high-bias problem. In case of overfitting or a high-variance model, the model is flexible enough to represent the training set but it does not generalize well to unseen examples. The model adapts to noise in the training set;

the loss function can be minimized well over the training set, but the model fails to give good predictions on the test set.[2]

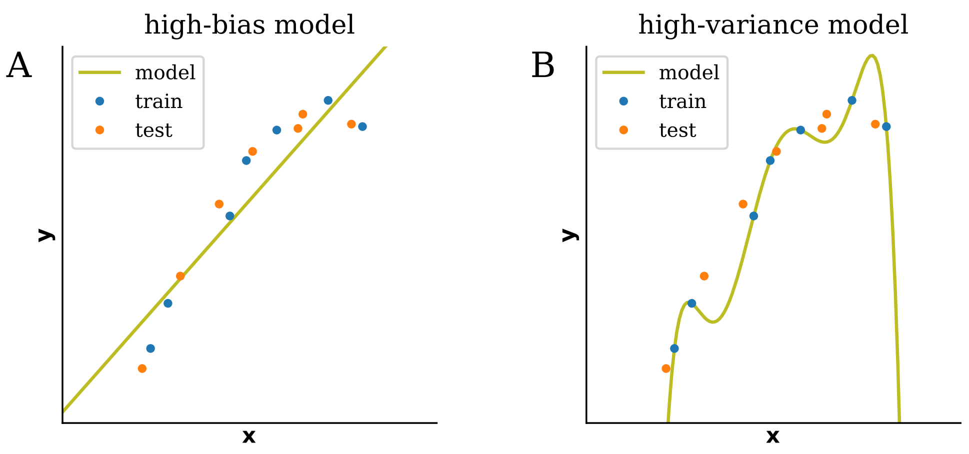


Figure 2.1: Example of under- and overfitting. The data suggests that the relationship between the input and output is approximately a second order polynomial up to some noise. **(A)** Fitting a linear function is a high-bias model (underfitting). It is unable to represent the second order dependency. The model similarly fails both on the training set and the test set. **(B)** Choosing a 6-th order polynomial is a high-variance model (overfitting). The model can represent the second order dependence, but it also goes beyond and adapts to the noise in the training set. Hence, the loss function can be minimized well over the training set but the model fails to provide good prediction for the test set.

Feed-forward abstract neural networks and the backpropagation algorithm

Artificial neural networks are highly non-linear machine learning models loosely inspired by the structure of the visual processing pathway in the brain [McCulloch and Pitts, 1943, Rosenblatt, 1962, Ivakhnenko, 1971, Rumelhart et al., 1986].[3] Here, we only treat the most basic type of neural networks: the feed-forward neural network. For advanced neural networks in machine learning see LeCun et al. [2015] and Goodfellow et al. [2016].

The basic building block of abstract neural networks are abstract neurons. Abstract neurons perform a weighted sum on their inputs s_i and apply a non-linear function to the results:

$$a = f\left(\sum_{i=1}^{N} w_i s_i\right) \quad . \tag{2.2}$$

Here, a is the activation or output of the given neuron, we sum over the inputs from other neurons (pre-synaptic partners, compare to section 2.2.1) and w_i is the

[2]For the mathematical treatment of the bias-variance trade-off see bias-variance decomposition in Bishop [2006].

[3]There is an on-going debate about who and when conceptualized neural networks first.

weight of the incoming input. The function f denotes the activation function of the neuron, that is the input-output function of the neuron. Early models used the logistic activation function $f^{\log}(u) = 1/(1+\exp(-u))$, but nowadays most models use the rectified linear unit (ReLu) $f^{\text{relu}}(u) = \max(0; u)$. The rectified linear unit (ReLu) activation function became popular due to its fast and cheap computability and due to the non-vanishing gradient for $u > 0$. The abstract neuron model lacks any internal dynamics and only models the input-output relation. In neuroscience literature, these are sometimes referred to as McCulloch-Pitts type neurons [McCulloch and Pitts, 1943] or neuron models of the first generation [Maass, 1997].

Neural networks consist of several layers of neurons, performing consecutive non-linear operations. The first layer is called input layer (figure 2.2 A). This is a virtual layer, it represents a sample $\mathbf{x}_i$ of the input data and not neurons with activations. The last layer is called label layer. Its value is the predicted label $\mathbf{y}_i^{\text{pred}}$ of the model. The layers in between are called hidden layers; $z_k^{(l)}$ denotes the activity of the k-th neuron in the l-th hidden layer. In a network with N layers, the information processing goes through the network iteratively as

$$\begin{aligned}
\mathbf{z}^{(1)} &= f\left(\mathbf{u}^{(1)}\right) \text{ with } u_i^{(1)} = \sum_k w_{ik}^{(1)} x_k \quad , \\
\mathbf{z}^{(l)} &= f\left(\mathbf{u}^{(l)}\right) \text{ with } u_i^{(l)} = \sum_k w_{ik}^{(l)} z_k^{(l-1)} \text{ for } l \in \{2, \ldots, N-1\} \quad , \\
\mathbf{y}^{\text{pred}} &= f\left(\mathbf{u}^{(N)}\right) \text{ with } u_i^{(N)} = \sum_k w_{ik}^{(l)} z_k^{(N-1)} \quad .
\end{aligned} \tag{2.3}$$

Here, we choose the symbol $\mathbf{u}^{(N)}$ for the weighted sum of the inputs, to emphasize its (loose) relation to neuronal membrane potentials (see section 2.2.2). We defined $\mathbf{W}^{(l)}$ as the weight matrix projecting from the $(l-1)$-th and to the l-th layer. The large number of consecutive non-linearities and parallel processing through wide layers make neural networks powerful models which can represent a wide range of tasks. Leshno et al. [1993] have even shown that feed-forward neural networks with non-polynomial activation functions can approximate any continuous function given a sufficient number of hidden units. This is called the universal approximation theorem.

Neural networks are trained via some form of gradient descent. The introduced loss function (equation (2.1)) is now parameterized by the weights of the neural network: $L(\boldsymbol{X}, \boldsymbol{Y}; \mathbf{W})$, where $\mathbf{W}$ symbolizes all weights of the network. The parameters are iteratively updated based on gradient descent with the formula

$$w_{ik}^{(l)} \to w_{ik}^{(l)} - \eta \frac{\partial L}{\partial w_{ik}^{(l)}} \quad , \tag{2.4}$$

where η is the learning rate. For a sufficiently small learning rate, gradient descent converges to a local minimum of the loss function. The derivative term contains a sum over the complete dataset, which is expensive (memory and computation) to

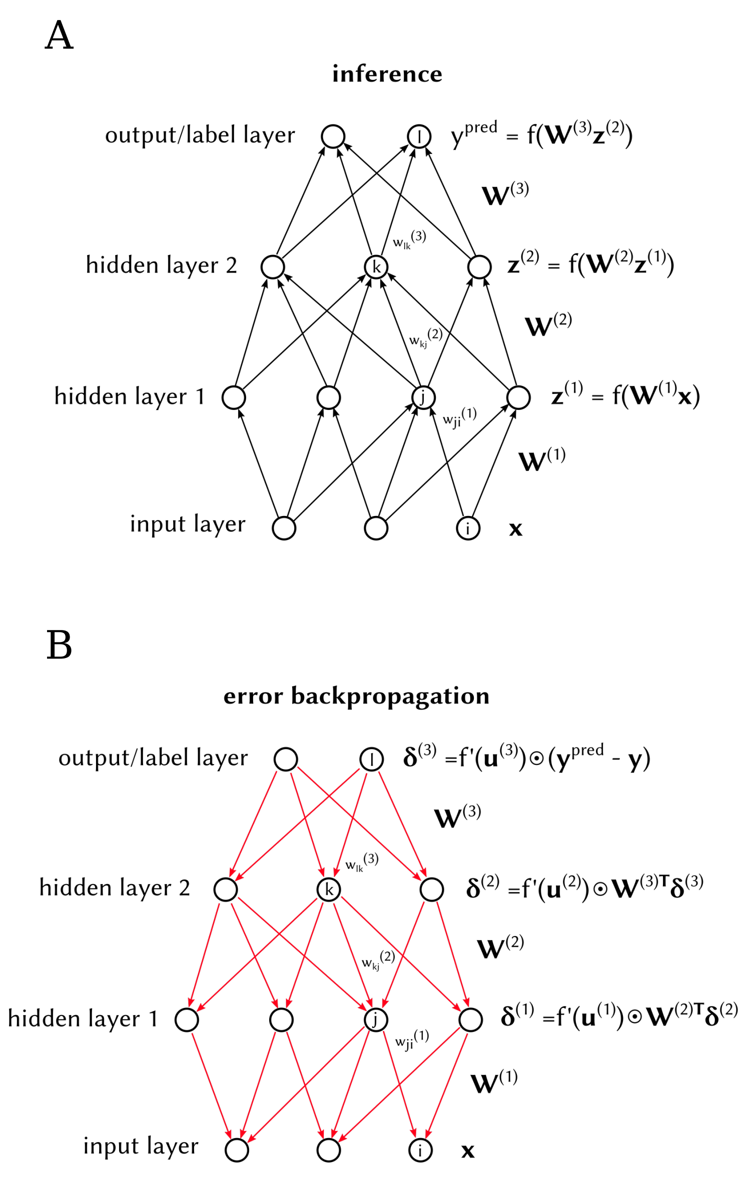

A
inference
output/label layer
l
$y^{pred} = f(\mathbf{W}^{(3)}\mathbf{z}^{(2)})$
$\mathbf{W}^{(3)}$
$w_{lk}^{(3)}$
hidden layer 2
k
$\mathbf{z}^{(2)} = f(\mathbf{W}^{(2)}\mathbf{z}^{(1)})$
$\mathbf{W}^{(2)}$
$w_{kj}^{(2)}$
hidden layer 1
j
$w_{ji}^{(1)}$
$\mathbf{z}^{(1)} = f(\mathbf{W}^{(1)}\mathbf{x})$
$\mathbf{W}^{(1)}$
input layer
i
$\mathbf{x}$
B
error backpropagation
output/label layer
l
$\delta^{(3)} = f'(\mathbf{u}^{(3)}) \odot (\mathbf{y}^{pred} - \mathbf{y})$
$\mathbf{W}^{(3)}$
$w_{lk}^{(3)}$
hidden layer 2
k
$\delta^{(2)} = f'(\mathbf{u}^{(2)}) \odot \mathbf{W}^{(3)T}\delta^{(3)}$
$\mathbf{W}^{(2)}$
$w_{kj}^{(2)}$
hidden layer 1
j
$w_{ji}^{(1)}$
$\delta^{(1)} = f'(\mathbf{u}^{(1)}) \odot \mathbf{W}^{(2)T}\delta^{(2)}$
$\mathbf{W}^{(1)}$
input layer
i
$\mathbf{x}$

Figure 2.2: Inference and error backpropagation in a feed-forward neural network. (A) In the feed-forward (inference) direction, the information is processed over several layers of neurons. In each layer, a weighted sum and a non-linear operation is applied. The several layers of non-linearities and the flexible parametrization enables the neural network to represent a broad range of models. **(B)** In the error backpropagation direction, the information flows from the label layer backwards towards the output layer. The easy computability of the gradient according to the network parameters enables fast learning. For the definition of the variables see the main text. Figure adapted from LeCun et al. [2015].

calculate. In practice, the derivative is only calculated on a subset of the dataset or even on a single example, and the subset is randomly chosen in each iteration. This procedure is known as mini-batch learning or stochastic gradient descent. Machine learning is not consistent about the terminology: Some authors define stochastic gradient descent strictly as applying parameter updates based on a single pair of input and output data. Others call any update scheme stochastic gradient descent which does not use the entire dataset for a single parameter update. In the second picture, mini-batch learning is a subset of stochastic gradient descent. If the derivative is calculated on the complete dataset, it is called batch learning. Batch learning has mostly no practical relevance because modern datasets have grown too large, hence calculating the gradient on the entire dataset in each iteration would be impractically slow.

Abstract neural networks are mainly trained via the backpropagation algorithm that is the application gradient descent to neural networks [Rumelhart et al., 1986]. We can derive the backpropagation formula by applying the chain rule to the derivative term in equation (2.4). For a pair of input and label data $(\mathbf{x}, \mathbf{y})$, we obtain the recursive formula

$$\begin{aligned} \delta^{(N)} &= f'\left(\boldsymbol{u}^{(N)}\right) \odot \left(\boldsymbol{y}^{\text{pred}} - \boldsymbol{y}\right) \quad , \\ \delta^{(l)} &= f'\left(\boldsymbol{u}^{(l)}\right) \odot \boldsymbol{W}^{(l+1)^T} \delta^{(l+1)} \text{ for } l \in \{1, \ldots, N-1\} \quad , \\ \frac{\partial L^{\text{single}}}{\partial \boldsymbol{W}^{(l)}} &= \delta^{(l)} f^T \left(\boldsymbol{u}^{(l-1)}\right) \text{ for } l \in [1, \ldots, N-1] \quad . \end{aligned} \tag{2.5}$$

Here, $\odot$ denotes the element-wise product and $\delta^{(l)}$ is the associated error-vector in the l-th layer. This time $L^{\text{single}} = \frac{1}{2}\left\|\boldsymbol{y}^{\text{pred}} - \boldsymbol{y}\right\|^2$ is the loss function taken over the single input output pair. The recursive form of the loss function implies that the error-vector propagates back through the network from the output to the input layer (figure 2.2 B). Although, backpropagation is merely the chain rule applied to the neural network structure and it has been already introduced in the 1980s the latest, it took until the development of affordable and fast Graphical Processing Units (GPUs) and the availability of large datasets that deep learning could become the state of the art [LeCun et al., 2015].

Deep learning is the collective name for the end-to-end training of models where the information processing goes through several non-linear processing steps, for example several layers in a feed-forward neural network. Here, end-to-end refers to the direct backpropagation of the gradients over the non-linearities. Each of these steps is parametrized separately, for example via the weight matrices between the layers. In contrast to deep learning, shallow models only learn the linear features from the training set, for example a neural network without hidden layers is a shallow learner.[4]

[4]Note, shallow learning models are usually extended by hand-engineered features, so-called kernels. For more on these methods see Bishop [2006].

2.1.2 Unsupervised learning and probabilistic generative models

In unsupervised learning, we deal with an unlabeled dataset S containing datavectors $\boldsymbol{x}_i$. Naturally, labeled datasets can be represented as unlabeled datasets by incorporating the label, or output, into the feature vector $\boldsymbol{x}_i$. The aim of unsupervised learning is to find structure in the dataset and to represent it in an accessible manner. Often, but not always, unsupervised learning implicitly means capturing the structure in the dataset with a lower dimensional representation (dimensionality reduction), while other popular applications are clustering and outlier detection.

We introduce unsupervised learning using the example of a general probabilistic generative model. In these models, we assume that the data stems from an underlying probability distribution $p^*(\boldsymbol{x})$. The dataset S is thought of as the sampled representation of this distribution:

$$p^*(\boldsymbol{x}) = \frac{1}{|S|} \sum_{\boldsymbol{x}_s \in S} \delta\left(\boldsymbol{x} - \boldsymbol{x}_s\right) \quad . \tag{2.6}$$

However, we keep in mind that the dataset is a finite sampled representation, and hence overfitting and underfitting issues can occur, just as in the case of supervised learning. For the model, we consider a probability distribution $p(\boldsymbol{x};\boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$.

During the learning procedure, we look for parameters $\boldsymbol{\theta}^*$ such that the model's probability distribution $p(\boldsymbol{x};\boldsymbol{\theta}^*)$ accurately approximates the probability distribution of the dataset $p^*(\boldsymbol{x})$. This is usually formulated with maximum log-likelihood learning

$$\boldsymbol{\theta}^* = \mathrm{argmax}_{\boldsymbol{\theta}} \left(\sum_{\boldsymbol{x}_s \in S} \log\left(p(\boldsymbol{x}_s;\boldsymbol{\theta})\right) \right) \quad . \tag{2.7}$$

This formula requires that the log-likelihood of the examples from the dataset should be maximized in the learned model. That is, the learned model should be able to generate samples from the learned dataset. Log-likelihood learning is equivalent to minimizing the Kullback-Leibler divergence between the distribution of the data and the learned model.

The Kullback-Leibler Divergence (D_{KL}) is a measure of the discrepancy between two probability distributions q_1 and q_2 defined as [Kullback and Leibler, 1951]:

$$D_{\mathrm{KL}}(q_1||q_2) = \sum_{x \in \boldsymbol{\Omega}} q_1(x) \ln\left(\frac{q_1(x)}{q_2(x)}\right) \quad . \tag{2.8}$$

The sum — and for continuous distributions the integral — runs over the entire state-space Ω. Depending on the context, the D_{KL} has several interpretations. In their original publication, Kullback and Leibler [1951] defined it as the average information over the states x to discriminate between q_1 and q_2 when using q_1 for the observation. Note, that the D_{KL} is not a real distance measure although we use it as a distance indicator. In particular, it not symmetric in general:

$$D_{\mathrm{KL}}(q_1||q_2) \neq D_{\mathrm{KL}}(q_2||q_1) \quad . \tag{2.9}$$

The base of the logarithm is *a priori* arbitrary. A popular choice is $\log_2$ to comply with the interpretation as information and to measure the discrepancy in bits.

For demonstrating the relation between the D_{KL} and maximum log-likelihood learning, we consider (omitting $\boldsymbol{\theta}$ for simplicity)

$$\begin{aligned} D_{\mathrm{KL}}(p^*||p) &= \sum_{\boldsymbol{x}\in S} \log\left(\frac{p^*(\boldsymbol{x}_s)}{p(\boldsymbol{x}_s)}\right) p^*(\boldsymbol{x}_s) = \\ &= \underbrace{\sum_{\boldsymbol{x}\in S} \log\left(p^*(\boldsymbol{x}_s)\right) p^*(\boldsymbol{x}_s)}_{-H(p^*)} - \underbrace{\sum_{\boldsymbol{x}\in S} \log\left(p(\boldsymbol{x}_s)\right) p^*(\boldsymbol{x}_s)}_{\text{log-likelihood}} \quad . \end{aligned} \tag{2.10}$$

We can recognize the Shannon entropy [Shannon, 1948] of the dataset $H(p^*)$ in the first term, which is independent of the model p. In the second term, we can recognize the log-likelihood. Hence, minimizing the D_{KL} between the dataset and the model distribution is equivalent to maximizing the log-likelihood.

It is especially interesting that unsupervised models can also solve other tasks such as pattern completion and classification if the conditional distributions are numerically or analytically accessible (see also chapter 3). For these cases, we split the datavector $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ and regard the $\boldsymbol{x}_2$ as the label of the data. In inference, the probability distribution of the label is given by the conditional probability $p(\boldsymbol{x}_2|\boldsymbol{x}_1;\boldsymbol{\theta})$.

2.1.3 Task-focused definition of reinforcement learning

Reinforcement learning is goal-directed learning via interaction with the environment [Sutton and Barto, 2018]. The definition of reinforcement learning is hence focused on the task and not on the method. An agent seeks to maximize the received reward while actively interacting with its environment. The reward feedback can be sparse in time, and a chain of actions might separate the causal action leading to the reward. A classic example is that of a chess player who interacts with his/her opponent throughout the game and only receives a reward feedback in the end when the game is either lost or won. Events like losing or capturing pieces are not considered as external reward signals.

Formally, the components of reinforcement learning are an agent and the environment (figure 2.3 A). We call the state of the environment S_t at time-step t, where we implicitly use discrete time. Any circumstances that are not part of the decision-making policy of the agent, are called the environment. For example, in case of a robot, the power level of the battery would be conceptually considered as part of the environment, although physically it is part of the robot.

Depending on the current state, the environment provides the agent a scalar-valued reward signal $R_t = R(S_t)$. The agent takes actions A_t based on the current state of the environment. Formally, the policy function $\pi_{\text{policy}}(A_t; S_t)$ describes the probability of each action depending on the state of the environment. The environment reacts to the actions of the agent by transitioning to a next state S_{t+1} and providing reward R_{t+1}. In general, the state transition of the environment can be deterministic or probabilistic and can be written as $\pi_{\text{env}}\left(S_{t+1}|S_t, A_t\right)$. Here, we implicitly assumed the Markovian property for the environment, the next state S_{t+1} only depends on the previous state and action. Reinforcement learning in non-Markovian or only partially observable environments is a subfield of reinforcement learning research, see for example Spaan [2012]. The aim of the agent is to find the policy that maximizes the gathered reward along the interactions.

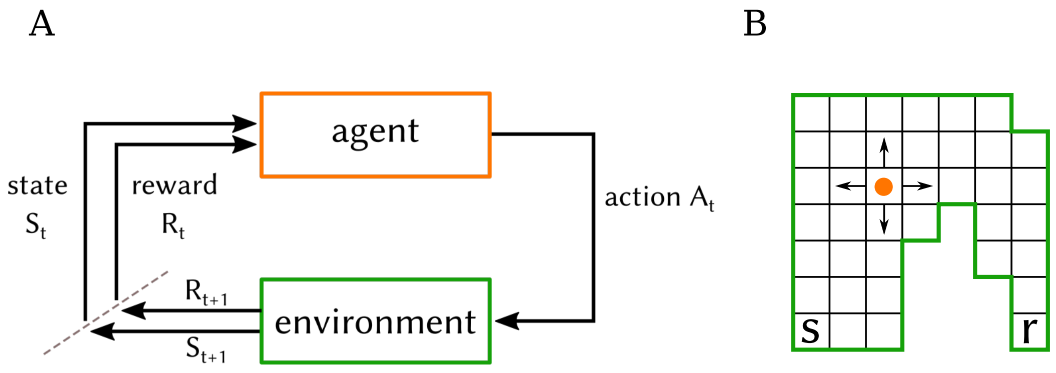


Figure 2.3: Problem-setup in reinforcement learning. (A) The two main components of reinforcement learning are the agent and the environment. The agent performs actions based on the current state of the environment. In turn, the environment makes a transition to a new state and provides a reward to the agent if applicable. **(B)** Arguably the simplest example of a reinforcement learning problem is a two-dimensional gridworld where the agent (orange dot) tries to get from the starting cell (**s**) to the rewarded cell (**r**) by moving into one of the available directions (four arrows). A reward is only provided if the agent reaches the target cell. Figures are reproduced following Sutton and Barto [2018].

A simple example of reinforcement learning scenario is the gridworld (figure 2.3 B). The environment is a two-dimensional grid where the agent can choose from four directions as actions. The state of the environment is the current position of the agent. Upon choosing a direction, the agent moves into that direction, unless it tries to cross the hard outer boundaries in which case nothing happens. The agent starts at a given square and when it reaches the goal square, it receives a reward. Gridworld exemplifies that the agent has to move over several state-action pairs to obtain the reward, and during the learning procedure it has to determine how its decisions at the start will later influence the reward.

Although reinforcement learning is sometimes regarded as a subtype of supervised learning, it is substantially different from it. In supervised learning, there is an already existing ordered knowledge, the dataset, from which the model can extract knowledge. In reinforcement learning, this knowledge has to be extracted from the interaction between the agent and the environment. Furthermore, in supervised learning, the feedback is instructive and dense. For each input data, the model receives feedback from the supervisor concerning what it should do, in particular a vector-valued feedback. For example, in the case of classification, the model receives the entire true label (output) vector from the supervisor. On the contrary in reinforcement learning, the reward signal is often sparse and instructive. It might arrive only after a long chain of interactions, and it only tells the model how good the outcome is. In particular, reward signals are scalar-valued, unlike the feedback in supervised learning. In computational neuroscience literature, there are models solving classification tasks with reinforcement learning models [Frémaux et al., 2010, Friedrich et al., 2011, Pozzi et al., 2018], but in these cases the focus of interest lies in the mechanistic models and the biological implications and not in finding efficient solutions.

In this section, we gave a brief description of reinforcement learning. For a thorough introduction, see Sutton and Barto [2018] and for a recent review on reinforcement learning using deep learning, see Arulkumaran et al. [2017].

2.2 Computational neuroscience: neurons, synapses and plasticity

The field of computational neuroscience studies the computational principles governing aspects of the nervous system such as vision, memory, learning and motor control. To this end, it uses mainly mathematical modeling and simulation studies while interacting with related fields such as experimental neuroscience, molecular chemistry and psychology. Computational neuroscience works with three kinds of models [Dayan and Abbott, 2001]: 1) descriptive models aim at characterizing experimental data, that is what the brain does, 2) mechanistic models explain how the neurons fulfill their function, and 3) interpretive models describe why the brain functions in a given way by, for example, looking at behavioral consequences.

In this section, we focus on the mechanistic models which are central to this work. In the following, we introduce the basics of biological neurons and synapses as well as simple models of the dynamics of point neurons. Furthermore, we describe the general mathematical formalism for models of learning and plasticity in networks of neurons.

There are many standard textbooks for computational neuroscience. We rely mainly on the works of Dayan and Abbott [2001], Gerstner and Kistler [2002b] and Izhikevich [2007b].

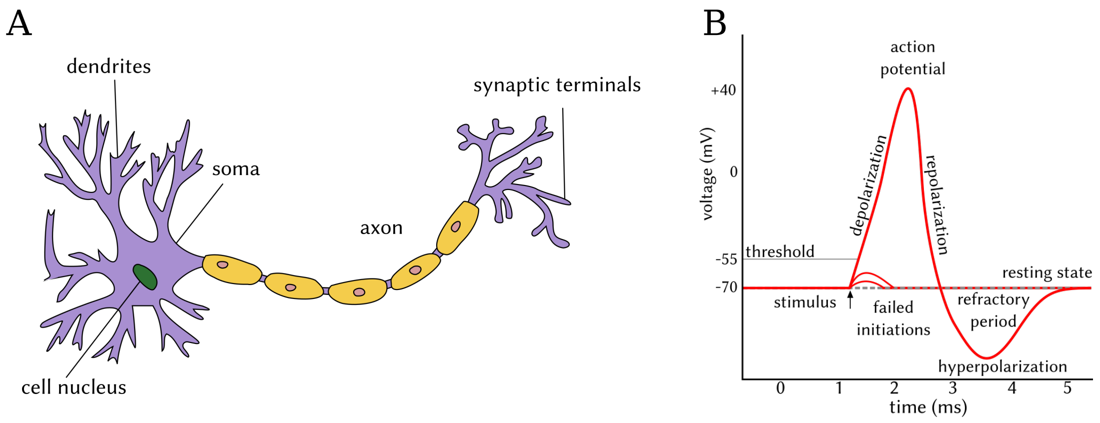


Figure 2.4: Morphology of neurons and the action potential. (A) Sketch of a neuron with annotated morphology. The tree-like dendritic structure receives input from other neurons, which are then integrated in the soma. The action potential is generated at the soma. It travels through the axon to the synaptic terminals where it triggers a synaptic transmission to other receiving neurons. Image is adapted from Jarosz [2009]. **(B)** Sketch of the temporal course of the action potential. If enough input arrives, the spike-generating mechanism is activated. The resulting action potential follows a stereotypical form beginning with a strong depolarization followed by a hyperpolarization. During the latter, the spike generation is prohibited hence it is called the refractory period. Image is adapted from Chris73 [2007].

2.2.1 Basics of information processing in the brain: neurons and synapses

Neurons or nerve cells are hypothesized as the basic information processing units of the nervous system. There are approximately 10^{11} neurons in the human brain [von Bartheld et al., 2016]. In a simplified view, neurons are electrically excitable cells specialized for receiving, integrating and transmitting information with electrochemical processes [Dayan and Abbott, 2001, Trepel, 2017]. Morphologically, neurons consist of three main parts: the dendritic tree, the soma and the axon (figure 2.4 A). The dendritic tree serves as the input part of the neuron: It receives the synaptic input and guides it towards the soma. It is estimated that each neuron receives input from about 10^4 other neurons [Pakkenberg et al., 2003], however this is an approximate number and it varies over different brain areas and cell-types. The soma integrates the input and if the accumulated input is sufficiently strong, it generates an all-or-nothing response in the membrane voltage, the so-called action potential (figure 2.4 B). Action-potentials are stereotypical: their shape and duration are always similar.

The electrically active properties of the neuron stem from the different ion-channels spanning through the cell membrane. These channels are in part passive or can be gated (opened or closed) by signaling proteins or by other stimuli, such as the membrane potential. The main ion-types are sodium (Na^+), potassium (K^+), calcium (Ca^{2+}) and chloride (Cl^-). Furthermore, $\mathrm{Na}^+/\mathrm{K}^+$ pumps actively transfer $3\,\mathrm{Na}^+$ out of the cell and $2\,\mathrm{K}^+$ into the cell. Due to the interaction of the pump and the other ion-channels, the inside of the neuron in the resting state (in humans) has an approximate potential of $-70\,\mathrm{mV}$ compared to the outside medium.

The action potential generation is the result of the non-linear dynamics of the voltage gated ion-channels. It was first modeled by Hodgkin and Huxley [1952]. In the following, we give a qualitative description of the firing mechanism. If the soma gathers enough input from external stimuli, a spike or action potential with a stereotypical shape is initiated (figure 2.4 B). First, the membrane potential rises quickly above $0\,\mathrm{mV}$, depolarizing the neuron. This is then followed by a fast drop below the resting potential, the so-called hyperpolarization. During this period, the generation of a next spike is suppressed, therefore we call it the refractory period. Finally, the membrane potential returns to the resting value. The shape and length of the action potentials is stereotypical; all the information is carried by the time of spiking[5]. Hence, we say that neurons communicate via all-or-nothing events. Note, the notion of sufficient input for spike generation varies among the different models. Simple models condition the spike-generation on reaching a membrane potential value. Other models — for example the Hodgkin-Huxley model [Hodgkin and Huxley, 1952] — generate action-potentials via the non-linear dynamics of the ion-channels, and hence do not have a strict threshold potential. Nevertheless, for the qualitative understanding and for many of the models, the assumption of a hard threshold potential is adequate.

[5]There are also other types of action potentials, where this notion could be violated. But we do not include them in this thesis.

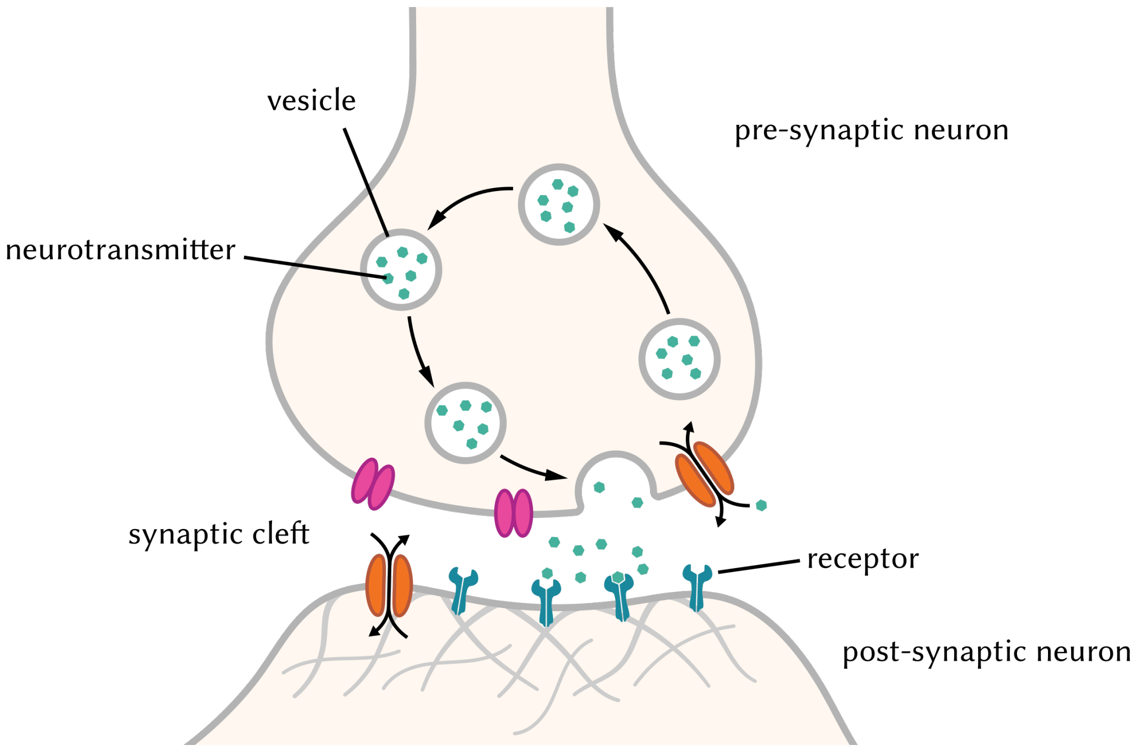


Figure 2.5: Sketch of a chemical synapse. When the action potential arrives at the synaptic terminals of the pre-synaptic neuron, it triggers the release of neurotransmitters into the synaptic cleft. The transmitters eventually bind to receptor proteins on the post-synaptic side opening ion-channels. This leads to a post-synaptic current (PSC) and in the end to a post-synaptic potential (PSP). Figure adapted from [Splettstoesser, 2015].

The generated action potential travels along the axon, eventually reaching the synaptic terminals. A synapse is a contact between two neurons that enables the passing of electric or chemical signals. Here, we only treat the chemical synapses (figure 2.5). The neuron with the axon at the synapse is called the pre-synaptic neuron (the sender of the signal), and the neuron with the dendrite is called the post-synaptic neuron (the receiver of the signal). However, synapses can also target other parts of the post-synaptic neuron. Signal-transmission in a chemical synapse is unidirectional and it goes from the pre-synaptic neuron to the post-synaptic one.

The gap between the two neurons is called the synaptic cleft. On the pre-synaptic side, neurotransmitters are produced and stored in vesicles. The incoming spike triggers the release of neurotransmitters into the synaptic cleft. The neurotransmitters eventually bind on receptor proteins on the post-synaptic side opening ion-channels. Through the activated channels current (in form of ions) can flow through the membrane, the so-called post-synaptic current (PSC). The PSC leads to changes of the membrane potential of the post-synaptic neuron, the so-called PSP. If the PSP pushes the neuron towards spiking, the synapse is called excitatory; and if the PSP hinders spiking, the synapse is called inhibitory. According to Dale's principle, a neuron releases the same type of neurotransmitters at all its synapses [Dale, 1953, Strata and Harvey, 1999]. Note that exceptions have been reported from Dale's principle [Sulzer and Rayport, 2000], and the effect of a synapse is defined by the combination of the neurotransmitter and the receptor. Nevertheless, it is mostly accepted that models should work with neurons that are either purely excitatory or purely inhibitory in their effect [Dayan and Abbott, 2001].

In this section, we only described the basic morphology and function of nerve cells. Several types of neurons, synapses and neurotransmitters have been identified and described, for more details see Kandel et al. [2000].

2.2.2 Models of neurons and synapses

There are several models describing the behavior and dynamics of neurons and their synaptic interactions. Their level of description stretches from detailed models capturing the spatial structure and different ion-channels of the neuron to point-neuron models restricted to the description of input-output relation. An important differentiation between the models is the treatment of spikes. In spike-based models, the neurons produce outputs in form of a series of spikes (spike-train). Hence, there is an ongoing conversion between the analog dynamics of the membrane potential and the more binary nature of the action potentials. In contrast to spike-based models, in rate-based models neurons produce a real-valued (usually time-continuous) signal as their output.

In the following, we introduce the leaky integrate-and-fire (LIF) model [Brunel and van Rossum, 2007], which is arguably the simplest spiking neuron model with biological relevance. The LIF model and its close alternatives are widely used in neuromorphic systems [Thakur et al., 2018] and in models of spiking neural networks [Tavanaei et al., 2019]. They are also implemented in the BrainScaleS 1 and 2 systems (chapters 3 and 4). Spiking neuron models are sometimes called neuron models of the third generation [Maass, 1997]. For an extensive collection of spiking neuron models see Gerstner and Kistler [2002b].

On the other hand, rate-based neuron models usually do not have specific names. They are sometimes referred to as neuron models of the second generation following Maass [1997]. In the following, we present the main characteristics of spike- and rate-based models on simple examples.

The leaky integrate-and-fire model

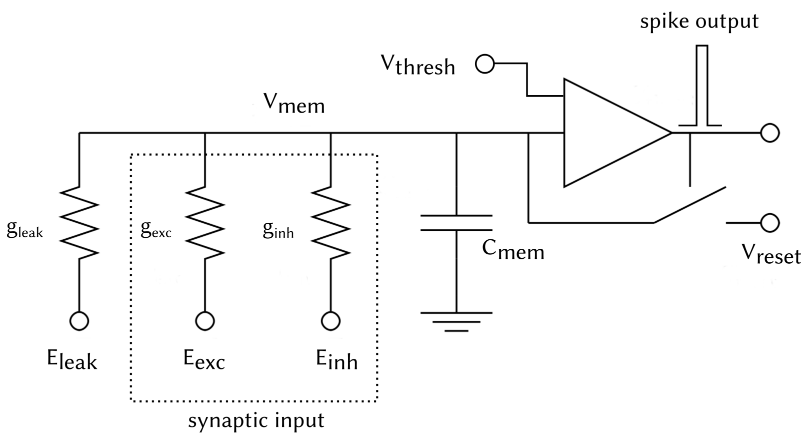


Figure 2.6: The circuit represented by the leaky integrate-and-fire neuron model with conductance-based synapses. The membrane of the neuron is modeled as a capacitor C_{mem} with a potential V_{mem}. The leakage and the synaptic inputs are connected to the membrane via conductances. If the membrane potential reaches the threshold value V_{thresh}, then the spiking is activated and the membrane is pulled to the reset potential V_{reset}. Image adapted from Stöckel [2015].

The model is based on an equivalent circuit model of a point neuron (figure 2.6). The cell membrane is represented by a capacitor C_{mem}, and the effective resting potential of the ion-pumps and channels is modeled by a single leak potential E_{leak} and a leak conductance g_{leak}. E_{leak} is also often referred to as resting potential. The firing and resetting mechanism is shown with a comparator and a voltage controlled switch. The dynamics of the membrane potential is governed by the first order ordinary differential equation:

$$C_{\mathrm{mem}} \frac{\mathrm{d}V_{\mathrm{mem}}}{\mathrm{d}t} = g_{\mathrm{leak}}(E_{\mathrm{leak}} - V_{\mathrm{mem}}) + I_{\mathrm{ext}} + I_{\mathrm{syn}} \quad , \tag{2.11}$$

where I_{syn} unifies the synaptic input from all other neurons both excitatory and inhibitory. Furthermore, I_{ext} symbolizes all the other external inputs, for example stimulus added by the experimenter. Without external input, the membrane potential relaxes to the leakage potential E_{leak} with the membrane time-constant

$\tau_{\text{mem}} = \frac{C_{\text{mem}}}{g_{\text{leak}}}$; while with external input, it performs a low-pass filtering on the external input. Additionally, a firing condition is imposed:

$$V_{\text{mem}}(t) = V_{\text{reset}} \quad \text{for} \quad t \in (t_{\text{spike}}, t_{\text{spike}} + \tau_{\text{ref}}] \quad \text{if} \quad V_{\text{mem}}(t_{\text{spike}}) \geq V_{\text{thresh}} \quad . \tag{2.12}$$

If the membrane potential reaches the firing threshold, a spike is produced and the membrane potential is instantaneously pulled to the reset potential V_{reset} for the duration of the refractory time τ_{ref}.

Note that the spiking mechanism is not explicitly modeled: the dynamics of the ion-channels — as discussed previously — is omitted due to the stereotypical shape of the action-potentials. Spikes are registered as instantaneous events with a time-stamp. The output of the spiking neuron $S(t)$ is called a spike-train, and it is written as a train of delta peaks

$$S(t) = \sum_{t_{\text{spike}} \in A_{\text{spikes}}} \delta\left(t - t_{\text{spike}}\right) \quad , \tag{2.13}$$

where the set $A_{\text{spikes}} = \{t_1, t_2, \ldots, t_N\}$ contains the time-stamps t_{spike} of the spikes generated by the spiking neuron.

Spike-based input to neurons is modeled via kernels. In the current-based (CUBA) model, the incoming spikes directly lead to a PSC via

$$I_{\text{syn}}^{\text{cuba}} = \sum_{\text{syn}\, k} \sum_{\text{spk}\, s_k} w_k \kappa(t - t_{s_k}) \theta(t - t_{s_k}) \quad , \tag{2.14}$$

where the sum over k goes over the incoming synapses, the sum over s_k goes over the spikes through these synapses. $\theta(\cdot)$ is the Heaviside step function to preserve causality and $\kappa(\cdot)$ is the synaptic kernel. Popular choices for the current-based synapse kernel are the delta peak kernel $\kappa(t) = \delta(t)$, exponential kernel $\kappa(t) = \exp\left(-t/\tau_{\text{syn}}\right)$ and the alpha-shaped kernel $\kappa(t) = \frac{t}{\tau_{\text{syn}}} \exp\left(-t/\tau_{\text{syn}}\right)$, where τ_{syn} is the synaptic time-constant. τ_{syn} can be different for the different synapses, and in case of analog neuromorphic implementations it often is (section 2.3.2). Sometimes the kernels are normed to a maximal height or to $\int_0^\infty \kappa(t) \mathrm{d}t = 1$, but there is no widely accepted consensus. In the case of current-based (CUBA) synapses, the weight w_k of a synapse has the dimension of a current. The idea behind the CUBA is that synaptic input from the dendrites is assumed to arrive mainly as a current at the soma.

In the case of conductance-based (COBA) synapses (shown in figure 2.6), the effect of the incoming synapses changes the value of conductance between the membrane potential and a synaptic reversal potential. The PSC is then given by:

$$I_{\text{syn}}^{\text{coba}} = \sum_{\text{syn}\, k} \sum_{\text{spk}\, \text{s}_\text{k}} (E_{\text{rev}}^x - V_{\text{mem}}) w_k \kappa(t - t_{s_k}) \theta(t - t_{s_k}) \quad , \tag{2.15}$$

where the symbols are the same as in equation (2.14) and $E_{\text{rev}}^x \in \{E_{\text{rev}}^{\text{inh}}; E_{\text{rev}}^{\text{exc}}\}$ is the synaptic reversal potential corresponding to the given synapses. In this case the synaptic weight w_k takes on the dimension of a conductance. For the kernel function, similar choices are popular as in the case of CUBA synapses, for example delta peak, exponential as well as alpha-shaped kernels. The idea behind this model is that synaptic input first modulates the conductance value of the ion-channels and only indirectly leads to a current generation.

A simple rate-based neuron model

The main difference between rate-based and spike-based models is the way of the output generation. For the sake of simplicity, we take the same membrane potential dynamics as is the case of the LIF model:

$$-C_{\mathrm{mem}} \frac{\mathrm{d}V_{\mathrm{mem}}}{\mathrm{d}t} = g_{\mathrm{leak}}(V_{\mathrm{mem}} - E_{\mathrm{leak}}) + I_{\mathrm{ext}} + I_{\mathrm{syn}} \quad . \tag{2.16}$$

However, there is no resetting mechanism. Instead, we define the output of the neuron as function of the membrane potential $f(V_{\mathrm{mem}})$. $f(\cdot)$ is called the activation function of the neuron (figure 2.7), similarly to the activation function of abstract neurons (section 2.1.1). A popular choice for the activation function is the logistic function[6]:

$$f(V_{\mathrm{mem}}) = \frac{1}{1 + \exp\left(-\frac{V_{\mathrm{mem}} - b}{\alpha}\right)} \tag{2.17}$$

with a width α and a bias value b, but sometimes the softplus function [Dugas et al., 2001] is also used:

$$f(V_{\mathrm{mem}}) = \lambda d \ln\left(\exp\left(\frac{V_{\mathrm{mem}} - b}{d}\right) + 1\right) \tag{2.18}$$

with λ being the slope, d the width and b the bias value.

Rate-based input to neurons can be formulated both for the CUBA and the COBA synapse models. In the CUBA case the PSC is given by

$$I_{\mathrm{syn}}^{\mathrm{cuba}} = \sum_{\mathrm{syn}\,k} w_k f(V_{\mathrm{mem}}^k) \quad , \tag{2.19}$$

where the only a single sum goes over the incoming synapses. Note that no kernels are involved. Similarly, in the COBA case, the PSC is given by

$$I_{\mathrm{syn}}^{\mathrm{coba}} = \sum_{syn\,k} w_k f(V_{\mathrm{mem}}^k)(E_{\mathrm{rev}}^x - V_{\mathrm{mem}}) \quad . \tag{2.20}$$

Again, $E_{\mathrm{rev}}^x \in \{E_{\mathrm{rev}}^{\mathrm{inh}}; E_{\mathrm{rev}}^{\mathrm{exc}}\}$ is the synaptic reversal potential corresponding to the given synapses.

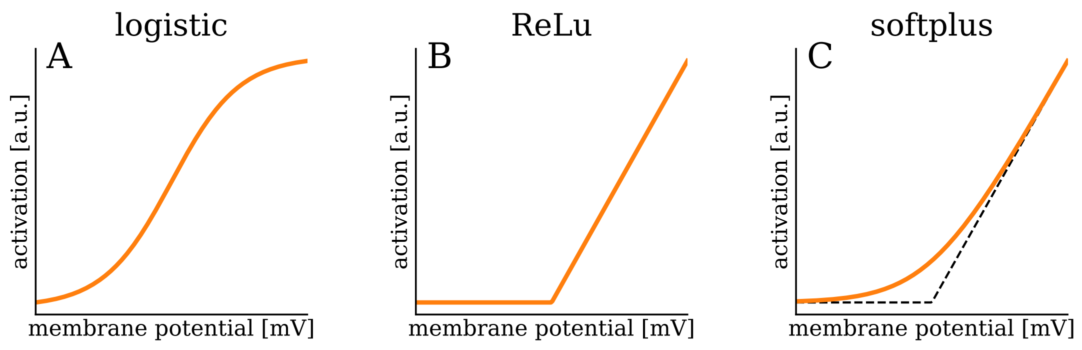


Figure 2.7: Examples of activation functions used in computational neuroscience. (A) The logistic activation function is probably the most widely used activation function. It saturates at a maximum activity for high membrane potentials. **(B)** The rectified linear unit (ReLu) is widely used in machine learning due to its easy computability. It is also used in studies in computational neuroscience. A drawback of the ReLu is the run-away activity for high membrane potentials. **(C)** The softplus function [Dugas et al., 2001] has been proposed as an everywhere differentiable version of the ReLu. The dashed line is the ReLu function as a comparison. The dimension of the activation depends on other modeling decisions. Usually, the activation is interpreted as the instantaneous spike-rate with a dimension of Hz.

Why spikes? The difference between spike-based and rate-based models

The difference in dynamics is exemplified in figure 2.8. The integration of input is similar in both models; the main difference is in the output generation process. Rate-based neurons produce a real-valued output at all times. For most models, this output is differentiable almost everywhere. Importantly, output is transmitted and generated even if this input is always zero. On the other hand, in spike-based models, the output is a spike-train, that is, a series of discrete events. Compared to rate-based models, spike-based models are silent and do not send signals if there is nothing to transmit. There is a second difference: in spike-based models, the spike generating mechanism has an effect on the membrane dynamics via the threshold and reset operations. This second effect is not modeled in every spike-based model, although it could lead to serious implications in plasticity models relying on the membrane potential. From a practical point of view, including spikes makes modeling more complex. The discontinuity at the point of the spike generation complicates the analytical tractability of the network dynamics.

The functional meaning of spike-based computation in the brain is still an open question. Opinions cover a wide range from 1) spikes hypothesized as mere proxies for rates to 2) assuming new computational paradigms, which cannot be realized with rates. Still, the presence of action potentials in the brain is undeniable.

[6]Cramer [2002] tracks back the development of the logistic function to Pierre François Verhulst (1804 - 1849).

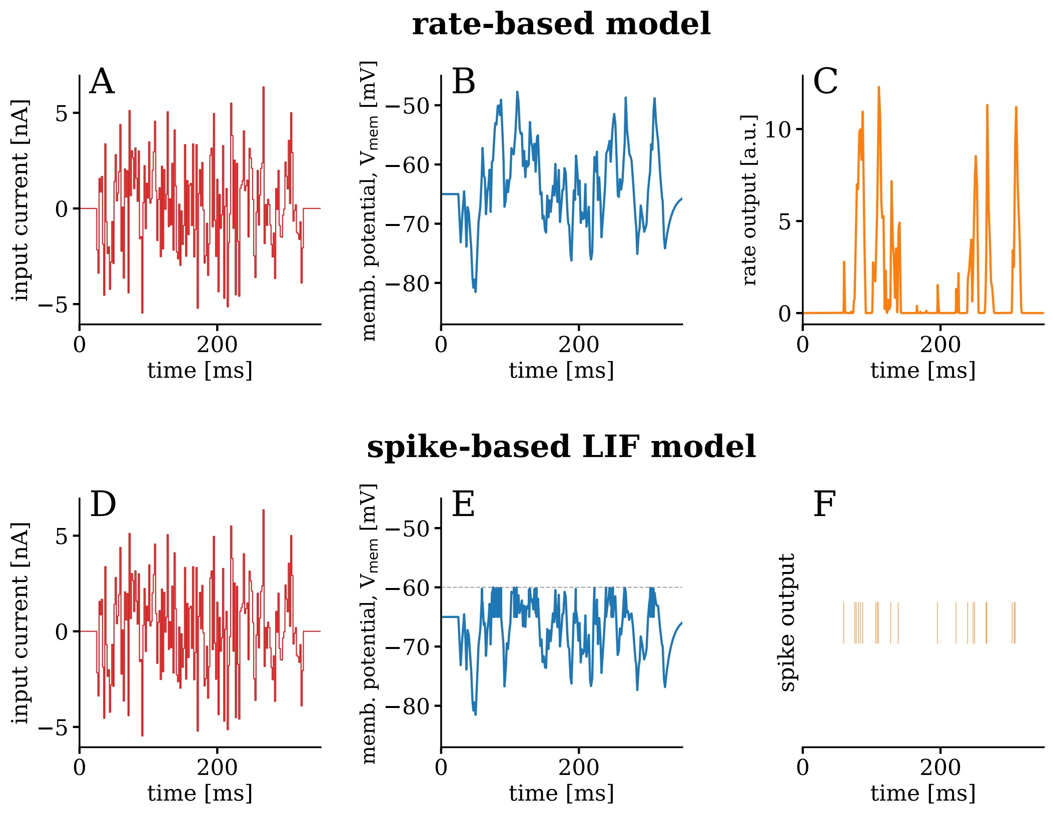


Figure 2.8: Comparison between a simple rate-based model and the LIF model. Both models have the same subthreshold dynamics and receive the same input but differ in their output generation. The upper row is a rate-based model. **(A)** The neuron receives time-varying Gaussian white noise input in form of a current and **(B)** performs a low-pass filtering on the input via its membrane potential dynamics with $E_{\mathrm{leak}} = -65\,\mathrm{mV}$. **(C)** The output is generated via the softplus activation function $f(V_{\mathrm{mem}}) = 0.5 \ln(\exp(2(V_{\mathrm{mem}} + 60)) + 1)$. Note that there is no resetting mechanism on the membrane potential. **(D)** In the case of the LIF model, the same input is received, which is **(E)** low-pass filtered by the membrane potential, but this low-pass filtering is disrupted by the resetting mechanism with $V_{\mathrm{reset}} = -65\,\mathrm{mV}$. The threshold value $V_{\mathrm{thresh}} = -60\,\mathrm{mV}$ is indicated with a gray dashed line. **(F)** The generated output is a spike-train, a series of discrete events.

In the following, we give a brief list of the proposed motivations and direction of research on the functional relevance of spikes:

Energy efficiency: Spike-based communication was suggested to maximize the amount of transmitted information per invested metabolic energy [Levy and Baxter, 1996, Harris et al., 2015]. It has been argued that in typical biological environments, spike-based communication is more energy efficient than rate-based communication.

Sparse coding: The property that no signal is transmitted if the output is constant zero is appealing for sparse coding. Recent results in neuromorphic engineering show that sparse coding with spikes can be faster than conventional sparse coding methods [Davies et al., 2018].

Signal multiplexing: Naud and Sprekeler [2017] propose a method how spike-based communication could realize signal multiplexing through neurons.

Spiking activity generated variability: The asynchronous irregular firing regime of spiking neural networks could serve as a source of variability or pseudo-stochasticity for other computations [Mazzucato et al., 2019, Jordan et al., 2019, Dold et al., 2019].

Measuring the causal effect: Lansdell and Kording [2019] propose, that the spiking dynamics could be used to estimate the causal influence of neurons. This idea might lead to new learning rules.

Novel coding schemes: The search for the functional meaning of spikes has lead to the construction of novel coding schemes. Some of them are unique to spikes, such as time-to-first-spike [Mostafa, 2017, Göltz et al., 2019], phasor networks [Frady and Sommer, 2019] or spiking sampling networks [Petrovici et al., 2016].

The search for the functional meaning of spikes in the nervous system is closely related to the development of neuromorphic engineering, which aims at using the emerging ideas to develop novel computing platforms (section 2.3).

2.2.3 Models of plasticity in biological neural networks

Plasticity is the capability of neurons and synapses to change their parameters depending on their past activity. Most frequently, we speak of synaptic plasticity, meaning the activity-dependent change of the synaptic weights, but we can consider changes in other neuron parameters as plasticity as well. Activity-dependent plasticity is generally considered as the basic phenomenon behind the learning of motoric skills and formation of memories [Dayan and Abbott, 2001]. A historical review on plasticity and on the perspectives of plasticity research is given in Markram et al. [2011a].

There are several models trying to explain experimental evidences regarding synaptic plasticity. Each of them relies on or explains aspects of plasticity experiments but the true plasticity rule(s) of the brain remain(s) elusive. The first qualitative description of synaptic plasticity goes back to Hebb's famous rule, often summarized as: if neurons fire together, they wire together [Hebb, 1949]. However, Hebb did not gave a mathematical formula to his qualitative description.[7] The most common requirement for plasticity is locality. Generally, we assume that the equations governing the change of the neuron parameters and synaptic weights should only depend on locally accessible variables, such as the spiking activity of the pre- and post-synaptic neurons. This requirement is often violated by implicitly requiring non-local computations for plasticity, for example Zenke and Ganguli [2018].

Note, that there is a difference between local variables and locally accessible variables. Locally accessible variables are quantities that are plausible accessible by the neurons and synapses to influence their plasticity. Local variables are specific for any given neuron or synapse, for example a function of the pre- and post-synaptic activities. By a global variable we mean a quantity that is the same over the entire (or at least large parts of) the neural network. An example for a global variable are the neuromodulators: neurotransmitters that not only influence the post-synaptic neuron but a larger group of neurons by, for instance, modifying their plasticity, see for example Gerstner et al. [2018]. In this sense, neuromodulators are global variables that are at the same time locally accessible.

In the following, we give the basic and simple structure of formulating plasticity rules proposed by Gerstner and Kistler [2002a]. Rate-based plasticity can be written in the general form:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = F_{\text{rate}}(\nu_i, \nu_j; w_{ij}) \quad , \tag{2.21}$$

where F_{rate} is a general function of the post- and pre-synaptic firing rates (activities) ν_i and ν_j as well as the synaptic weight w_{ij}. By taking the Taylor expansion of F_{rate} up to a given degree, we can reconstruct many rate-based plasticity rules found in literature, for example the plasticity models in Oja [1982] and Bienenstock et al. [1982]. This formulation only implicitly includes plasticity rules defined on neuron models with spatial extension and plasticity rules using membrane potentials. Still, it captures the main belief of locality in synaptic rules.

Similarly, for spike-based models, we write:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = F_{\text{spike}}(S_i^{\text{post}}(t'), S_j^{\text{pre}}(t''); w_{ij}) \quad , \tag{2.22}$$

where $S_i^{\text{post}}(t')$ and $S_j^{\text{pre}}(t'')$ are the spike-trains from the post- and pre-synaptic neurons. F_{spike} is now a functional of the spike-trains. The different times t' and

[7]Hebb's original formulation was: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." — from Hebb [1949].

t'' indicate that the rate of change of the synaptic weight does not only depend on the current spiking properties but can also shows memory effects. A Volterra expansion of F_{spike} yields the different spike-based plasticity models.[8] For example, the classic spike-time-dependent plasticity (STDP) learning rule — first observed experimentally by Bi and Poo [1998] — can be written as:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} = \underbrace{\int_{-\infty}^{t} W(t'-t) S_i^{\text{post}}(t') S_j^{\text{pre}}(t) \mathrm{d}t'}_{\text{post before pre}} + \underbrace{\int_{-\infty}^{t} W(t-t') S_i^{\text{post}}(t) S_j^{\text{pre}}(t') \mathrm{d}t'}_{\text{pre before post}} \quad , \tag{2.23}$$

where $W(t_{\text{pre}} - t_{\text{post}})$ is the STDP function. The choice of the variables t_{pre} and t_{post} highlights that the STDP function always takes relative spike-timing differences as an argument. A popular choice for the STDP function is the exponential kernel (see also figure 2.9):

$$W(t_{\text{pre}} - t_{\text{post}}) = \begin{cases} a^{+} \exp\left(-\dfrac{t_{\text{post}} - t_{\text{pre}}}{\tau_{+}}\right) & \text{if } t_{\text{pre}} < t_{\text{post}} \\ a^{-} \exp\left(-\dfrac{t_{\text{pre}} - t_{\text{post}}}{\tau_{-}}\right) & \text{if } t_{\text{pre}} > t_{\text{post}} \quad , \end{cases} \tag{2.24}$$

where a^{+} and τ_{+} parameterizes the causal (pre-before-post) branch and a^{-} and τ_{-} parameterizes the anti-causal (post-before-pre) branch of the spike-time-dependent plasticity (STDP) function. Equation (2.23) is formulated to evoke synaptic plasticity over all spike pairing, but often STDP is formulated in a way to only include next neighbor spike-pairs. After the discovery of Bi and Poo [1998], STDP was considered as always causally strengthening and anti-causally weakening. Nowadays, as it is also apparent from the general formulation of the theory, the STDP function is not looked at as a hard-wired rule. Instead, it is thought of as a building block for more complex learning rules based on causal and anti-causal correlation measurements, which are carried out locally at the synapses.

2.3 A brief overview on neuromorphic engineering

In the following section, we give a brief overview on the different aspects of neuromorphic engineering. We discuss the origin of the term, its definition as it is most commonly used nowadays, the different approaches of hardware implementation and the current challenges of the field. The aim of this section is not to give an extensive review on the entirety of the field, but to sketch the place of this thesis and that of the BrainScaleS-1 [Schemmel et al., 2010] and BrainScaleS-2 [Friedmann et al., 2017] systems in the field of neuromorphic engineering.

Compared to computational neuroscience and machine learning, the neuromorphic community is small and there are no standard textbooks summarizing the principles and aims of the field, hence, a more extensive introduction is required.

[8]For more on Volterra expansion see for example Schetzen [1980].

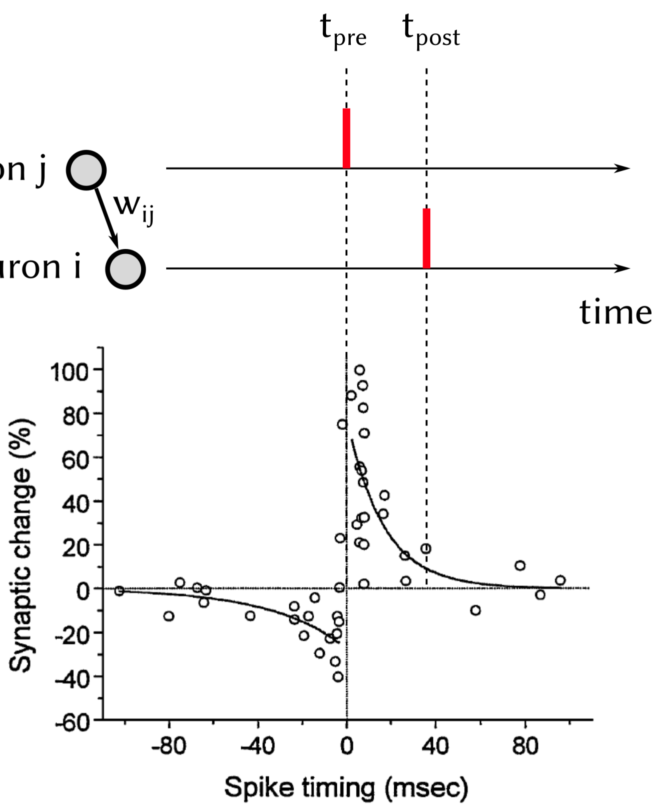


Figure 2.9: Sketch of the classic Bi-Poo STDP mechanism. Neuron *j* is connected to neuron *i* via the synapse w_{ij}. In this setup, the post-synaptic neuron *i* fires after the pre-synaptic neuron *j*. Because of the causal spike-pairing, the synapse is strengthened. In the case of an anti-causal spike-paring, the synapse would be weakened. The lower part of the figure is taken from Bi and Poo [2001] containing data from Bi and Poo [1998].

In the recent years, several reviews appeared on the different aspects of neuromorphic engineering [Indiveri et al., 2011, Indiveri and Horiuchi, 2011, Vanarse et al., 2016, Furber, 2016, Nawrocki et al., 2016, Schuman et al., 2017, Thakur et al., 2018, Li et al., 2018, Pfeiffer and Pfeil, 2018, Lee et al., 2019, Roy et al., 2019, Rajendran et al., 2019].

From these reviews, we highlight three: Schuman et al. [2017] did the spadework of summarizing, ordering and synthesizing publications over a 35 years span. Pfeiffer and Pfeil [2018] review and categorize the modeling approaches for neuromorphic systems, however, they only focus on spiking neural networks. Thakur et al. [2018] give a list of current neuromorphic platforms with strong focus on the hardware implementation.

2.3.1 Imitating the brain: Definitions and motivation of neuromorphic engineering

The term neuromorphic computing or neuromorphic engineering was coined by Carver Mead in 1989 [Mead, 1989, 1990]. Originally, it referred only to the analog and mixed-signal neuron emulations in Very Large Scale Integration (VLSI). VLSI is the technique of combining millions of transistors on a single integrated circuit [Mead and Conway, 1979, Mead, 1989]. It became available in the 1970s and became widely adopted in the 1980s, that is almost coinciding with the first ideas of neuromorphic engineering. Nowadays, authors often refer to any hardware implementation or algorithm as neuromorphic if it is inspired by aspects of information processing in the brain or if it uses any non von-Neumann architecture. Neuromorphic hardware can be seen as an alternative to von-Neumann architecture featuring low-power and highly parallel information processing. In contrast to the strict separation of memory and processing unit in the von-Neumann design [Von Neumann, 1993], neuromorphic engineering co-locates memory and the processing units, usually the neurons, synapses and their parameters [Schuman et al., 2017].

In the beginning, neuromorphic engineering was more explorative and lead to the development of new types of sensors [Vanarse et al., 2016]. As the aggressive digital node development towards always smaller node sizes faces serious limitations, the interest renewed for alternative computation paradigms, among others for neuromorphic computing [Monroe, 2014]. The development of traditional digital von-Neumann architectures is limited by three main factors: 1) Moore's law is assumed to near its end [Waldrop, 2016] and certainly its future pursuit is increasingly capital-intensive [Khan et al., 2018], 2) the power efficiency does not scale with the reduced transistor size resulting in increasing energy demand and by that heat production (Dennard-scaling, Esmaeilzadeh et al. [2011]), 3) and efficiency loss becomes more significant due to the separation of memory and central processing unit (CPU), known as the von-Neumann bottleneck. Neuromorphic computing could replace or complement traditional hardware, such as CPUs and GPUs, in certain applications. Schuman et al. [2017] identify ten different motivations researchers gave for neuromorphic engineering.

Parallelism: Neuromorphic hardware emphasizes a design based on many simple processing components and dense interconnection between them, usually called the neurons and synapses. This biologically inspired design gives rise to massively parallel computation.

Von-Neumann bottleneck: By co-locating memory and processing units — again usually the neurons and synapses —, neuromorphic engineering promises to avoid the von-Neumann Bottleneck.

Scalability: Due to the uniform structure of the chips and the distributed computation paradigm, the size of the neuromorphic chips could, in principle, be easily scaled. In practice, scalability also requires the development of appropriate communication channels between the subunits.

Real-time performance: This was a motivator for earlier works on neuromorphic engineering. It mainly refers to the speed of execution for an application independent of the algorithm. In terms of application speed, neuromorphic sensors excel.

Low power: An increasingly popular motivation is the low power consumption, which is a central aspect for robotics, wearable devices or in Internet of Things (IoT) applications.

Footprint: A small footprint (physical chip size) can be important in robotics and wearable devices.

Fault tolerance: Inspired by the remarkable fault tolerance of biological brains [Levin et al., 1987, Aerts et al., 2016], silicon neural circuits promise fault tolerance due to their redundant architecture and self-learning capabilities.

Faster computation: Related to the massive parallelism, some neuromorphic architectures promise faster neural network emulation than conventional chips. Hence, neuromorphic hardware could be used as an accelerator for machine learning applications or simulators. This motivation focuses on neuromorphic hardware as an accelerator for already existing models or close derivatives (compare to real-time performance). In practice, the speed of computation is often reduced by the system overhead [van Albada et al., 2018, Kungl et al., 2019].

Online learning: By mimicking the brain's continuous learning and adaptation properties, neuromorphic engineering aims to build self-learning systems with plasticity algorithms running on the neuromorphic chip. However, continuous learning in the brain is not completely understood, and therefore designers can only get a vague inspiration. Designing networks and learning algorithms that both respect the constraints of the platform and fulfill the computational task is challenging, see e.g. Kungl et al. [2019] and chapter 3.

Neuroscience: Related to the faster computation and online learning motivations, neuromorphic hardware could be a tool for computational neuroscience. Simulation of large-scale neural networks is only feasibly for short simulations even on supercomputers, because of the limited speed of the calculations and the high power consumption [Jordan et al., 2018]. Neuromorphic hardware could serve as a platform for fast network emulation [Rhodes et al., 2019]. This requires a highly customizable platform to be able to serve the diverse needs of modelers on the one hand, and a user-friendly software stack to enable the hardware usage for a broad scientific community on the other hand.

A common challenge with these motivations is designing neural networks and algorithms that can 1) realize these motivations 2) fulfill the given computational task and 3) respect the constraints of the hardware. This is most apparent in the case of fault tolerance and online learning. We discuss the approaches of modeling for neuromorphic hardware in section 2.3.3.

2.3.2 Different approaches of neuromorphic engineering

Since the beginnings of neuromorphic computing, researchers developed several different approaches to implement aspects of the nervous system *in silico*. The proposed and built implementations vary in their level of abstraction from neuron models including spatial structure and ion-channels to digital matrix-vector multiplication accelerators that are not always called neuromorphic. We give a list of the neuromorphic approaches ordered by the choice they make between flexibility and the (potential) advantages in terms of speed of computation and energy efficiency (figure 2.10). For each type, we give some prominent examples, without claim to completeness (table 2.1). Note, that this ordering is only one potential choice and the reduction to a single dimension necessarily neglects details, which might be crucial for some applications. We also mention CPUs and GPUs as a comparison, although, they are never considered neuromorphic. The list is a snapshot of the state-of-the-art of neuromorphic computing as it is perceived at the time of writing. This snapshot nature of the list is most apparent in the case of CPUs, where a long tradition of tool-chain development marks the place of the CPU at the far side of the scale regarding ease of use. In the case of new materials, we can sometimes only talk about theoretical projections and single device measurements; they still have to prove their advantages in an integrated system.

On the extreme side at flexibility and ease-of-use, there is the traditional *central processing unit (CPU)* based on the von-Neumann architecture [Von Neumann, 1993]. They implement approximately the opposite of the ideas of neuromorphic computing. The memory and the processing unit are sharply separated [Drepper, 2007], the system is built on binary logic and the operations are carried out following a central clock in a synchronized manner. Although, the appearance of multi-core CPUs introduced parallelism. Due to the long tradition of compiler and tool-chain development, using CPUs is easy and flexible.

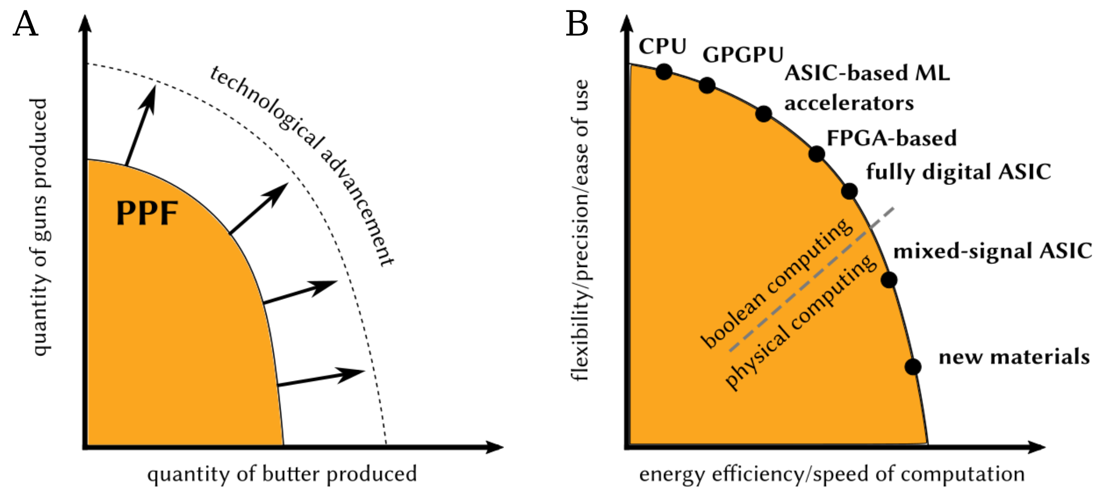

A
PPF
technological advancement
quantity of guns produced
quantity of butter produced
B
CPU
GPGPU
ASIC-based ML accelerators
FPGA-based
fully digital ASIC
mixed-signal ASIC
new materials
boolean computing
physical computing
flexibility/precision/ease of use
energy efficiency/speed of computation

Figure 2.10: Trade-off choice of different neuromorphic approaches. (A) The production-possibility frontier (PPF) model summarizes the production trade-off an economy has to cope with when choosing what to produce [Samuelson, 2010, Mankiw, 2012]. For each additional unit of gun produced the country has to give up a part of the butter production. Technological advancement can push the frontier farther to higher quantities of both but the necessary trade-off persists. **(B)** Inspired by the PPF model, the plot shows the trade-off choice of the different neuromorphic approaches. The figure is a snapshot of the existing neuromorphic approaches. Due to the two dimensional projection, there are simplifications, for example there is a trade-off between speed of computation and the energy efficiency as well. The "/" sign denotes an "and/or" connection: for example most research regarding new materials focuses on energy efficiency and less on speed of computation. The dashed line marks the approximate border between systems based on boolean logic and systems based on computations using directly the physics of the underlying substrate.

A *General Purpose Graphical Processing Unit (GPGPU)* is a digital stream processor specialized for massive vector operations suited for parallel computations, more precisely for single instruction multiple data computations [Owens et al., 2007, Navarro et al., 2014]. They grew out of GPUs specialized for maximizing performance on computer graphics applications. The return and emergence of deep learning relied heavily on the availability of affordable GPUs [LeCun et al., 2015], because the evaluation of neural networks contains many parallelizable operations such as the matrix-vector multiplication. At the beginning, GPGPU required lot of expertise to program, but with the appearance of the CUDA application programming interface [Nickolls et al., 2008] and of deep learning libraries, for example TensorFlow [Abadi et al., 2015], PyTorch [Paszke et al., 2019] or Theano [Theano Development Team, 2016]; the usage of GPGPUs became more accessible. GPGPUs are a widely used resource in scientific computing including research in computational neuroscience and machine learning.

An interesting type of neuromorphic systems are digital *machine learning accelerators based on application-specific integrated circuit (ASIC)*. These systems are custom built chips with the aim to accelerate the evaluation and to reduce the energy consumption of machine learning models, usually convolutional artificial neural networks [LeCun et al., 1989]. They do not focus on emulating aspects of the nervous system, but they are tailored to existing models. In a simplified view, they are matrix-vector multiplication accelerators with additional special features for parallel computation of activation functions and data-reuse during calculations. The most prominent example is the Tensor Processing Unit (TPU) from Google [Jouppi et al., 2017], which showed increased inference speed and reduced power consumption compared to GPUs. Similar machine learning accelerators have been produced, for example Intel's machine learning accelerators [Intel, 2019] or the Eyeriss chip [Chen et al., 2016, 2019]. The usage of these systems is similar to that of GPGPUs, because necessary libraries are (often) integrated into popular machine learning frameworks, for example TensorFlow has corresponding software back-end for TPUs. There seems to be an increasing trend for building custom designed machine learning accelerators. ASIC-based machine learning accelerators are first of all commercial products, but sometimes they are used due to their speed in machine learning research as well [Vinyals et al., 2019]. The inclusion of machine learning accelerator modules and vector extensions on CPUs blurs the border between CPUs, GPGPUs and stand-alone machine learning accelerators [Rodriguez et al., 2018].

A large group of *neuromorphic platforms* is implemented *on commercially available field-programmable gate arrays (FPGAs)*. They aim at exploiting the speed and power efficiency of the FPGA boards while benefiting from the available development tools. Note that FPGAs are usually power efficient compared to CPUs and GPUs but not compared to the following approaches. Because no custom hardware is produced, FPGA-based neuromorphic features a high degree of flexibility and short development cycles. Often, FPGA-based simulations are used for prototyping and pre-production testing of digital neuromorphic systems based on ASIC [Schuman et al., 2017]. A prominent example of an FPGA based neuromorphic

Category	*Feature*	*Application*	*Remark*	*Example*
CPU	general purpose application	general purpose, flexible control flow	mature tool-chain	off-the-shelf commercial
GPGPU	parallel compute units, parallel data interface	machine learning, scientific computing	originally tailored for graphics	off-the-shelf commercial
ML accelerator	ASIC-based	machine learning acceleration	developed for existing models	TPU, Eyeriss
FPGA-based	on commercial FPGAs	fast simulation, prototyping	on off-the-shelf hardware	DeepSouth
fully digital ASIC	simulated neuron dynamics	spiking computation, robotics	exploits digital design tools	TrueNorth, Loihi, SpiNNaker, Tianjic
real-time mixed-signal ASIC	emulated neuron dynamics	medical application, robotics	ultra-low power consumption	Neurogrid, DYNAPse
accelerated mixed-signal ASIC	emulated neuron dynamics	long experiments, plasticity experiments	simulator alternative, low energy	BrainScaleS 1/2
new materials	novel circuit elements	inference, emulated synaptic plasticity	in its infancy	memristors

Table 2.1: Summary of the different neuromorphic approaches. Table corresponds to the shown approaches in figure 2.10. References and abbreviations are given in the main text. Only the approaches below the dashed line are usually referred to as neuromorphic.

system is the DeepSouth cortex emulator [Wang et al., 2018]. FPGA-based systems have applications as machine learning accelerators, fast neural network simulators and co-processors for CPUs. These group of hardware are the first on the list that are generally referred to as neuromorphic.

A rapidly growing family of systems is based on *fully digital ASIC neuromorphic* emulator platforms. Unlike the digital ASIC-based machine learning accelerators, they do not aim at accelerating already known and well-explored models, but they provide a platform to emulate networks of (usually) spiking neurons with own internal dynamics. However, the neural dynamics are not emulated on designated analog circuits, but calculated using small neuromorphic cores, where a single core is responsible for the dynamics of typically several hundred neurons. Some of these systems also use asynchronous digital logic. Because the neural dynamics are simulated, the speed of execution can be regulated or even halted. Most systems are designed for real-time operation, meaning that the calculations are done with the same speed as the calculated neural dynamics. Examples of fully digital neuromorphic systems are the TrueNorth [Akopyan et al., 2015], the Darwin [Shen et al., 2016], the Loihi [Davies et al., 2018] and the Tianjic chips [Pei et al., 2019].

The SpiNNaker neuromorphic platform exemplifies a unique approach in this category [Furber et al., 2012, 2014]. It uses small Advanced RISC Machine (ARM) processors as neuromorphic cores connected with a custom spike-router. The processor enables the flexible implementation of neuron and synapse models, and the spike-router enables networks with arbitrary connectivity. Due to the combination of biological neuron models and relatively flexible usage, SpiNNaker systems find broad applications such as neuromorphic robotics, interfacing with neuromorphic sensors and prototyping brain-inspired computations.

Until this point, all systems used digital logic based on boolean algebra to calculate the dynamics of the neurons. These systems benefit from the established tool-chains of digital design such as error correction protocols, verification tools and the resulting short development cycles. They can almost immediately use new digital technologies. Furthermore, they can reach lower power consumption and smaller footprint by using smaller transistors. Finally, all digital systems benefit from the deterministic calculations of boolean logic. But there is a limit in efficiency: digital systems first use inherently analog signals (voltage, current) to represent digital values, and use them to calculate dynamics, which are analog in their nature such as the neural dynamics in the brain. We call this approach *boolean computation*, although according to my best knowledge there is no single collective term for it.

Physical model systems or *physical computing* on the other hand — sometimes simply called analog systems — take a different approach. They assume that most relevant problems can be solved via analog calculations, a view inspired by the nervous system. Physical model systems use dedicated analog circuits to emulate analog dynamics. These systems have the promise a faster and more energy efficient computation compared to their fully digital counterparts.

These potential advantages come with drawbacks: 1) Analog circuits show circuit-to-circuit variability introduced by manufacturing imperfections, the so-called *fixed-pattern noise*. Fixed-pattern noise is also present in digital circuits, but its effect on calculations is usually hidden from the user due to the digitization of the signals and due to the established error correcting protocols. Fixed-pattern noise reduces the controllability of the circuit parameters. 2) Analog circuits also have *temporal noise* on the their analog quantities. With temporal noise we mean any kind of fluctuation that is variable on the relevant time-scale of the experiment or application. Temporal noise can have diverse sources such as write cycle-to-cycle variation, thermal noise or fluctuations in the supply voltage. Physical model systems share similar challenges as biological neural networks. The brain also has to cope with heterogeneous neurons and synapses, with uncontrolled fluctuations and with restrictions in the potential synaptic connections and in the possible learning rules.

Analog mixed-signal ASIC hardware based on conventional complementary metal-oxide-semiconductor (CMOS) transistors uses a combination of analog and digital electronics to realize brain-inspired physical computation. The emulation of the spiking network dynamics is carried out on dedicated analog circuits, while communication between the neurons, and between the chip and the external components happens in a digital way. Spikes are usually interpreted as digital events with solely temporal information. The emulated neuron, synapse and plasticity models vary broadly, for a review see Indiveri et al. [2011]. There are two main types of these systems.

In *sub-threshold design*, the transistors operate in their sub-threshold regime and the neurons are emulated with typical time-constants close to biological neurons, e.g. approximately 10 ms for the membrane time-constants. Such real-time systems feature ultra-low power consumption and — due to the similar time-constants — simple interfacing with neuromorphic sensors. Hence, they are suitable for IoT applications, wearable devices and neuromorphic robotics. However, real-time systems suffer from large device-to-device variations due to the exponential sub-threshold characteristics of the CMOS transistors. Examples of such systems are the Neurogrid project [Benjamin et al., 2014], the BrainDrop [Neckar et al., 2018], the ROLLS [Qiao et al., 2015] and DYNAPs chips [Moradi et al., 2017].

Accelerated mixed-signal neuromorphic systems drive the transistors in the supra-threshold regime, and emulate shorter time-constants than biological neurons due to larger currents. The acceleration factor varies depending on the design typically between 10 and 10^5. The *supra-threshold design* allows for more precision, but also leads to higher power consumption and requires more complex circuits. Designers of these systems argue that the computation can be still not power but energy efficient: the devices use more power than the sub-threshold solutions but the computation is also executed faster (see Wunderlich et al. [2019] and chapter 4). Due to the short time-constants, interfacing with the environment — for example with robots or sensors — is a challenging task, and the fast emulation requires suitably fast I/O circuits. Examples of accelerated systems are the two generations of the BrainScaleS system [Schemmel et al., 2010, BrainScaleS, 2011, Friedmann et al., 2017], also discussed in detail in chapter 3 and chapter 4, respectively.

Several *new materials* have been proposed as good candidates for neuromorphic computing; for a review see Lee et al. [2019]. Many of them stem from conventional memory research, where they were suggested as fast and low-power non-volatile memory. They also show a history dependent parameter adaptation, that resembles short- and long-term plasticity observed in neuroscience experiments; and can be used for emulation of synapses and plasticity. Neuromorphic computing with these new materials is in its infancy; publications often focus on the properties of single devices. Examples for these new candidate materials are conductive-bridging random-access memory (RAM), phase change memory, spintronic devices, superconducting electronics, optical implementations and organic electronics [Nawrocki et al., 2016, Schuman et al., 2017, Cheng et al., 2019, Lee et al., 2019].

The most mature of these new materials are memristors [Jo et al., 2010, Li et al., 2018]. A memristor is a two-terminal non-volatile memory with history dependent conductivity. The history dependence of the resistance resembles synaptic plasticity, and hence memristors are mostly used as synapses in neuromorphic chips. They feature low power consumption and fast read-write speed but suffer from high fixed-pattern noise and cycle-to-cycle variability. Further technical problems limit the maximum size of memristor arrays. Recently published memristor chips have on the order of 10^3 to 10^4 memristive devices on a single chip, e.g. in Cai et al. [2019].

The arguably most successful branch of neuromorphic engineering is the development of *event-based sensors* for vision, auditory and olfactory signals [Vanarse et al., 2016]. These devices convert the detected signals directly into output spike-trains. For example, the Dynamics Vision Sensor (DVS) sensor detects changes in luminosity and turns them into ON and OFF events (spikes). It excels at low latency and high sample rate compared to conventional frame-based cameras. Some of the neuromorphic sensors are commercially available [iniVation, 2020].

2.3.3 Computational models for neuromorphic hardware

Neuromorphic hardware requires appropriate computational models to realize its potential advantages. Creating programs is fundamentally different from programming software on conventional CPUs. It means designing neural networks and corresponding learning schemes using the available neuron models and other resources on the respective neuromorphic hardware. Proposed networks use a huge variety of neuron models and coding schemes, see for example Tavanaei et al. [2019] and references therein. It is still a challenging task to create models that can both fulfill the computational task on a state-of-the-art level, compared to for example machine learning solutions on GPGPU, and respect the constraints of the neuromorphic platform (see e.g. chapter 3). This might change in the future if the basic computational paradigms are clearly established. Then programming on a higher abstraction level should be available.

Inspired by the review by Pfeiffer and Pfeil [2018], we identify four main methods to set up models for neuromorphic hardware. We mainly concern spiking neuron models, because the majority of neuromorphic hardware uses spiking neurons, but similar considerations apply to rate-based models as well. Further, we only concern models for neuromorphic hardware, that is from the FPGA-based neuromorphic approach to the research of new materials. Creating models for CPUs, GPGPUs and machine learning accelerators is the task of mainstream machine learning research.

Hand engineered networks use predefined network architectures to fulfill a given task. The main aspect is that the parameters of the network are not changed via learning. Examples for these kind of algorithms are realized in hard-wired navigation tasks and robotic control tasks [Blum et al., 2017, Cartiglia et al., 2018, Billaudelle et al., 2019b], in solving constraint-satisfaction problems [Fonseca Guerra and Furber, 2017, Steidel, 2018], in the Neural Engineering Framework [Eliasmith and Anderson, 2004], in the cellular neural networks [Roska, 2007] and in purpose-built spiking algorithms [Severa et al., 2016]. Learning as usually considered in neural networks in machine learning (section 2.1.1) is missing from these models. Hand engineered solutions are most suited for digital neuromorphic hardware; in mixed-signal hardware we have to mitigate the effect of fixed-pattern noise.

Conversion methods use already trained artificial neural networks (ANNs) and convert them into spiking neural networks using one of the several conversion techniques, reviewed in Pfeiffer and Pfeil [2018]. This approach has the advantage that it can immediately use the full available toolkit of deep learning. However, most methods use rate-based coding, meaning that spikes merely represent imperfect proxies of real-numbered rates, and do not use information in the timing of spikes. This deems the spiking implementation intrinsically inefficient and puts a burden on the neuromorphic hardware when comparing to conventional machine learning solutions. This approach does not explicitly mitigate the disruptive effect of fixed-pattern noise.

The *chip-in-the-loop* paradigm suggests a training loop between the neuromorphic hardware and a conventional hardware, reviewed in Pfeiffer and Pfeil [2018]. Model execution, for example feed-forward inference, is done on the neuromorphic hardware and parameter updates are calculated on the conventional hardware based on the output of the hardware and on a differentiable model. A positive impact of this approach is that it implicitly mitigates the fixed-pattern noise and other distorting effects. However, the execution can be tedious, and the time and power consumption could be dominated by the I/O processes between the two platforms. Sometimes, chip-in-the-loop methods are combined with conversion methods, for example in Schmitt et al. [2017] and Kungl et al. [2019].

Learning using *on-chip local learning rules* is currently the targeted long-term goal of models for neuromorphic hardware. Using on-chip or close to chip features — such as the local plasticity — could both maximize the speed and power efficiency by minimizing the I/O requirements and mitigate disturbing effects. Designing local learning rules is challenging on the theoretical side, which is related to the challenge of learning in biological systems[9] (section 2.2.3). Especially, we lack good mechanistic models for deep learning — or equivalently powerful learning — using local learning rules (see chapter 5). Existing implementations of local learning rules are usually based on pure shallow learning [Pfeil et al., 2013b, Kreiser et al., 2017, Wunderlich et al., 2019, Feldmann et al., 2019]. On the hardware side, plasticity rules are often hard-wired and only implement a single model, such as the classic Bi-Poo STDP [Bi and Poo, 1998] or some type of short-term plasticity. This greatly reduces the flexibility of modeling. Recently, BSS-2 [Friedmann et al., 2017] and Loihi [Davies et al., 2018] added more flexibility to their plasticity mechanisms to enable an easier implementation of custom on-chip plasticity rules.

Finally, we mention the learning-to-learn approach [Hochreiter et al., 2001], which proposes to encapsulate the learning model into an outer-loop of gradient free optimization on a family of tasks. The approach mimics slow evolutionary and developmental processes to endow the underlying model with transfer-learning capabilities, as it was shown in spiking neural networks [Bellec et al., 2018]. The learning-to-learn approach could greatly benefit from accelerated neuromorphic hardware because the outer loop requires several iterations of the inner loop [Bohnstingl et al., 2019]. The learning-to-learn approach does not fit into the list, it is rather a proposed mechanism to encapsulate the models, for example the on-chip learning or the chip-in-the-loop paradigms.

2.3.4 Current challenges of neuromorphic engineering

Neuromorphic engineering has to tackle several problems in order to reach further progress and to live up to the promises that have been driving the field since its birth in 1980s. Here, we propose a set of five challenges for the coming years focusing on modeling and application, without claim to completeness. These challenges are connected among each other and they are also connected to similar challenges in other fields.

Finding the "killer" application: Machine learning research dismisses neuromorphic as the future of deep learning, based on the lack of proven advantages in any practical applications [LeCun, 2019]. Until now, computing with neuromorphic hardware and computing with spiking neurons failed to find an application where a clear advantage over conventional solutions would be apparent. It is expected that the "killer" application will use the temporal information stored in spikes and sparse computations. Davies et al. [2018] argue to have found one in spike-based sparse coding.

Lack of good benchmarks: Connected to the previous challenge, neuromorphic computing lacks widely accepted benchmarks as there are in conventional computing or in machine learning [Davies, 2019]. The usually reported

[9]Note, that the requirement of locality is only similar but not the same for biology and neuromorphic hardware.

benchmarks are either too microscopic such as synaptic operations per time or they are tailored for mainstream machine learning such as ImageNet [Deng et al., 2009] and CIFAR [Krizhevsky et al., 2009]. The latter two consist of static images containing no temporal information. Davies [2019] suggests a set of benchmarks consisting of several applications where the temporal information is of relevance. An example is the Heidelberg spoken digits dataset, in which the sound is pre-processed with a model based on the human cochlea [Cramer et al., 2019b].

Powerful and robust algorithms for neuromorphic hardware: Powerful algorithms are required to solve complex tasks or the proposed broad suite of benchmarks. This challenge is closely related to the need for powerful mechanistic models in biology section 2.2.3. It is not clear, which path the progress should pursue. Two main rivaling approaches seem to dominate among the possibilities. On the one hand, we could strive for an approximation of backpropagation based on local learning rules (similarly as in chapter 5). On the other hand, we could develop brain-like structured hierarchical models closely trying to mimic the brain's ability to learn from a few examples combined with past experience.

Further network scalability: The existing implemented neural networks on neuromorphic hardware [Esser et al., 2016, Schmitt et al., 2017, Liu et al., 2018, Kungl et al., 2019] are mostly smaller than contemporary neural networks in mainstream machine learning [He et al., 2016]. Further scalability of the networks is a challenging task for hardware design, for system software development and for model development. From the perspective of hardware design, neuromorphic chips follow the two dimensional design of integrated circuits, which greatly restricts the possible connections compared to the three dimensional structure of the brain. Three dimensional integrated circuits might be a possible way of further development [Pavlidis et al., 2017]. From the system software perspective, efficient mapping algorithms are required that can map the user-defined neural networks to the hardware while respecting the constraints of the substrate [Galluppi et al., 2012, Lin et al., 2018, Passenberg, 2019]. From the model-development perspective, we require network models that can be easily scaled and distributed over the available neuromorphic hardware. Popular models, such as convolutional neural networks [LeCun et al., 1989], are ill-suited for this purpose due to their dense connectivity between the layers.

Unified computing framework for principled development: According to Jennifer Hasler [Hasler, 2016]: "Analog computation seems to be a bottom-up design approach practiced by a few artistic masters." Both hardware development and model development are rather explorative and lack a unified framework. Establishing such a framework could help defining the advantages of using analog and mixed-signal devices. Further, such a framework could open the way to a more principled and standardized hardware development.

Improving supporting software for accessible usage: A vital component of neuromorphic hardware is the supporting software toolchain. It enables the high-level programming of the hardware, the mapping of networks to the substrate and debugging, reviewed in Schuman et al. [2017]. Currently, most published experiments on neuromorphic hardware were conducted by either the designers themselves or in close collaboration with them, for example in Esser et al. [2016], Kreiser et al. [2017], van Albada et al. [2018], Kungl et al. [2019], and see appendix A.2 as well. In order to reach out to non-expert users outside the community, neuromorphic engineering needs supporting software that reduces the barriers to entry for using neuromorphic hardware. For example, the PyNN simulator-independent neural network description language [Davison et al., 2009] aims to provide a common high-level Application Programming Interface (API) for designing experiments on different back-ends, including diverse neuromorphic hardware.

3 Accelerated Bayesian inference on the BrainScaleS-1 neuromorphic platform

The content of this chapter was published in Kungl et al. [2019]. Here, we follow the publication but we give a more detailed description of the project for the sake of clarity and completeness.

The aggressive development of microchip production following Moore's law is becoming more and more capital intensive by the day [Khan et al., 2018]. At the same time, in the ever finer lithographic process, physical effects will become more and more difficult to overcome with heat production and quantum effects being the main challenges. Further, the so-called von-Neumann bottleneck between the memory and the processing unit limits the speed of conventional CPUs inherently by their design. Neuromorphic engineering promises to create massively parallel non-von-Neumann architectures that could overcome the limitations of CPUs using inspiration from the mammalian nervous system (section 2.3).

One particular subset of the neuromorphic devices implements the idea of "physical modeling". Instead of calculating the dynamics of the neurons, these systems instantiate distinct circuits and use the physics of the substrate to emulate the dynamics of the neurons and synapses [Mead, 1990, Indiveri et al., 2006, Schemmel et al., 2010, Jo et al., 2010, Pfeil et al., 2013a, Qiao et al., 2015, Chang et al., 2016, Feldmann et al., 2019]. In a somewhat simplified view: Instead of solving the differential equations governing the dynamics of the system, these systems let the dynamics evolve, hence "physics solves itself". These systems promise fast computation and/or low power/energy consumption (section 2.3.2). However these advantages come with drawbacks: variability in the manufacturing process introduces variation between the implemented neurons (fixed-pattern noise), and temporal noise is always present on the circuits. These two effects limit both the range and the effective resolution of the parameters. The challenge lies in finding/designing models that can perform powerful computations using the available resources (neuro-synaptic models and parameter range), while staying robust against the distorting variations (fixed-pattern and temporal noise). Just as in the case of the devices themselves, it is worth looking for inspiration in neuroscience because the brain has to cope with similar (although certainly not identical) constraints and challenges.

In the last decade, experimental evidence has been gathered [Berkes et al., 2011, Pouget et al., 2013, Orbán et al., 2016, Haefner et al., 2016] about the probabilistic information processing mechanisms of the brain. The Bayesian brain hypothesis [Doya et al., 2007] suggests that the brain implements a probabilistic computation according to a realization of Bayesian statistics.

In this picture, during learning the brain shapes the underlying probability distribution to be a model — bias in the Bayesian terminology — of the environment. We can interpret perception (or any reasoning) according to Bayes' theorem. Consider the example of an animal living in the jungle. During its lifetime, it learns to avoid the tiger (dangerous predator), and hence learns in its internal model that seeing black and orange stripes in the bush could mean the presence of a tiger. This serves as its bias accumulated via learning. In perception, the input — e.g. an image or sounds — serves as posterior, which is then combined with the previous knowledge (bias) to deduce the likelihood of an observation.

Theories of neural sampling [Buesing et al., 2011, Hennequin et al., 2014, Aitchison and Lengyel, 2016, Petrovici et al., 2016, Kutschireiter et al., 2017] suggest frameworks for particular forms of Bayesian information processing, in which the dynamics of neural network realizes sampling from an underlying probability distribution. The dynamics of the neurons are considered as a physical interpretation of the sampling process. This interpretation suggests that the inherent variability of biological neurons observed *in-vivo* [Mainen and Sejnowski, 1995, Reinagel and Reid, 2002, Toups et al., 2012, Masquelier, 2013] is not a disturbing nuisance that has to be corrected for — via population coding or via a different error correction procedure — but is rather a resource and a hallmark of the ongoing probabilistic computation. Learning in such a framework corresponds to shaping the underlying probability distribution to model the statistics of the environment/dataset, analogously to the jungle-animal example and to unsupervised learning in machine learning (section 2.1.2).

In this project, we present the scalable implementation of sampling with leaky integrate-and-fire neurons [Petrovici et al., 2016] on the BrainScaleS-1 system [Schemmel et al., 2010] an accelerated mixed-signal analog neuromorphic platform. The variability of the analog parameters can be compensated and incorporated into the network structure with an appropriate training procedure. The feasibility of the implementation is verified by sampling from low-dimensional arbitrary probability distributions (section 3.4.1). Furthermore, we demonstrate the capabilities of the approach on standard datasets by solving classification and pattern completion tasks (section 3.4.2). After setup and learning, the entire network is fully contained on the neuromorphic hardware; communication is only used to insert external stimulus (here images) to the network and to receive spike responses from parts of the network. This work contributes to the development of beyond von-Neumann computation using (neuro-inspired) physical model systems while coping with their inherent limitations and benefiting from their advantages at the same time.

3.1 The BrainScaleS-1 large-scale neuromorphic platform

The BrainScaleS-1 system (BSS-1) is a platform for accelerated emulation of large-scale spiking neural networks [Schemmel et al., 2008, 2010] developed in the BrainScaleS Project [BrainScaleS, 2011] and the Human Brain Project [Markram et al., 2011b]. The aim of the platform is two-fold: On the one hand, it should enable the accelerated emulation of large spiking networks. This feature aims to accelerate simulations that take a considerable amount of time, such as simulations of life-long learning or evolution. On the other hand, BSS-1 offers a research platform to discover, prototype and test novel computation paradigms, which aim to outperform conventional solutions in the long run regarding speed and/or energy consumption.

The BSS-1 is a spiking, mixed-signal, analog, accelerated neuromorphic system. The neuron model at the heart of the system is spiking (opposed to rate based models, section 2.2.2). Analog circuitry realizes the dynamics of the emulated neuron model, but once the neurons spike, the action potentials are communicated through a digital circuitry between the neurons as logical events. In this sense, we say that the BSS-1 is a mixed-signal system. The emulation of the neural dynamics is accelerated compared to their biological counterparts; the realized time-scales — such as the membrane time-constant or the synaptic time-constant — are shorter on the neuromorphic chip than in biological neurons. The acceleration factor can be varied between 10^3 and 10^5, but in this project and as a default setting it is used with 10^4-fold acceleration. We refer to time always in the biological equivalent time unless specified otherwise; 1 ms biological equivalent time corresponds to 0.1 μs wall-clock time emulation on BSS-1.

The BSS-1 emulates the so-called Adaptive Exponential Integrate-and-Fire (AdEx) neuron model [Brette and Gerstner, 2005] in a highly modular design (figure 3.1 A). The AdEx model is a two-dimensional extension of the LIF model (section 2.2.2) with additional adaptation and exponential terms. The exponential term emulates the sharp depolarization ramp of the stereotypical action potential (section 2.2.1). The adaptation term models a spike triggered and continuously relaxing change of excitability of the neuron. Depending on the settings, consecutive spikes can be elicited more or less easily. The AdEx can reproduce several firing regimes of a single neuron, such as tonic spiking, spike frequency adaptation and chaotic spiking; these have been realized on prototype chips of the BSS-1 [Tran, 2013]. Due to the modular design of the circuit, the single components can be independently configured or turned-off, for example to emulate purely the LIF model. In this project, we only use the LIF features of the neuron model, for further details on the AdEx components we refer to the works of Schemmel et al. [2010], Millner [2012], Kleider [2017]. The parameters of the neurons are stored in on-chip analog parameters storages, so-called Floating Gates [Lande et al., 1996, Loock, 2006, Srowig et al., 2007, Kononov, 2011, FGs], with 10-bit nominal resolution.

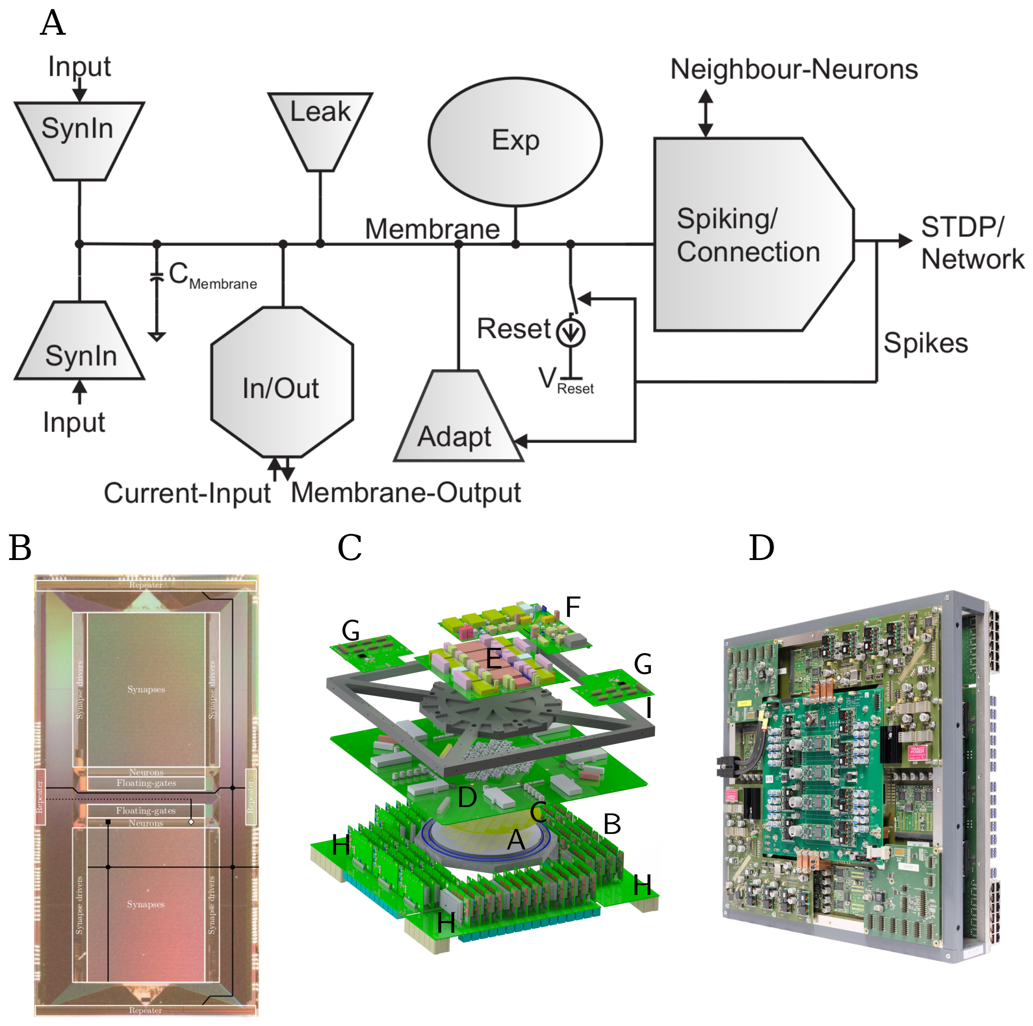


Figure 3.1: The BrainScaleS-1 system. (A) Schematics of the implemented AdEx model on the BSS-1. Due to the highly modular design, single components of the neuron can be configured and turned-off independently. In this project, we use this feature to emulate the LIF model. Image taken from Millner [2012]. **(B)** The High Input-Count Analog Neural Network (HICANN) chip is the heart of the system with 512 neurons and 112 640 synapses. On-chip routing network occupies the outer parts of the chip. Image taken from Millner [2012]. **(C)** Schematics of a module of BSS-1. (A) The wafer module is connected via (B) the positioning mask and (C) elastomeric connectors to the (D) the main PCB board. Further support PCBs (E,F) provide power supply and (G) access to the membrane trace measurements. On the four edges (H) inter-wafer and host connectivity is provided with USB slots and Gigabit-Ethernet slots, respectively. An aluminum frame gives mechanical stability (I). Image taken from Schmitt et al. [2017]. **(D)** Image of a fully assembled wafer module, taken from Schmitt et al. [2017].

Communication between the neurons takes place via action-potentials. If a neuron generates an action-potential according to its model, then this spike is registered by the digital part of the circuitry as a logical event. The resulting package is transmitted digitally to the post-synaptic neuron, which receives it via its synaptic circuit. The weight of the synapses is stored in a 4-bit static random-access memory (SRAM) cell for each synapse individually. The synaptic circuit emulates the COBA synapse model with an exponential synaptic (conductance) kernel (section 2.2.2). The synaptic circuits feature both short-term plasticity (STP) based on a reduced version of the Tsodyks-Markram short-term plasticity model [Tsodyks and Markram, 1997] and a form of STDP based on the classic STDP model [Bi and Poo, 1998].

The BSS-1 system has a hierarchical organization. The basic building block is the HICANN neuromorphic chip (figure 3.1 B). We only consider the version HICANNv4.1 chip, which was used in the experiments; for differences between this and older HICANN generations see Koke [2017]. One HICANN chip hosts 512 dendritic membrane units (denmems), each of them containing the circuits of the AdEx model with an excitatory and an inhibitory synaptic circuit. 220 synaptic circuits belong to each of the denmem units, amounting to 112 640 synaptic circuits per HICANN. Several denmem units can be combined to form larger neurons with more potential incoming synapses (pre-synaptic partners). In the extreme case, 64 denmems can be combined to form a single neuron with 14 080 pre-synaptic partners. Hence, there is a trade-off between the number of distinct incoming synapses to the neurons and the number of used neurons.

The 384 HICANN chips of a single wafer module are organized into 48 reticles with 8 HICANNs in each of these reticles. Each reticle has a corresponding FPGA that is responsible for experiment control, and the communication between host and neuromorphic chips on the reticle. This modular organization makes it possible that several single reticle experiments can be run in parallel on a wafer module. The HICANN chips are connected in a post-processing step [Zoschke et al., 2017]. Post-processing is required to realize wafer-scale integration. Opposed to the production of single chips, where the individual chips are cut out from the silicon wafer, on the BSS-1 system our aim is to connect them. Because of restrictions of the standard CMOS lithography procedure, only individual reticles can be produced with a single lithography mask [Zoschke et al., 2017], hence a post-processing step is required to create inter-reticle connections. Vertical and horizontal buses, called the layer 1 (L1), enable the communication between the HICANN chips. Similarly layer 2 (L2) is designed for direct communication between wafer modules.

By system design intention, experiments on BSS-1 are written in the PyNN simulator-independent neural network description language [Davison et al., 2009]. The BSS-1 implementation of PyNN is called the PyHMF package. The abbreviation HMF stands for Hybrid Multiscale Facility [Müller, 2015]. PyNN provides a common interface for several neural network simulation back-ends including simulators written for CPUs and/or GPUs and neuromorphic hardware platforms. PyNN gives a high-level control over the system; low-level control is (in an ideal case) handled by the back-end specific implementation and the own software-stack of the back-end. When working with the BSS-1 system the end-user should only use the PyNN interface to design the emulations. In the background the BSS-1 software-stack converts the defined network over several software layers into an executable hardware experiment description, runs the emulations and retrieves the measured data (figure 3.2). The complete operation system behind BSS-1 system is described in Müller et al. [2020b].

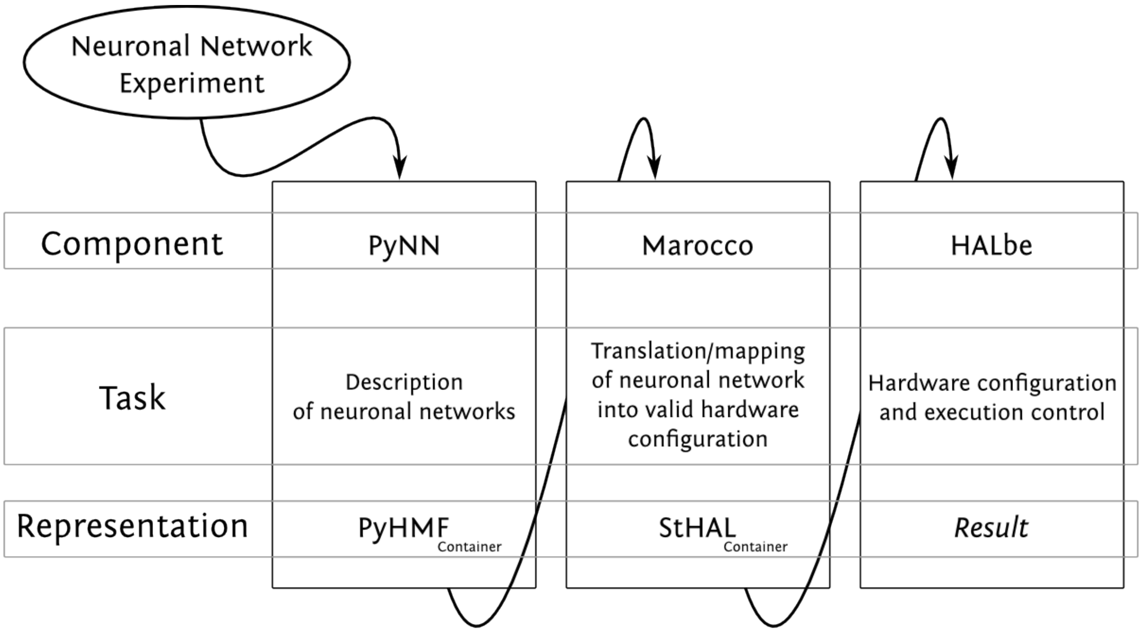

Neuronal Network Experiment
Component
PyNN
Marocco
HALbe
Task
Description of neuronal networks
Translation/mapping of neuronal network into valid hardware configuration
Hardware configuration and execution control
Representation
PyHMF Container
StHAL Container
Result

Figure 3.2: The BrainScaleS-1 system software pipeline. The described experiment and neural network is pipelined through several layers of the software-stack with decreasing degree of abstraction and increasing proximity to the hardware. By design, the user of the system should only see the PyNN [Davison et al., 2009] description and its BSS-1 specific implementation PyHMF. The complete software stack is discussed in detail in Jeltsch [2014] and Müller [2015]. Image taken from Müller [2015].

3.1.1 Challenges for the user on the BrainScaleS-1 platform

The accelerated emulation on BSS-1 comes with a price tag. The user of the system has to overcome several challenges that mostly originate in the idea of physical emulation, but which are not necessarily unique to BSS-1. We discuss these challenges with a focus on BSS-1 at the current status of commissioning (as of 2019 August), but highlight that some of these limitations are inherent in other systems as well.

If we understand BSS-1 as an accelerated neural network simulator — that is a replacement/alternative to e.g. BRIAN [Stimberg et al., 2019] or NEST [Gewaltig and Diesmann, 2007] — then we face the fact of limited controllability. The parameters on BSS-1 can be set only up to a limited precision, which is much less than the typical limiting machine precision for CPU simulators. On BSS-1 the analog neuron parameters can be set with a resolution of 10bit stored on Floating Gate (FG) analog memory cells [Lande et al., 1996, Loock, 2006, Millner, 2012], while the digital synapse weights have a resolution of 4bit stored on dedicated static random-access memory (SRAM) cells. The limited resolution is inherent to both analog and digital neuromorphic systems, as each parameter has to be stored close to its application point, for example synapse weights at the synapses. There is a trade-off between the resolution of the parameters and the density of neuronal and synaptic circuits that can fit on a chip.

The emulation precision is further restricted by the presence of different types of noise. Fixed-pattern noise is the deviation between realized transistors and other circuit elements due to manufacturing limitations. Such noise is constant in time; and effectively, it causes inhomogeneity between the neurons. Fixed-pattern noise can be compensated by calibration [Schwartz, 2013, Koke, 2017, Kleider, 2017] but cannot be reduced completely. Naturally, fixed-pattern noise is only relevant for analog systems, since typically in digital systems the discrete values (1 and 0) are separated by a large margin, and unlike the analog case there are correction methods for bit-flip errors. We collectively call any type of noise that is time-dependent temporal noise. Categories of temporal noise are the diverse disturbances, such as thermal noise, changes in the environmental temperature and variations in the supply voltage. They all appear on the membrane potential of single neurons and cause variations during an experiment and between distinct experiments. Figure 3.3 shows an example trace on the BSS-1 showing the strong temporal noise observed on the membrane potential. More precisely, it shows an overlay of the temporal noise on the membrane potential and the read-out noise. Analog and asynchronous digital systems are affected by this unintentional temporal noise. In synchronous digital systems with a single clock (or several aligned clocks), temporal noise is eliminated by design, by the discretization of time via the clock. On BSS-1 a particular source of temporal noise is the trial-to-trial variation of the FGs, called FG variations [Kononov, 2011, Kungl, 2016]. The writing precision of the FG is limited and two consecutive rewrites with the same target values result in different stored values in the FGs. The magnitude of these variations on the BSS-1 system led to the established method, that analog parameters are stored only once for experiments and later on, only the digital values are updated to realize for example learning [Schmitt et al., 2017]. This approach is similar to the chip-in-the-loop approach in section 2.3.3.

On every neuromorphic platform the emulated network has to be mapped from the abstract description — for example defined in PyNN — to the corresponding hardware components. This procedure is collectively called mapping. Ideally, the mapping procedure should create from the abstract network description a hardware graph that respects all the restrictions imposed by the system while keeping the user-defined components together with their connections. The diversity of the hardware constraints and the great variability of the emulated networks make the implementation of a generic mapping algorithm a difficult task; an ideal and per-

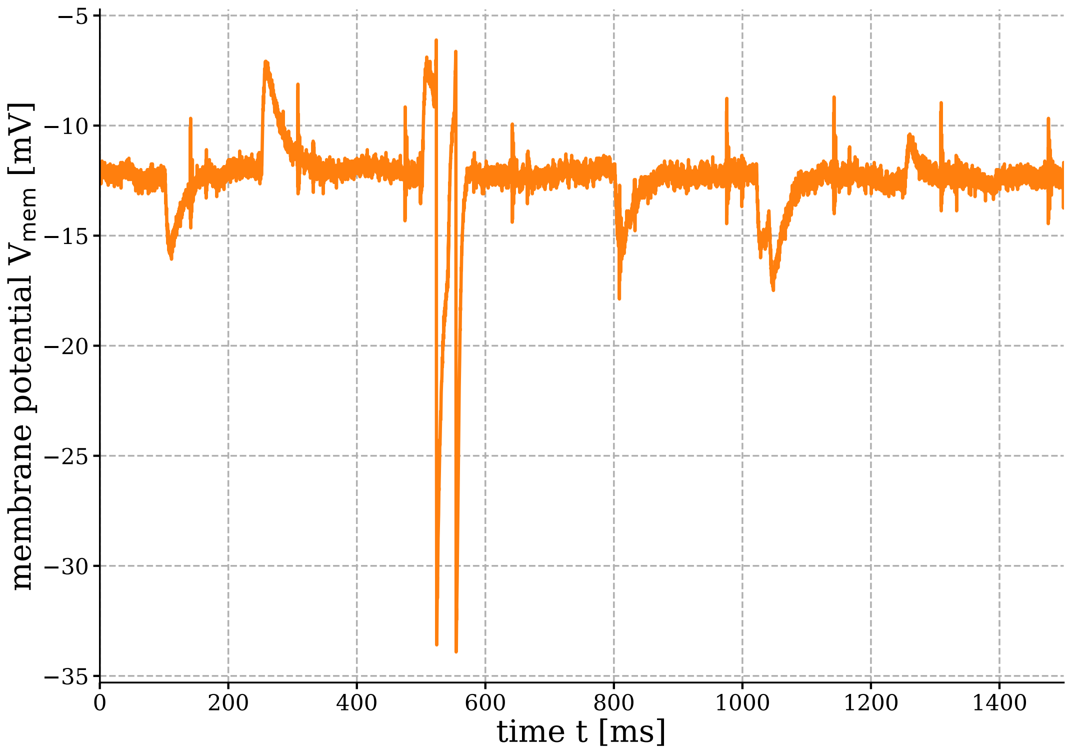


Figure 3.3: Example trace on the BSS-1. We can observe the strong temporal noise on the membrane potential, but it is a hard task to distinguish noise truly on the membrane from noise originating on read-out circuits. Note the difference between the realized resting potential and the user-set $E_{\text{leak}} = -20\,\text{mV}$, which is a result of the fixed-pattern noise. Finally, there is a periodic disturbance with a period of approximately 100 ms of unknown origin. This experiment was conducted without calibration of the neuron parameters.

fect mapping is in fact an NP-complete problem [Cook, 1971, Jeltsch, 2014]. There are no canonical solutions to address the mapping-problem. The software *marocco* from the BSS-1 software-stack uses a combination of heuristics and fall-back to standard algorithms [Jeltsch, 2014, Passenberg, 2019]. At the current state of commissioning, the *marocco* software can handle networks on the order of hundreds of neurons without losing synapses (figure 3.4). The challenge of mapping applies to all neuromorphic systems, even large scale software simulations suffer from this problem, however clocked simulators can sacrifice simulation speed for more synaptic connections.

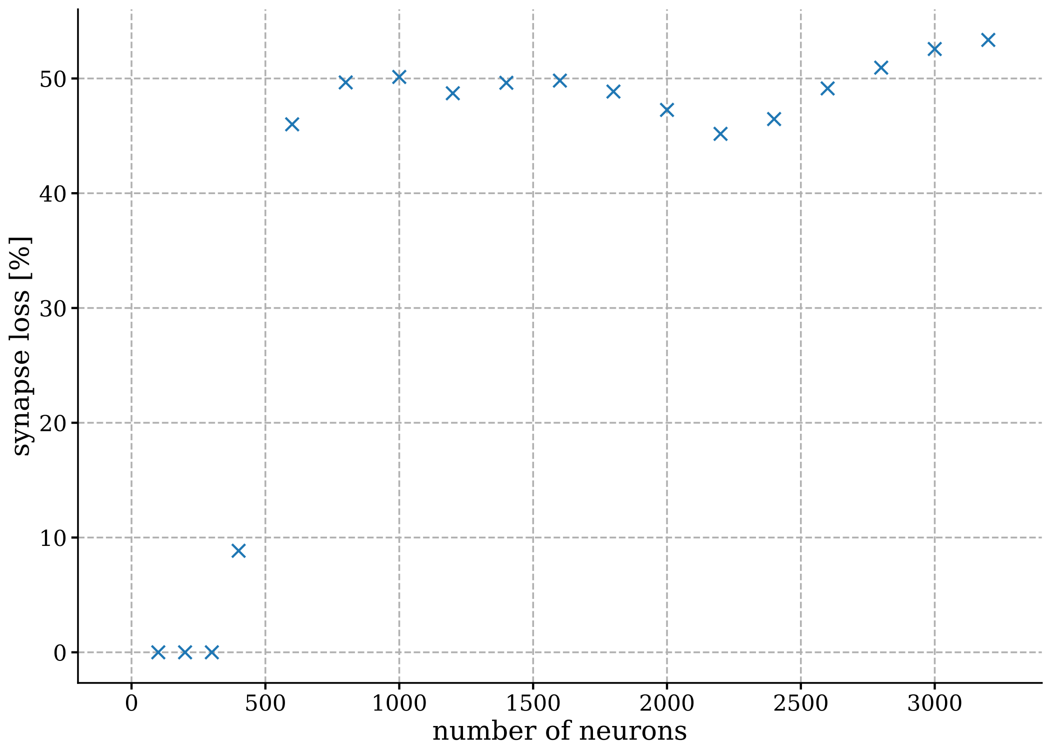


Figure 3.4: Synapse loss during mapping of a random network. Relative synapse loss as a function of number of neurons in a random network with 10 % connectivity. The figure exemplifies the problem of mapping large-scale networks onto the BSS-1 system. The figure was made from the system benchmark repository *brainscales-benchmarks* and with the *nmpm_software/2019-08-21* (appendix B.3).

The communication bandwidth is limited between the hardware and its environment, which limits the rate at which spikes can be read out or sent into the system. The main bottleneck is between the FPGAs and the wafer. The input and the output ratio of spikes are limited each to 1780Hz per HICANN chip and 12800Hz for a reticle [Müller, 2015], with rates given in the biological domain. To some extent, the user can circumvent this bottleneck by placing the inputs on other HICANNs and routing them via the on-wafer communication network.

The bandwidth of the on-chip communication — 50MHz HICANN to HICANN [Müller, 2015] — is not a limiting factor compared to the external bandwidth and the limitations imposed by the mapping.

Finally, a minor but often overlooked issue: Compared to a simulation the network does not have a well-defined initial state at the start of the emulation. As soon as the parameters of the network are configured, the neurons and synapses evolve according to their own dynamics. The user has to make sure to prepare the network into the desired initial state by choosing appropriate parameters (low leakage potential for a quiescent network) or strong external input. Alternatively, we can say that the BSS-1 requires applications that do not need well-specified initial conditions.[1]

All the constraints above impose serious challenges for proposed models and they implicitly test the viability of the models not only for neuromorphic application, but for biological plausibility as well, since biological substrates feature similar (but not identical) constraints.

3.2 Theoretical background — sampling with neurons

The theory of sampling with leaky integrate-and-fire neurons (LIF-Sampling) [Petrovici et al., 2013, 2016, Petrovici, 2016] can be introduced from several directions depending on the focus of the project. Here, we concentrate on the applicability, requirements and potentials of framework on the BSS-1 system. First, we introduce the concept of Boltzmann machines [Hinton et al., 1984], a probabilistic model and neural network from the field of machine learning. We discuss methods of sampling from the probability distribution defined by a Boltzmann machine. Based on the analogy between neurons of a Boltzmann machine and biological neurons, we introduce neural sampling [Buesing et al., 2011]. Finally, we conclude the theoretical background with the LIF-Sampling framework, which provides us a framework through which networks of LIF neurons approximately sample from a Boltzmann distribution.

In this project, we only give a brief introduction of LIF-Sampling that is necessary for understanding and replicating the implementation of sampling on BSS-1. In this logic, we follow a previous work on implementing sampling on BSS-1 system [Kungl, 2016]. For a comprehensive discussion of the framework we recommend the thesis Petrovici [2016] describing the framework in great detail, and the original publications [Petrovici et al., 2013, 2016].

3.2.1 Boltzmann machines: constraint satisfaction models with learning capabilities

A Boltzmann machine is a probabilistic generative model defined over N binary units with symmetric connections [Hinton et al., 1984]. These elements z_k are often

[1]Mind the similarity to experiments in quantum physics where the initial state of the system is also not granted but has to be prepared by the experimenter.

referred to as neurons both due to their loose similarity to biological neurons and because Boltzmann Machines are an example for neural network models in the field of machine learning. The two states of the neurons are called *on-state* for $z_k = 1$ and *off-state* for $z_k = 0$. The entire state-space of the Boltzmann machine is then $\Omega = \{0,1\}^N$. The probability distribution of the Boltzmann machine is defined by:

$$p(\mathbf{z}) = \frac{1}{Z} \exp\left[\frac{1}{2}\mathbf{z}^T \mathbf{W} \mathbf{z} + \mathbf{z}^T \mathbf{b}\right] \quad , \tag{3.1}$$

where Z is the partition function given by

$$Z = \sum_{\mathbf{z}} \exp\left[\frac{1}{2}\mathbf{z}^T \mathbf{W} \mathbf{z} + \mathbf{z}^T \mathbf{b}\right] \quad , \tag{3.2}$$

where $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ is the state vector, $\mathbf{b} = (b_1, b_2, \ldots, b_N)$ is the bias vector of the neurons and $\mathbf{W} = [w_{ij}]_{i,j \in \{1,2,\ldots,N\}}$ is the connection matrix. The bias $b_i \in \mathbb{R}$ gives the "preference" of the neuron to be in the on-state, or equivalently to be active. The connections w_{ij} correspond loosely to the expected coactivation of the neurons, that is a positive w_{ij} means that the neurons are more likely to be active together. However, w_{ij} is only indirectly connected to the correlation of activity between the neurons. Self-connections are excluded $w_{ii} = 0, \forall i \in \{1, \ldots, N\}$; they would correspond to extra bias values. The connection between each pair of neurons is strictly symmetric $w_{ij} = w_{ji}, \forall i,j \in \{1, \ldots, N\}$.

An intuitive insight into the mechanism of a Boltzmann machine gives the conditional probability distribution of a single neuron given the current state of the rest of the network. After elementary calculations we find:

$$\begin{aligned} p(z_i = 1 | z_{\setminus i}) &= \frac{p(z_1, \ldots, z_i = 1, \ldots, z_N)}{p(z_1, \ldots, z_i = 0, \cdots, z_N) + p(z_1, \ldots, z_i = 1, \ldots, z_N)} = \\ &= \frac{\exp(u_i)}{1 + \exp(u_i)} = \frac{1}{1 + \exp(-u_i)} \quad , \end{aligned} \tag{3.3}$$

where $z_{\setminus i} = (z_1, z_2, \ldots, z_{i-1}, z_{i+1}, \ldots z_N)$ is the vector of the neurons excluding z_i and the parameter u_i is given by

$$u_i = b_i + \sum_{j=1}^{N} w_{ij} z_j \quad . \tag{3.4}$$

First, we see that the probability to be in the on-state is determined by the linear sum of the own bias and the input received from the other active neurons. Additionally, we identify the logistic function $\sigma(x) = \frac{1}{1+\exp(-x)}$ as the non-linear transfer or activation function of the neurons. This form is already suggestive for a more biological neuron-based implementation. Second, from a computational point of view, the accessibility of the conditional distribution is suggestive to be used for a sampling mechanism. Third, from the point of view of physics

Boltzmann machines are Spin-glasses [Edwards and Anderson, 1975], which can be seen as a generalized form of the Ising-model. Analogously, we can understand Boltzmann machines as energy based systems with the energy function:

$$E(\mathbf{z}) = -\frac{1}{2}\mathbf{z}^T\mathbf{W}\mathbf{z} - \mathbf{z}^T\mathbf{b} \quad , \tag{3.5}$$

and the probability distribution:

$$p(\mathbf{z}) = \frac{1}{Z}\exp[-E(\mathbf{z})] \quad , \tag{3.6}$$

$$Z = \sum_{\mathbf{z}} \exp[-E(\mathbf{z})] \quad . \tag{3.7}$$

These equations are formally identical to Boltzmann distributions with $\beta = \frac{1}{k_\mathrm{B}T} = 1$ the inverse temperature. Hence, we can define a temperature for Boltzmann machines as well:

$$p(\mathbf{z}) = \frac{1}{Z}\exp[-\beta E(\mathbf{z})] \quad , \tag{3.8}$$

$$Z = \sum_{\mathbf{z}} \exp[-\beta E(\mathbf{z})] \quad . \tag{3.9}$$

This analogy and the close relationship to physics, and more precisely to spin-glasses, give rise to the name of Boltzmann machines.

Boltzmann machines and their extensions have found several applications in solving constraint satisfaction problems [Jonke et al., 2016, Fonseca Guerra and Furber, 2017], prediction of temporal sequences [Sutskever and Hinton, 2007], movement planning [Taylor and Hinton, 2009, Alemi et al., 2015], simulation of solid-state systems [Edwards and Anderson, 1975] and tackling quantum many-body problems [Carleo and Troyer, 2017, Czischek et al., 2018].

Sampling from Boltzmann machines: Gibbs sampling

To make Boltzmann machines actually calculate practical measures, we need to obtain the probability distribution $p(\mathbf{z})$ of the states for a given set of parameters $(\mathbf{b}, \mathbf{W})$. The number of states grows exponentially with the number of neurons, that is $|\Omega| = 2^N$, where the $|\cdot|$ symbol denotes the cardinality of a set, that is the number of elements in the set. The direct calculation of the probability distribution becomes rapidly infeasible. For example, a naive direct calculation on an off-the-shelf CPU is feasible up to approximately $N = 15$ neurons. To mitigate this problem we use sampling methods, more exactly Gibbs sampling [Geman and Geman, 1984].

Gibbs sampling (algorithm 3.1) is a type of Markov Chain Monte Carlo sampling which explicitly uses conditional distributions. Here, we only show Gibbs sampling, Markov Chain Monte Carlo Methods can be found in several textbooks, for example in Bailer-Jones [2017].

Algorithm 3.1: Gibbs sampling. A simple realization of Gibbs sampling. The selection of the next variable for updates typically varies according to the specific implementation.

```
Data: Vector $\mathbf{x}$ of length $N$, probability distribution $p(\mathbf{x})$
Result: Chain of samples from $\mathbf{x}$ distributed according to $p(\mathbf{x})$
start from given $\mathbf{x}^{(\mathbf{0})}$;
while required number of samples not reached do
    choose a $x_i^{(n)}$ from $\mathbf{x}^{(\mathbf{n})}$ randomly;
    calculate conditional probability $p(x_i^{(n)}|\mathbf{x}_{\backslash\mathbf{i}})$ ;
    generate $x_i^{(n+1)} \sim p(x_i^{(n)}|\mathbf{x}_{\backslash\mathbf{i}})$;
    obtain $\mathbf{x}^{(\mathbf{n+1})} = (x_1^{(n)}, \dots, x_{i-1}^{(n)}, x_i^{(n+1)}, x_{i+1}^{(n)}, \dots, x_N^{(n)})$
end
```

Geman and Geman [1984] proved that this algorithm produces a Markov Chain that converges towards the targeted distribution. Note, that the update sequence of the terms in $\mathbf{x}$ varies depending on the specific application.

We characterize the distance between two probability distributions with the D_{KL} introduced in Kullback and Leibler [1951]; we use it to measure and quantify the distance between the sampled and the targeted distribution in experiments (see also section 2.1.2). Here, we use the natural logarithm ln to preserve compatibility with previous related works [Petrovici et al., 2016, Leng et al., 2018, Jordan et al., 2019, Dold et al., 2019].

Restricted Boltzmann machines and learning with Contrastive Divergence

A fully-connected Boltzmann machine (where $\mathbf{W}$ is dense) is indeed not practical for machine learning applications because the state of the single neurons has to be updated in series, rendering the sampling procedure slow. Restricted Boltzmann Machine (RBM) [Smolensky, 1986] introduces constraints on the connectivity matrix $\mathbf{W}$ giving a bi-partite or two-layer graph structure to the network. The two layers are referred to as *visible* and *hidden* layer. The layers are connected to each other but lateral intra-layer connections are excluded. The connection matrix $\mathbf{W}$ has hence a block-wise structure. Alternatively, we can write the probability distribution of the RBM with separate visible $\mathbf{v} \in \Omega_{\mathrm{v}} = \{0,1\}^N$ and hidden states $\mathbf{h} \in \Omega_{\mathrm{h}} = \{0,1\}^M$. We use the notations $\mathbf{z} = (\mathbf{v}, \mathbf{h})$ and $\Omega = \Omega_{\mathrm{v}} \times \Omega_{\mathrm{h}}$:

$$\begin{aligned} p(\mathbf{z}) &= \frac{1}{Z} \exp\left(-E(\mathbf{z})\right) \quad , \\ E(\mathbf{z}) &= -\left(\mathbf{a}^T\mathbf{v} + \mathbf{b}^T\mathbf{h} + \mathbf{v}^T\hat{\mathbf{W}}\mathbf{h}\right) \quad , \\ Z &= \sum_{\mathbf{z}\in\Omega} \exp\left(-E(\mathbf{z})\right) \quad . \end{aligned} \tag{3.10}$$

The matrix $\hat{W} \in \mathbb{R}^{(N,M)}$ contains only the connections between the hidden and visible neurons. For the sake of simplicity, we drop the hat notation of the connectivity matrix. The visible layer represents the dataset, while the hidden layer can represent higher-order relationships in the data. In classification tasks the labels of the different classes are best imagined as part of the visible layer, although sometimes they are visualized as an additional third label layer. Because the labels are part of the dataset, the two visualizations are indeed mathematically equivalent.

The two-layered structure of RBM makes it possible that we can sample from the RBM layer-wise, in a parallel fashion. Learning in RBMs happens with the Contrastive Divergence (CD) algorithm [Ackley et al., 1987, Hinton, 2012]. The generative model of the RBM is given via the marginalization over the hidden variables

$$p(\mathbf{v}) = \sum_{\mathbf{h}\in\Omega_\mathrm{h}} p(\mathbf{v},\mathbf{h}) \quad . \tag{3.11}$$

In the language of machine learning CD is based on maximum likelihood learning (section 2.1.2):

$$\hat{\boldsymbol{\theta}} = \mathrm{argmax}_{\boldsymbol{\theta}} \left(\langle \ln p(\mathbf{v}|\boldsymbol{\theta}) \rangle_{p^*(\mathbf{v})} \right) = \mathrm{argmax}_{\boldsymbol{\theta}} \left(\sum_{\mathbf{v}\in\mathbf{S}} \ln p(\mathbf{v}|\boldsymbol{\theta}) \right) \quad , \tag{3.12}$$

where $\mathbf{S}$ represents the dataset to be learned, and $p^* = \frac{1}{|S|} \sum_{x_s \in S} \delta\left(\boldsymbol{x} - \boldsymbol{x}_s\right)$ represents the distribution corresponding to the dataset $\mathbf{S}$. In practice, we search for the desired $\boldsymbol{\theta}$ parameters via gradient descent. To derive the update rules, we first calculate:

$$\begin{aligned}
\frac{\partial \sum_{\mathbf{h}} \ln p(\mathbf{z})}{\partial w_{ij}} &= \frac{1}{\sum_{\mathbf{h}} p(\mathbf{z})} \sum_{\mathbf{h}} \frac{\partial}{\partial w_{ij}} \left[\frac{\exp(-E(\mathbf{z}))}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} \right] = \\
= \frac{1}{\sum_{\mathbf{h}} p(\mathbf{z})} \sum_{\mathbf{h}} &\left\{ \frac{v_i h_j \exp(-E(\mathbf{z}))}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} - \frac{\exp(-E(\mathbf{z})) \sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}})) \hat{v}_i \hat{h}_j}{\left[\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))\right]^2} \right\} = \\
&= \sum_{\mathbf{h}} \frac{p(\mathbf{z})}{p(\mathbf{v})} v_i h_j - \frac{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}})) \hat{v}_i \hat{h}_j}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} \frac{\sum_{\mathbf{h}} p(\mathbf{z})}{\sum_{\mathbf{h}} p(\mathbf{z})} = \\
&= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) v_i h_j - \sum_{\hat{\mathbf{z}}} p(\hat{\mathbf{z}}) v_i h_j = \langle v_i h_j \rangle_{\mathrm{data}} - \langle v_i h_j \rangle_{\mathrm{model}} \quad ,
\end{aligned} \tag{3.13}$$

and similarly for the bias term (here for the hidden bias)

$$\begin{aligned}\frac{\partial \sum_{\mathbf{h}} \ln p(\mathbf{z})}{\partial b_j} &= \frac{1}{\sum_{\mathbf{h}} p(\mathbf{z})} \sum_{\mathbf{h}} \frac{\partial}{\partial b_j} \left[\frac{\exp(-E(\mathbf{z}))}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} \right] = \\ = \frac{1}{\sum_{\mathbf{h}} p(\mathbf{z})} \sum_{\mathbf{h}} &\left\{ \frac{h_j \exp(-E(\mathbf{z}))}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} - \frac{\exp(-E(\mathbf{z})) \sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))\hat{h}_j}{[\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))]^2} \right\} = \\ &= \sum_{\mathbf{h}} \frac{p(\mathbf{z})}{p(\mathbf{v})} h_j - \frac{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))\hat{h}_j}{\sum_{\hat{\mathbf{z}}} \exp(-E(\hat{\mathbf{z}}))} \frac{\sum_{\mathbf{h}} p(\mathbf{z})}{\sum_{\mathbf{h}} p(\mathbf{z})} = \\ &= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) h_j - \sum_{\hat{\mathbf{z}}} p(\hat{\mathbf{z}}) h_j = \langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}} \quad .\end{aligned} \tag{3.14}$$

Here $\langle\cdot\rangle_{\text{model}}$ means the expectation value over the complete probability distribution $p(\mathbf{v}, \mathbf{h})$ of the RBM, and the $\langle\cdot\rangle_{\text{data}}$ term refers to the expectation value conditioned on a visible layer clamped to a sample from the dataset, that is $p(\mathbf{h}|\mathbf{v})$. Hence the resulting update rules for the weights and the biases are:

$$\begin{aligned}\Delta w_{ij} &= \eta \sum_{\mathbf{v}\in\mathbf{S}} \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right) \quad , \\ \Delta a_i &= \eta \sum_{\mathbf{v}\in\mathbf{S}} \left(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}} \right) \quad , \\ \Delta b_i &= \eta \sum_{\mathbf{v}\in\mathbf{S}} \left(\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}} \right) \quad .\end{aligned} \tag{3.15}$$

The above expression represents batch learning of the dataset. Dropping the summation leads to a maximum likelihood learning rule with stochastic gradient descent:

$$\begin{aligned}\Delta W_{ij} &= \eta \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right) \quad , \\ \Delta a_i &= \eta \left(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}} \right) \quad , \\ \Delta b_i &= \eta \left(\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}} \right) \quad .\end{aligned} \tag{3.16}$$

A practical problem arising from this learning rule is that both the data-term and the model-term can only be sampled with finite precision. Based on these equations the CD$_n$ learning rule [Ackley et al., 1987] proposes a practical simplification. It can be summarized in the following steps:

1. Obtain sample $\mathbf{s}$ from the dataset and clamp it to the visible layer, that is $\mathbf{v} := \mathbf{s}$. Sample the states of the hidden neurons with $p(\mathbf{h}|\mathbf{s})$ and obtain the values $\langle v_i h_j \rangle_{\text{data}}$, $\langle v_i \rangle_{\text{data}}$ and $\langle h_j \rangle_{\text{data}}$.

2. Let the BM freely sample for n steps alternating between $\mathbf{v}$ and $\mathbf{h}$ starting from the last sampled $\mathbf{h}$ from the first step. Obtain the values $\langle v_i h_j \rangle_{\text{model}}$, $\langle v_i \rangle_{\text{model}}$ and $\langle h_j \rangle_{\text{model}}$.

3. Update the values according to the learning rule in equation (3.16).

In this section, we gave an introduction of learning in RBMs that is required for this work. For a comprehensive review on training RBMs including extensions and modifications of the learning rule see Hinton [2012].

3.2.2 Neural sampling — sampling with spiking neurons

The model of sampling with still abstract but spiking neurons was first described by Buesing et al. [2011]. In their work, the authors propose a model where a network of abstract neurons samples from the probability distribution of a Boltzmann machine. The novelty of their work was that they explicitly included the spiking behavior and the refractory mechanism of the neurons. Further, with the mathematical equivalence to Boltzmann machines, this model not only provides a mechanistic model for sampling in the brain but it also connects machine learning to computational neuroscience. Hence, existing methods, for example annealing, can serve as inspiration to tackle questions in the neuroscience. We can motivate the model with the following observations:

- In biological neurons [Gerstner and Kistler, 2002b], after the production of an action potential the generation of another spike is prohibited for a certain τ_{ref} refractory time. Hence, it is tempting to model the neuron as a two-state system. Immediately after spike the neuron is in the refractory state and otherwise the membrane potential freely evolves while spike generation is possible.
- We can see neurons in the sense of a linear non-linear response model [Simoncelli et al., 2004]: The neuron sums up/integrates the incoming input linearly (for current based synapses) and produces a stochastic output via a non-linear transfer function. The concept of linear summation is similar to the idea of abstract membrane potential in Boltzmann machines (equation (3.4)) and the non-linear response to the sigmoid probability function to be in state 1 (section 3.2.1).
- The interaction between neurons takes place via all-or-nothing action potentials, and the generation of these action potentials is strictly coupled to the refractoriness. This mechanism closely resembles the Glauber-dynamics of sampling from a Bolzmann machine.

In the setup of neural sampling, we treat the time as a discrete variable increasing in time-steps of dt, such that the time is represented by a natural number $t \in \mathbb{N}$. In this sense the model is close to the Glauber dynamics of Gibbs-sampling because the model introduces time not as a naturally continuous variable but rather by relating the time-steps to characteristic time-constants in neural dynamics. We define the state $z_k(t)$ of neuron k at time t as 1, or on-state, if the neuron has spiked in the last $\tau_{\text{ref}} \in \mathbb{Z}^+$ time-steps and 0, or off-state, otherwise. τ_{ref} is the refractory time-constant of the neuron. In equation:

$$z_k(t) = \begin{cases} 1 & \text{if } t < t_s + \tau_{\text{ref}} \quad , \\ 0 & \text{otherwise} \quad . \end{cases} \tag{3.17}$$

Here is t_s is the timestamp of a spike generated by the k-th neuron. The connection matrix and the bias vector of the network are the same as in case of a

Boltzmann Machine (section 3.2.1). For the mathematical description of the dynamics we introduce the auxiliary variable $\zeta_k(t) \in \mathbb{Z}$ with $0 \leq \zeta_k(t) \leq \tau_{\mathrm{ref}}$. $\zeta_k(t)$ plays the role of a countdown variable of neuron k during its refractory time. The relation between ζ_k and z_k is given by

$$z_k(t) = \begin{cases} 1 & \text{if } \zeta_k \geq 1 \quad , \\ 0 & \text{if } \zeta_k = 0 \quad . \end{cases} \tag{3.18}$$

With the help of the auxiliary variable ζ_k now we can define the dynamics of the model as a first order Markov process (a description using directly z_k would not be Markovian):

- If a neuron is refractory, i.e. $\zeta_k(t) \geq 1$, then the auxiliary variable counts down:

$$\zeta_k(t+1) = \zeta_k(t) - 1 \tag{3.19}$$

- If a neuron is in the off-state or at the end of the refractory time in the on state, i.e. $\zeta_k \in \{0, 1\}$, then the neuron can fire with the probability:

$$p(\zeta_k(t+1) = \tau_{\mathrm{ref}} | \zeta_k \in \{0,1\}) = \sigma(u_k - \ln(\tau_{\mathrm{ref}})) \quad , \tag{3.20}$$

  where $\sigma(\cdot)$ is the logistic function

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad , \tag{3.21}$$

  and u_i is the abstract membrane potential given by Equation (3.4). If the neuron fired, then ζ_k is set to τ_{ref}. The state transition to spike from $\zeta_k = 1$ is necessary, so that continuous on-states are possible. The factor $\ln(\tau_{\mathrm{ref}})$ corrects for over-counting the on-state, for cases when $\tau_{\mathrm{ref}} > 1$. If no spike occurred then we set $\zeta_k = 0$ and $z_k = 0$.

- In the original publication, the neurons are updated sequentially just as in Gibbs sampling. If all the neurons are updated at the same time, mimicking a continuous time, then the model still provides reasonable results in practical tasks.

The above process is compactly depicted in figure 3.5. According to the dynamics, we can interpret the interaction between the neurons. When a neuron spikes, then it generates a rectangular PSP in its post-synaptic partners, because the effect of the pre-synaptic neuron on the membrane potential of the post-synaptic neurons is flat. The synaptic time-constant, that is the length of the PSP, exactly matches the refractory time of the pre-synaptic neuron. The interaction between any two neurons is strictly symmetric as the model is tailored for Boltzmann Machines. Buesing et al. [2011] proved that the described dynamics of the network constitutes an exact sampling from the probability distribution of the underlying Boltzmann Machine, and they showed that the model features biological phenomena like switching between modes when an ambiguous picture is presented to an observer (binocular ambiguity).

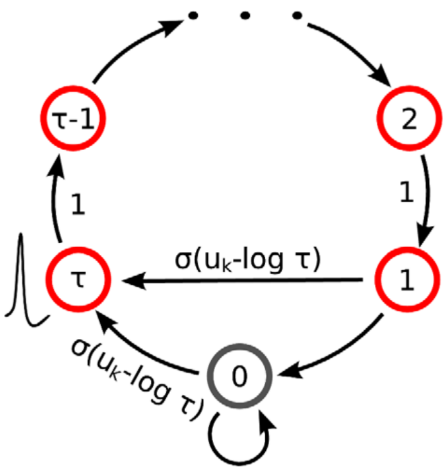


Figure 3.5: Dynamics of neural sampling: After the neuron spikes, the auxiliary variable ζ_k counts down during the refractory time. *Red circles* represent on-states and the *black circle* represents an off-sate. After the end of the refractory period, i.e. $\zeta_k \in \{0, 1\}$ the neuron can spike with the probability of $p_{\text{spike}} = \sigma(u_k - \ln(\tau_{\text{ref}}))$ (with $\sigma(\cdot)$ defined in Equation (3.21)). It can be shown that this Markov chain samples exactly from the probability distribution of a BM. Image taken from Buesing et al. [2011].

3.2.3 Sampling with leaky integrate-and-fire neurons

The theory of sampling leaky integrate-and-fire neurons with conductance based synapses [LIF Sampling, Petrovici et al., 2016] improves upon the ideas of Buesing et al. [2011] by introducing a neuron model with true time-continuous dynamics and explicit spiking mechanism. Similarly to neural sampling, LIF sampling maps Boltzmann machines to spiking neurons (figure 3.6 A-B). This step achieves a twofold improvement: 1) it greatly increases the biological fidelity and 2) by using the LIF neurons it connects Boltzmann machines to the neuron model of the BSS-1 system. The LIF Sampling theory was originally published in Petrovici et al. [2016] and it is described in detail in Petrovici [2016].

In order to realize the sampling scenario the model has to solve three main challenges:

1. Introduce stochasticity to the otherwise deterministic conductance-based leaky integrate-and-fire (COBA-LIF) model in a biologically plausible manner

2. Show that the activation or transfer function of the neurons approximates a sigmoid function. Note that the activation function of a deterministic LIF neuron is highly asymmetric.

3. Create a translation rule between the abstract weights and biases and their corresponding quantities in the LIF model.

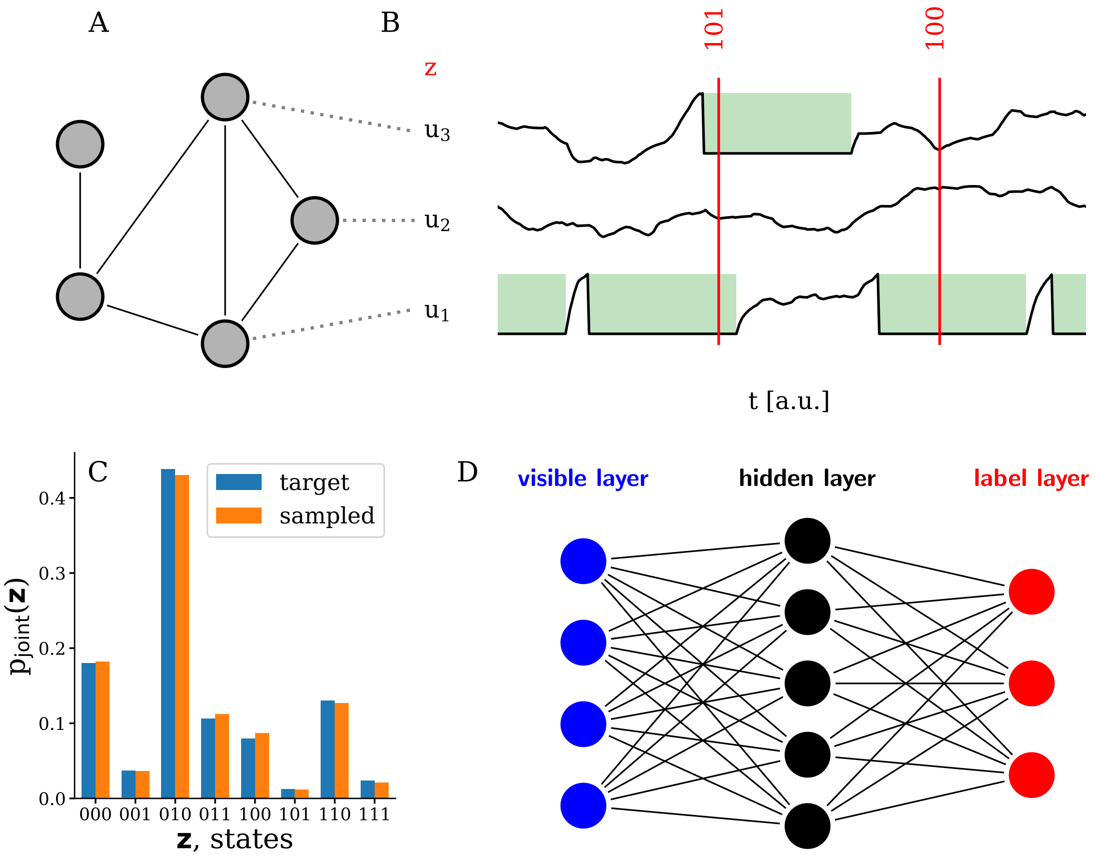


Figure 3.6: Sampling with leaky integrate-and-fire neurons. **(A)** Sketch of a spiking sampling network (SSN) with 5 neurons. Each line represents two reciprocal synaptic connections with equal weights. **(B)** Membrane potentials of three neurons in the network. Following a spike, the refractory mechanism effectively clamps the membrane potential for the duration τ_{ref} of the refractory time. During this time, the variable corresponding to neuron k is in the state $z_k = 1$ (marked in green). At any point in time, the state sampled by the network can therefore be constructed directly from its past spikes and the knowledge of the refractory time τ_{ref} of the neurons. **(C)** Probability distribution sampled by an SSN with three neurons as compared to the target distribution in a software simulation. **(D)** Based on this framework [Petrovici et al., 2016], hierarchical sampling networks can be built, which can be trained on real-world data. Each line represents a reciprocal connection (two synapses) between the connected neurons. Figure taken from Kungl et al. [2019].

To introduce stochasticity and a sigmoid activation function, each neuron receives high frequency Poisson noise both excitatory and inhibitory to the membrane. For the sake of simplicity, we assume that the synaptic time-constants of both noise types are equal $\tau_{\mathrm{syn}} = \tau_{\mathrm{exc}} = \tau_{\mathrm{inh}}$. We can rewrite the COBA-LIF equations section 2.2.2 in terms of effective memebrane potential u_{eff} and effective membrane time-contant τ_{eff},

$$\tau_{\mathrm{eff}} \frac{\mathrm{d}V_{\mathrm{mem}}}{\mathrm{d}t} = u_{\mathrm{eff}} - V_{\mathrm{mem}} \quad , \tag{3.22}$$

with u_{mem} the membrane potential and the effective membrane potential u_{eff} is given by:

$$u_{\mathrm{eff}}(t) = \frac{I_{\mathrm{ext}} + g_{\mathrm{leak}} E_{\mathrm{leak}} + \sum_{syn} g_{syn}(t) E_{syn}}{g_{\mathrm{tot}}(t)} \quad \text{, and} \tag{3.23}$$

$$\tau_{\mathrm{eff}} = \frac{C_{\mathrm{mem}}}{g_{\mathrm{tot}}(t)} \quad , \tag{3.24}$$

where the total conductance $g_{\mathrm{tot}}(t)$ is:

$$g_{\mathrm{tot}}(t) = g_{\mathrm{leak}} + g_{\mathrm{syni}}(t) + g_{\mathrm{synx}}(t) \quad , \tag{3.25}$$

where $g_{\mathrm{syni}}(t)$ is the sum of all conductances from the inhibitory synapses and $g_{\mathrm{synx}}(t)$ is the sum of all conductances from the excitatory synapses. For the time being, we set the external current to zero $I_{\mathrm{ext}} = 0$ to simplify the discussion; I_{ext} has the same effect as changing the leakage potential. In the LIF Sampling theory, we choose such a high Poisson frequency that the neuron is elevated into the high-conductance state [Destexhe et al., 2003] meaning that the mean total conductance is dominated by the contribution of the Poisson noise $\langle g_{\mathrm{tot}} \rangle \gg g_{\mathrm{leak}}$. The expectation value is taken of the stochasticity of the Poisson noise. Further, in the high-conductance state we can assume that the membrane is very fast, specifically the effective membrane time constant is much smaller than the typical time constants $\langle \tau_{\mathrm{eff}} \rangle \ll \tau_{\mathrm{mem}}; \tau_{\mathrm{ref}}$. We can, hence, assume that the realized membrane potential instantaneously follows the effective membrane potential. In this setup the dynamics of the membrane potentials can be described by an Ornstein-Uhlenbeck process [Petrovici et al., 2016]. There the expected membrane potential acts as the equilibrium point and the fluctuations in the Poisson noise as the noise term. However, the description only holds for observation times longer than the synaptic time-constant of the Poisson noise connections. The Wiener process part of the noise is not scale-free, as typically assumed in ideal examples. Due to the finite τ_{syn}, there is non-zero autocorrelation in the noise on short time-scales, unlike in an ideal Ornstein-Uhlenbeck process. The parameters of the steady-state distribution of the membrane potential can be calculated if the spiking mechanism is turned off:

$$\begin{aligned}\langle u \rangle = \mu &= \frac{g_{\text{leak}} E_{\text{leak}} + \sum_{syn} \nu_{syn} w_{syn} E_{syn} \tau_{syn}}{\langle g_{\text{tot}} \rangle} \quad , \\ \sigma^2 &= \frac{\sum_{syn} \nu_{syn} [w_{syn}(E_{syn} - \mu)]^2 \tau_{syn}}{\langle g^{\text{tot}} \rangle^2} \quad .\end{aligned} \tag{3.26}$$

Petrovici et al. [2016] have shown that the high-conductance state symmetrizes the activation function of the neurons. The activation function is defined here as the probability of the on-state as a function of the mean free membrane potential. We call "free membrane potential" the membrane potential that would be realized without the spiking and resetting mechanism if the neuron experiences the same Poisson noise. We know from the properties of the Ornstein-Uhlenbeck process that the steady-state membrane potential follows a Gaussian distribution. If we turn on the spiking mechanism, then as soon as the membrane potential u_{mem} reaches the threshold potential, the neuron generates a spike and the membrane potential is pulled back to the V_{reset} reset potential. After the refractory time has ended, the membrane potential resumes following the effective membrane potential. In the high-conductance state we can think in a zero-order approximation of the membrane potential instantly following the effective membrane potential. The importance of the high-conductance state becomes apparent when the neuron bursts. This means it produces several spikes with the minimal temporal separation of a refractory time τ_{ref}. After reaching the firing threshold for the first spike in the burst, the membrane potential is immediately pulled to V_{reset} as the LIF model requires. At the same time, the free membrane potential evolves on and by definition of the burst it is above the spiking threshold after τ_{ref} time. The now again released membrane potential begins to catch up with the free membrane potential, increases, reaches V_{thresh} and produces the next spike in the burst. In an imagined perfect high-conductance state, $\frac{\tau_{\text{eff}}}{\tau_{\text{ref}}} \to 0$, this rise time can be thought of as negligible. But if the high-conductance state is insufficient, then the rise time influences the activation function by causing a slower than logistic convergence to the highest reachable spiking probability for $\langle u \rangle \to \infty$. First, the finite rise time decreases the firing rate compared to the perfect high-conductance state because the separation of the spikes in the bursts is larger then τ_{ref}. With increasing mean free membrane potential, this separation slowly decreases, leading to a slow convergence towards the pure *on-state*.

Having established the qualitative correspondence between a single neuron of an RBM and a LIF neuron with high-frequency Poisson noise, we proceed by defining the mapping between the parameter space of a Botzmann-machine and its spiking analog. We map the biases by equating $p(z_i = 1)$, the probability of the single neurons to be in the on-state:

$$\sigma(b_i) = \sigma\left(\frac{\langle u_i \rangle - \bar{V}_i^0}{\alpha}\right) \quad , \tag{3.27}$$

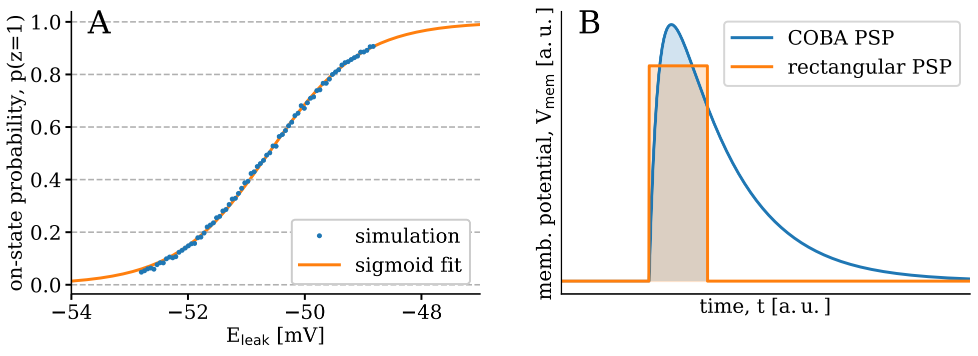


Figure 3.7: Equating LIF sampling and neural sampling (A) Activation function of a single LIF neuron with high-frequent Poisson noise. The occurrence of the on-state is plotted against the leakage potential E_{leak} which controls the bias. A logistic function is fitted to the simulation results. The fitted activation function is used to calibrate the bias of the sampling unit. **(B)** To transform the weights from the abstract domain (orange) to the LIF domain (blue), the PSPs are equated during the refractory time. Mainly the long tail of the PSP for COBA-LIF neurons causes the difference between the stochastic spiking network and its inspiration the Boltzmann machine [Baumbach, 2016, 2020 in preparation]. The measurements were made with the software SBS (appendix B.3) using the Nest simulator [Gewaltig and Diesmann, 2007].

where b_i is the bias of the abstract neuron as in equation (3.2), α is the fitted slope and $\bar{V}_i^0$ is the fitted midpoint of the LIF activation function $p(z_i = 1; \langle u \rangle = \bar{V}_i^0)$.

Finally, we obtain the mapping of the weights by equating the effect the neurons have on each other as a result of their spikes during the refractory time. In the case of LIF neurons this effect is the PSP that is caused by the spiking activity of a pre-synaptic neuron. In general, we lack a closed expression for the PSP of an LIF neuron with conductance based synapses. Still, in the limit when the reversal potentials are far away from the dynamic range of the membrane dynamics we can approximate it. For the abstract neurons, we can associate a rectangular PSP of height W_{ij} to the weight between the abstract neurons. Hence, by requiring

$$\frac{1}{\alpha}\int_0^{\tau_{\text{ref}}} V_{\text{PSP}}(t)\mathrm{d}t \overset{!}{=} W_{ij}\tau_{\text{ref}} \quad , \tag{3.28}$$

we obtain the translation formula

$$W_{ij}\tau_{\text{ref}} = \frac{1}{\alpha}\int_0^{\tau_{\text{ref}}} \frac{w_{ij}(E_{\text{syn}_j} - \mu)\tau_{\text{syn}}}{\langle g_{\text{tot}} \rangle} \frac{\left(\exp(-\frac{t-t_s}{\tau_{\text{eff}}}) - \exp(-\frac{t-t_s}{\tau_{\text{syn}}})\right)}{\tau_{\text{eff}} - \tau_{\text{syn}}}\mathrm{d}t \quad , \tag{3.29}$$

with E_{syn} being the reversal potential of the respective synapse. The mapping between the abstract and LIF weights gives rise to the requirement of equal synaptic time-scale and refractory time, $\tau_{\text{syn}} = \tau_{\text{ref}}$, because the PSPs are the most similar in this setup.

Note, that there are several approximations between the neural sampling theory and LIF-sampling: 1) the activation functions are only similar in the limit of a perfect high-conductance state 2) the time is inherently continuous in LIF-sampling while discrete in neural sampling 3) the state updates are simultaneous in LIF-sampling while sequential in neural sampling 4) the interactions are strictly rectangular in neural sampling while they take approximately a difference-of-exponentials shape for LIF-sampling. Due to these differences, the network of LIF neurons samples only approximately from the probability distribution of the corresponding Boltzmann machine (figure 3.6 C). To emphasize this difference we refer to a sampling network according to the LIF-sampling theory as Stochastic Sampling Network (SSN) to distinguish it from classic Boltzmann machines and from networks according to the neural sampling theory. Still, the similarity is close enough such that applications and modifications found in Boltzmann machines can be used in SSNs. We can build hierarchical sampling networks (figure 3.6 D) based on RBMs [Leng et al., 2018, Dold et al., 2019], we can train SSNs via contrastive divergence [Leng et al., 2018, Dold et al., 2019], we can facilitate mixing between the modes by changing the corresponding temperature of the network [Korcsak-Gorzo et al., 2020] and we can study the analogy between magnetic materials and networks of neurons along the idea of spin glasses [Baumbach, 2016, 2020 in preparation].

Sampling without explicit noise

A particularly interesting feature of the LIF-Sampling theory is that it does not require explicit noise to realize networks that behave for all practical purposes as stochastic systems. Jordan et al. [2019] have shown that the private Poisson noise of the sampling neurons can be replaced by the spiking activity of an appropriate excitatory-inhibitory network of spiking neurons. Dold et al. [2019] extended this idea and showed that SSNs can serve for each other as sources of stochasticity; hence an ensemble of SSNs can mutually provide noise for each other while each SSN still performs its own distinct task. In the following, we present the main works and their connections, which lead to this conclusion.

The dynamic state of spiking neural networks is a thoroughly studied phenomenon, but most studies mainly rely on computer simulations and analytic calculation in rate-based approximations, e.g. Brunel [2000], Ledoux and Brunel [2011]. Note that most simulations respect Dale's principle and use neurons that are either strictly excitatory or inhibitory (section 2.2.1). In his seminal paper, Brunel [2000] mapped the state-space of random spiking neural networks as a function of the strength of the excitatory and inhibitory weights. Brunel found that inhibition dominated random spiking networks exhibit an Asynchronous Irregular firing regime, in which the activity of the neurons shows apparent chaotic behavior. Interestingly, the spiking mechanism aids the emergence of this microscopic chaos, which lacks a similar realization in rate-based simulations. Building on these observations, several studies claim that spiking networks show chaotic behavior and show the best computational capabilities at the edge of chaos [Levina et al., 2007, Chialvo, 2010, Cramer et al., 2019a].

For LIF sampling it is suggestive that the spiking activity of these irregularly spiking networks could replace the artificial Poisson noise. Jordan et al. [2019] studied the sampling quality of LIF Sampling as a function of the size of the random network. They found that in the limit of large networks the quality of sampling in terms of D_{KL} is on par with the sampling quality of Poisson noise. The dominating inhibition in the random network actively cancels out cross-correlations between the neurons of the random network, such that cross-correlations in the noise activity do not limit the sampling quality. In their setup, the authors respected Dale's principle both in the connections of the random network and in the connections from the random network to the sampling network.

Bytschok et al. [2017] studied the effect of cross-correlations on the noise to the sampling neurons. The authors found that cross-correlations in the background Poisson noise are equivalent to weights and biases in the sampling network. This is more apparent if we transform the abstract description of the probability distribution from the $\{0;1\}^N$ state-space to the $\{-1;1\}^N$ state-space, where cross-correlations map to weights in the abstract description. Because of this correspondence, the distorting effect of cross-correlations can be trained away with the same CD training as we use for learning from data (section 3.2.1). More precisely, training incorporates the noise cross-correlations into the SSN. Dold et al. [2019] extended these concepts and showed that an ensemble of SSN can provide each other with appropriate stochasticity without any additional noise and via training they can all simultaneously solve functional tasks.

These results contribute to understanding the origin of noise and trial-to-trial variability in the brain: the behavior of neurons in in-vitro experiments is surprisingly deterministic compared to their inherent trial-to-trial variability in *in-vivo* experiments. The results above suggest that the apparent stochasticity stems from the (uncontrolled) activity of other brain areas. Here, we use these findings to circumvent the limited external bandwidth between the FPGA and the wafer module. By placing a random network, that is a source of stochasticity, directly onto the neural substrate, we can implement larger SSNs on BSS-1.

3.3 Experimental setup

Having established the theory and introduced the main characteristics of the hardware, we can now design an experiment which both respects the constraints of the substrate and the requirements of the theory, while keeping the setup scalable. We point out the central aspects and challenges of the implementation:

1. A main challenge is the required Poisson noise for the sampling neurons. From simulations [Kungl, 2016] we know that LIF Sampling requires at least 300 Hz Poisson input excitatory and inhibitory each. Combined with the bandwidth limitation of 1250 Hz per HICANN chip we can at most provide

enough noise for two sampling neurons per HICANN not counting the data-input. Hence, we opt for the implementation of sampling with an on-chip random network as a noise source and additional CD training to compensate for the residual cross-correlations in the background input [Jordan et al., 2019, Bytschok et al., 2017, Dold et al., 2019]. We use a small fully-connected SSN to compare the implementation using random network to using Poisson spikes generated on the host computer.

2. The excitatory-inhibitory random network used by Jordan et al. [2019] turned out to be impractical for hardware implementation (data not shown). The parameters of the network have to be tuned to find the balance between falling into a quiescent state and a run-away activity. On hardware this is even more challenging because the run-away activity might not be detected due to the external band-width: the generated spikes are lost during read-out at times of high-activity. Hence, we use the purely inhibitory network self-sustained via a leakage potential higher than the threshold potential used in Pfeil et al. [2016], which is easier to control.

3. From a previous work [Kungl, 2016], we know that the writing precision of the analog parameter storage is too low for the direct implementation of LIF Sampling on the BSS-1 system. Additionally, at the time of this project writing the analog parameters took approximately 20 s compared to the smaller than 1 s writing time of the digital parameters. Hence, we use the weight from regularly spiking bias neurons to the sampling neurons to set the desired bias values.

4. At the time of the project, calibration was available only for a subset of the parameters. We use calibration wherever it is possible and use CD training to reduce the effect of fixed-pattern noise.

5. Due to the fixed-pattern noise, the symmetry of the weights is not granted on analog hardware, but LIF Sampling theory requires strictly symmetric connections between the sampling neurons. Two weights of the same value in terms of bits can have different effects on the post-synaptic neuron. Because we lack a scalable mechanism to ensure symmetry, we set the weights of the reciprocal connections equally in terms of set bits. Based on software simulations and measurements on BSS-1 [Fehre, 2017], we expect that the violated symmetry requirement is not a limiting factor for the implemenatation on BSS-1.

6. The lack of weight-calibration affects the mapping of Boltzmann machines from the abstract domain to the hardware as well. Without calibration we cannot use the theoretical weight translation formula (equation (3.29)). Instead, we use an *ad-hoc* translation based on the observed activation functions on the BSS-1 and use CD training to fine-tune the weights and biases.

7. The size of the network was limited by the synapse loss. We implemented the largest where the synapse loss stayed below approximately 2 %.

Similar considerations also apply for any neuromorphic hardware implementation of computational models in terms regarding availability of calibration and controllability of the parameters (e.g. Wunderlich et al. [2019], chapter 4, Göltz et al. [2019]). It would be tempting to fine-tune components of the network to the purpose or include tedious optimizations in the setup. Notably, in Kungl [2016] and in Petrovici et al. [2016] the authors used a cross-correlation-based procedure to symmetrize the weights between the sampling neurons, but the time required for it scales linearly with the number of synapses; which is not practical for desired large-scale implementations. Here, we explicitly refrain from such non-scaling approaches and mainly rely on the inherent robustness of the computational model and the compensating mechanism of the training.

3.3.1 Network setup

All presented experiments were carried out on a single wafer module of the BSS-1 system on the available HICANNs (section 3.1). Due to different reasons, such as malfunctioning analog and digital parts, restrictions in the mapping algorithm and most prominently the ongoing commissioning, only a subset of the wafer was ready to be used. The networks were specified in PyNN [Davison et al., 2009] and we used the standard calibration [Koke, 2017, Kleider, 2017, Schmitt et al., 2017] to set the hardware parameters. Each experiment has two main components: 1) the sampling network with symmetric connections between connected neurons, 2) a source of stochasticity.

In the original formulation [Petrovici et al., 2016], the model sets the bias values of the neurons by changing neuronal potentials (E_leak, E_exc, E_inh). On BSS-1, writing the analog parameters is much slower than changing the digital values stored in SRAM cells. We modify the implementation of biases: We replace the single neurons by a two-neuron module consisting of a sampling and a bias neuron (figure 3.8 A-B). The sampling neuron still represents the binary variable and receives the noise input. We configure the bias neuron to regular firing by setting $E_\text{leak} > V_\text{thresh}$ (figure 3.8 C inset). The bias neuron projects to the sampling neuron, setting the weight of this connection hence controls the bias of the sampling neuron. Because the excitatory and inhibitory synapses are mapped to distinct circuits on BSS-1, we have to set two projections to enable a sign change in the bias. For larger networks, for a more parsimonious resource usage, we can reduce the number of bias neurons by letting the samplers share several bias neurons (connected via distinct synapses). Similarly, we implement the connections between sampling neurons via pairs of synapses to allow for sign change.

We provide stochasticity to the sampling neurons in two distinct ways. In the first setup, we replicate the straightforward implementation [Petrovici et al., 2016] by providing externally generated Poisson noise to the neurons (figure 3.8). This setup is restricted only to small SSNs, hence we will only use it as a comparison. In the second setup, we use the purely inhibitory random network with self-starting neurons ($E_\text{leak} > V_\text{thresh}$) as used in Pfeil et al. [2016]. The sparse random mutual inhibition ensures a relatively constant average firing rate and reduces the

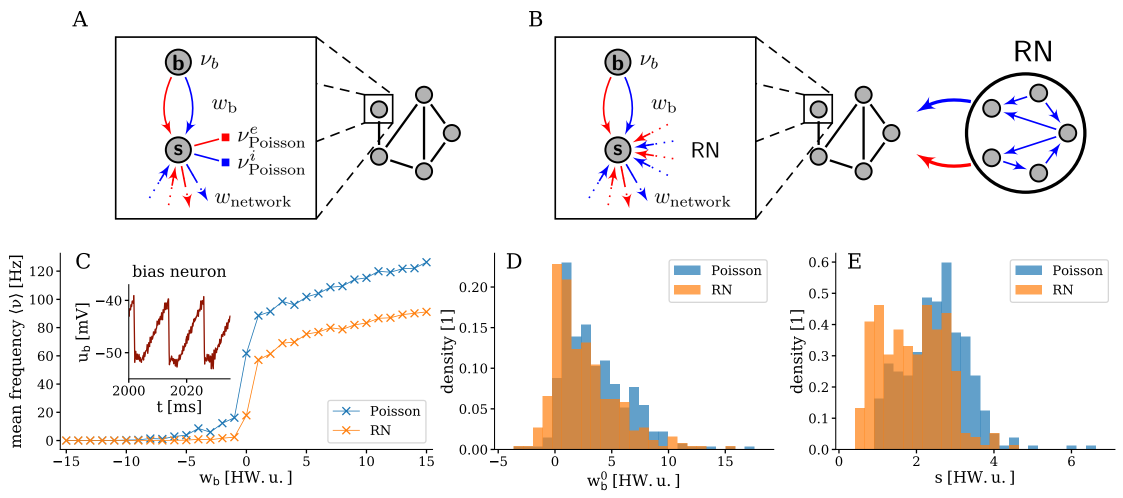


Figure 3.8: Experimental setup. Each sampling unit is instantiated by a pair of neurons on the hardware. The bias neuron ⓑ is configured with a suprathreshold leak potential and generates a regular spike train that is fed to the sampling neuron ⓢ, thereby serving as a bias, controlled by w_b. **(A)** As a benchmark, we provided each sampling neuron with private, off-substrate Poisson spike sources. **(B)** Alternatively, in order to reduce the I/O load, the noise was generated by a random network (RN). The RN consisted of randomly connected inhibitory neurons with $E_\mathrm{leak} > V_\mathrm{thresh}$. Connections were randomly assigned, such that each sampling neuron received a fixed number of excitatory and inhibitory pre-synaptic partners (table 3.2). **(C)** Exemplary activation function (mean firing frequency) of a single sampling neuron with Poisson noise and with an RN as a function of the bias weight. The standard deviation of the trial-to-trial variability is on the order of 0.1 Hz for both activation functions, hence the error bars are too small to be shown. The inset shows the membrane trace of the corresponding bias neuron. **(D-E)** The figures show histograms over all neurons in a sampling network on a calibrated BSS-1 system. The width s and the midpoint w^0_b of the activation functions with Poisson noise and with an RN are calculated by fitting the logistic function $\langle\nu\rangle = \nu_0/\{1+\exp[-(w_\mathrm{b}-w^0_\mathrm{b})/s]\}$ to the data. Figure taken from Kungl et al. [2019].

cross-correlation between the neurons of the random network. Training with CD compensates the residual correlations by incorporating them into the structure of the SSN. Note, how the same plasticity plays three *a priori* distinct roles: 1) it guides the classic learning procedure, that makes the network learn the dataset, 2) it mitigates the effect of fixed-pattern noise, 3) it compensates for the cross-correlations in the background noise. These attributes of learning enabled us to replace the ideal Poisson noise by a random network in the hardware implementation.

With both settings, we measured the activation functions of the sampling neurons and they took an approximately sigmoid shape as required by the theory (figure 3.8 C). Due to the variability of the hardware circuits and the precision of the analog parameter storage, the shape of the activation function varied strongly between neurons (figure 3.8 D-E). Effectively, this introduces an additional random deviation on the initial weight and bias parameters and this renders the effective learning rate different across the neurons. Still, it does not prevent the learning to reach good solutions.

The used parameters are shown in table 3.1 and the network setup is summarized in table 3.2. Our largest network spread over 28 HICANN chips distributed over 7 reticles of a single wafer module, consisting of 609 neurons with 208 sampling neurons, 400 random network neurons and 1 bias neuron. Each neuron was realized using 4 denmem units. Nominally, 7 reticles contain 28 672 denmem units. We required 7 reticles due to the diverse constraints discussed in section 3.1.1 and we spread out the network for a more convenient mapping.

3.3.2 Training in-the-loop

To train the network without on-chip plasticity, we used the idea of in-the-loop training (Schmuker et al. [2014], Esser et al. [2016], Schmitt et al. [2017]; section 2.3.3). As discussed before, the analog parameters are only written once at the beginning of the experiment. During the training procedure, we only change digital parameters, namely the weights between the sampling neurons and the bias connections from the bias neurons to the samplers. This resulted in a larger experiment-repetition frequency and therefore faster training speed in terms of wall-clock time spent compared to the case if we changed the analog parameters.

For the updates, we configure the network, we execute the experiment on BSS-1 and read out the spike-trains of the sampling neurons. We turn the spike-trains into states according to LIF Sampling theory and we calculate the parameter updates with the CD rule. During training, the parameter values are stored in double precision float numbers; and for experiment execution we discretize them deterministically to the nearest available 4-bit value. In machine learning, this is called the method of "shadow weights" [Courbariaux et al., 2015]. We speed up the training using the momentum method [Rumelhart et al., 1986], but otherwise we refrained from more elaborate optimization procedures. In this study, we want to verify the implementation and demonstrate the feasibility of LIF Sampling on BSS-1. In principle, the training could be combined with any optimization

Table 3.1: Neuron parameters. Parameters of the network setup specified in table 3.2. The analog parameters are shown as specified in the software setup and not as realized on the hardware. For details on the calibration procedure see, e.g., [Schmitt et al., 2017]. *Legend:* * the calibration of the membrane time constant was not available at the time of this work, and the corresponding technical parameter was set to the smallest available value instead (fastest possible membrane dynamics for each neuron). Table taken from Kungl et al. [2019].

A		**Sampling neuron**
Name	*Value*	*Description*
V_{reset}	$-35\,\mathrm{mV}$	reset potential
E_{leak}	$-20\,\mathrm{mV}$	resting potential
V_{thresh}	$-20\,\mathrm{mV}$	threshold potential
E_{inh}	$-100\,\mathrm{mV}$	inhibitory reversal potential
E_{exc}	$60\,\mathrm{mV}$	excitatory reversal potential
τ_{ref}	$4\,\mathrm{ms}$	refractory time
τ_{mem}	ca. $7\,\mathrm{ms}$	membrane time constant*
C_{mem}	$0.2\,\mathrm{nF}$	membrane capacity
$\tau_{\mathrm{syn}}^{\mathrm{exc}}$	$8\,\mathrm{ms}$	excitatory synaptic time constant
$\tau_{\mathrm{syn}}^{\mathrm{inh}}$	$8\,\mathrm{ms}$	inhibitory synaptic time constant
B		**Bias neuron**
Name	*Value*	*Description*
V_{reset}	$-30\,\mathrm{mV}$	reset potential
E_{leak}	$60\,\mathrm{mV}$	resting potential
V_{thresh}	$-20\,\mathrm{mV}$	threshold potential
E_{inh}	$-100\,\mathrm{mV}$	inhibitory reversal potential
E_{exc}	$60\,\mathrm{mV}$	excitatory reversal potential
τ_{ref}	$1.5\,\mathrm{ms}$	refractory time
τ_{mem}	ca. $7\,\mathrm{ms}$	membrane time constant*
C_{mem}	$0.2\,\mathrm{nF}$	membrane capacity
$\tau_{\mathrm{syn}}^{\mathrm{exc}}$	$5\,\mathrm{ms}$	excitatory synaptic time constant
$\tau_{\mathrm{syn}}^{\mathrm{inh}}$	$5\,\mathrm{ms}$	inhibitory synaptic time constant
C		**Neurons of the random network**
Name	*Value*	*Description (all analog)*
V_{reset}	$-60\,\mathrm{mV}$	reset potential
E_{leak}	$-10\,\mathrm{mV}$	resting potential
V_{thresh}	$-20\,\mathrm{mV}$	threshold potential
E_{inh}	$-100\,\mathrm{mV}$	inhibitory reversal potential
E_{exc}	$60\,\mathrm{mV}$	excitatory reversal potential
τ_{ref}	$4\,\mathrm{ms}$	refractory time
τ_{mem}	ca. $7\,\mathrm{ms}$	membrane time constant*
C_{mem}	$0.2\,\mathrm{nF}$	membrane capacity
$\tau_{\mathrm{syn}}^{\mathrm{exc}}$	$8\,\mathrm{ms}$	excitatory synaptic time constant
$\tau_{\mathrm{syn}}^{\mathrm{inh}}$	$8\,\mathrm{ms}$	inhibitory synaptic time constant
D		**Synapse**
Name	*Value*	*Description*
w_{bias}	[0,15]	synaptic bias weight in hardware values (digital)
w_{network}	[0,15]	synaptic network weight in hardware values (digital)
d	on the order of $1\,\mathrm{ms}$ (uncalibrated)	synaptic delay, estimated in [Schemmel et al., 2010]

Table 3.2: Network parameters. Parameters are shown for the three different cases described in the manuscript: **(A)** Target Boltzmann distribution, Poisson noise. **(B)** Target Boltzmann distribution, random network for stochasticity. **(C)** Learning from data, random network for stochasticity. Note that the *in-degree*, sometimes also referred to as a *fan-in factor*, represents a neuron's number of pre-synaptic partners coming from some specific population. Table taken from Kungl et al. [2019].

A		**Probability distribution with Poisson Noise**
Name	*Value*	*Description*
N_s	5	number of sampling neurons
N_b	1	number of bias neurons
N_r	0	number of random neurons
K_RN	-	within-population in-degree of neurons in the random network
K_noise	-	in-degree of sampling neurons from the random network
w_RN	-	synaptic weights in the random network in hardware units
$\nu_\mathrm{Poisson}^\mathrm{e/i}$	300 Hz	Poisson frequency to sampling neurons per synapse type
B		**Probability distribution with random network**
Name	*Value*	*Description*
N_s	5	number of sampling neurons
N_b	1	number of bias neurons
N_r	200	number of random neurons
K_RN	20	within-population in-degree of neurons in the random network
K_noise	15	in-degree of sampling neurons from the random network
w_RN	10	synaptic weights in the random network in hardware units
$\nu_\mathrm{Poisson}^\mathrm{e/i}$	-	Poisson frequency to sampling neurons per synapse type
C		**High-dimensional dataset**
Name	*Value*	*Description*
N_s	$\{207, 208\}$	number of sampling neurons, { rFMNIST, rMNIST }
N_b	1	number of bias neurons
N_r	400	number of random neurons
K_RN	20	within-population in-degree of neurons in the random network
K_noise	15	in-degree of sampling neurons from the random network
w_RN	10	synaptic weights in the random network in hardware units
$\nu_\mathrm{Poisson}^\mathrm{e/i}$	-	Poisson frequency to sampling neurons per synapse type

procedure since the CD update provides the required parameter gradient. In table 3.3 we show the used learning parameters.

Table 3.3: Parameters for learning. We did not carry out any systematic hyper-parameter optimization. Note that the used learning parameters in the experiments in section 3.4 are not directly comparable because the different statistics of the background noise (Poisson or random network) correspond to different effective learning rates. Table taken from Kungl et al. [2019].

Experiment	*Learning rate*	*Momentum factor*	*minibatch-size*	*Initial* $(\mathbf{W}, \mathbf{b})$
target distr., Poisson	1.0	0.6	-	$\mathcal{U}(-15, 15)$
target distr., RN	0.5	0.6	-	$\mathcal{U}(-15, 15)$
rMNIST	0.4	0.6	7/class	pre-trained
rFMNIST	0.4	0.6	7/class	pre-trained

After, and for hierarchical structures, also during training; clamping of the neurons (i.e. forcing them into state 1 or 0) was required. We realized the clamping by injecting regular spike trains with 100 Hz via multapses with a multiplicity of 5, excitatory for $z_k = 1$ and inhibitory for $z_k = 0$. A multapse means several simultaneous synaptic connections from the pre-synaptic neuron to the post-synaptic neuron. This was needed to overcome the limited input from a single synaptic connection and to achieve proper clamping even in the presence of input to the neurons from other parts of the network.

3.4 Experiments and results

3.4.1 Training to represent probability distributions

First we verify in experiments that an SSN trained with the combination of the proposed algorithm and hardware can indeed sample from the distribution of a Boltzmann machine. To this end we train an SSN driven by an RN to sample from a specified fully visible Boltzmann machine. As a benchmark, we compare sampling accuracy to the target distribution of the Boltzmann machine and to a trained SSN driven by perfectly uncorrelated Poisson spike trains generated on the host computer. We keep the target Boltzmann machine small on purpose both to make the Poisson noise comparison feasible despite bandwidth limitations and to keep the evaluation simple.

We generated 5 neuron Boltzmann machines by choosing the weights and biases from a zero-centered Beta distribution: $b_i, w_{ji} \sim 2[\text{Beta}(0.5, 0.5) - 0.5]$. The Beta distribution is motivated by previous studies [Petrovici et al., 2016, Jordan et al., 2019] to give higher probability to larger weights and biases in order to generate more interesting (rough) energy landscapes even in small networks. The initial weights and biases were generated from a uniform random distribution. Due to the small number of neurons in the target Boltzmann distribution, we calculated the "wake" or "data" part of the learning rule (equation (3.15)) analytically as $\langle z_i z_j \rangle = p^*(z_i = 1, z_j = 1)$ and $\langle z_i \rangle = p^*(z_i = 1)$ by taking the expectation values explicitly based on the target distribution.

We trained the networks for 500 iterations with 1×10^5 ms sampling time per iteration. We considered the parameter configuration with the lowest $D_{\mathrm{KL}}(p \| p^*)$ as the result of the training and tested it in a longer (5×10^5 ms) experiment. The limited and discretized resolution of the weights and biases made this unusual definition of the result necessary: in the late phase of learning, the resulting $D_{\mathrm{KL}}(p \| p^*)$ produces wiggles, while the network tries to find the best parameters from the available resolution. Finally, we studied the inference properties of the network by clamping two of the five neurons to fixed states $(z_1, z_2) = (0, 1)$ and compared the obtained sampled conditional distribution to the expected target. Results of these experiments are shown in figure 3.9.

For both experiments, the learning was fast in the first approximately 20 to 50 iterations and for the Poisson case it reached a plateau after 200 iterations, where the limited resolution of the weights became the limiting factor. Convergence was slower for the RN case — as expected — because the network had to also cope with the cross- and auto-correlations in the background noise, but still achieved similar performance (figure 3.9 A).

In both cases, the observed D_{KL} converged with the same power law to the target distribution, which suggests similar mixing properties and hence sampling speed (figure 3.9 B). After long sampling ($\gg 1 \times 10^3$ ms), the approximate nature of the sampling became apparent. The observed $D_{\mathrm{KL}}(p \| p^*)$ reaches the same plateau for both noise sources at $D_{\mathrm{KL}}(p \| p^*) \approx 2 \times 10^{-2}$. This is half to one order of magnitude higher than in software simulations [Petrovici et al., 2016] using identical neurons (no fixed-pattern noise), perfect Poisson spike trains as noise source and the direct translation of weights and biases (equations (3.27) and (3.29)). The results on the neuromorphic hardware were consistent over independent experiments using 20 different target distributions (figure 3.9 E). Similar observations hold for the inference experiments. The speed of convergence happened slightly faster because taking the conditional reduced the explored state space (figure 3.9 G). The sampled joint and conditional distributions still capture the main features of the target distribution qualitatively well (figure 3.9 H and I, respectively). The lower sampling quality compared to the full distribution stems from the asymmetry of the reciprocal connections. Because they are not considered in the learning algorithm, the training cannot compensate for them.

The training benefited from the speed-up of the BSS-1 system. The entire training took 5×10^2 s wall-clock time, which includes the initialization of the experiment, the pure emulation time on the neuromorphic substrate and the calculation of the parameter updates on the host computer. Compared to the (emulated) biological time of 5×10^4 s, this means a 100-times speedup two orders of magnitude less than the nominal speed-up of 10^4 of the hardware. The reduction is due to the

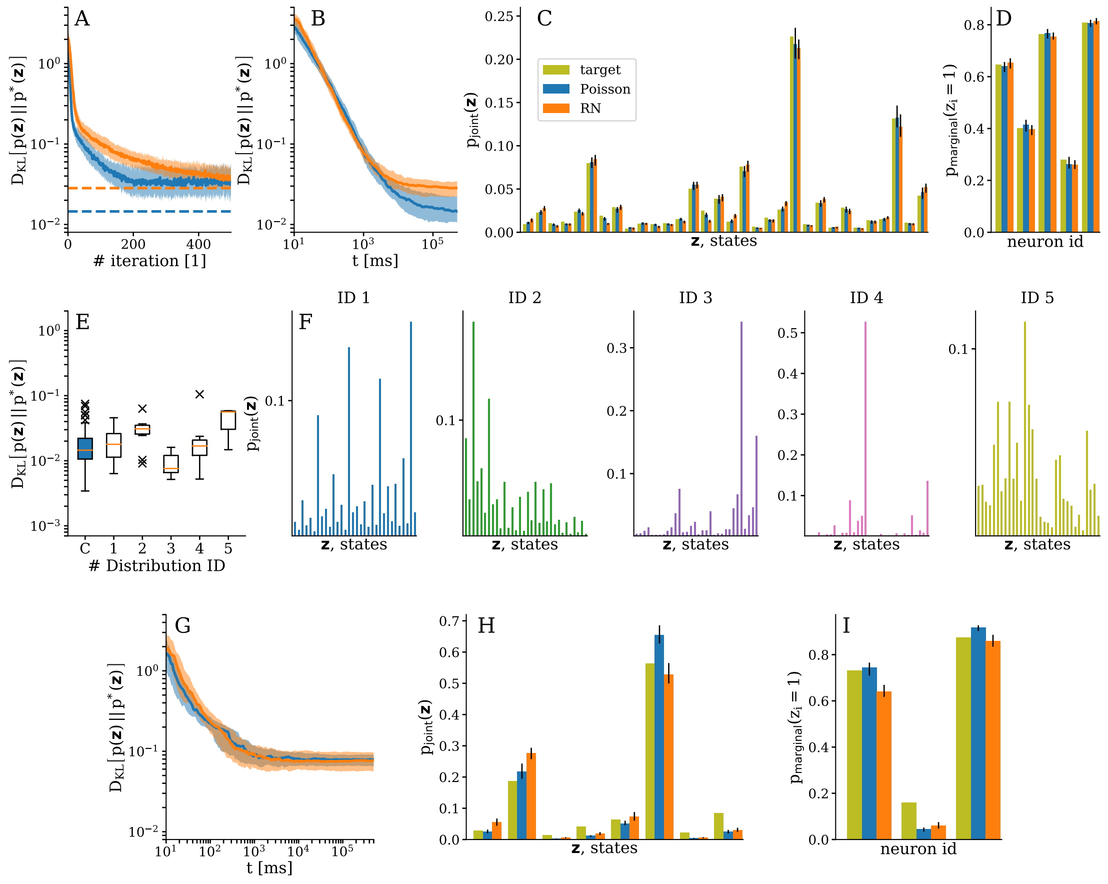


Figure 3.9: Emulated SSNs sampling from target Boltzmann distributions. We show the sampled distributions with Poisson noise (blue), using RNs (orange) and the target distribution (green). Training and sampling were repeated in 150 runs with random initialization. On each plot, we show the median value and the interquartile range. **(A)** The sampling quality improves with training. The precision is mainly limited by the discretization and resolution of the weights. We choose the parameters with the lowest D_{KL} value as the result of the training, and show them as dashed lines. **(B)** Convergence of the sampling distribution during a run. **(C-D)** We show the sampled joint and marginal distributions after training. **(E)** The results are consistent over several target distributions. Here, we show 6 representative distributions with 10 independent runs each. The data is plotted following the traditional box-and-whiskers scheme: the orange line represents the median, the box represents the interquartile range, the whiskers represent the full data range and the $\times$ represent the far outliers. **(F)** corresponding target distributions. Sampling from the conditional distribution after training: **(G)** convergence during a run, **(H)** conditional joint and **(I)** marginal distribution. Figure taken from Kungl et al. [2019].

overhead of network (re)configuration, I/O processes between host and chip, and the update calculation on the host computer.

We tested the robustness of the implementation by repeating the training and testing procedure on 20 different samples of the target distribution with 10 repetitions each both with Poisson noise and the RN as a source of stochasticity (figure 3.10). The reached sampling accuracy stayed approximately constant over the different realizations of the target Boltzmann distribution. The imprecise writing of the floating gates and the inherent stochasticity of the learning causes the trial-to-trail variability in the $D_{\mathrm{KL}}(p \| p^*)$ over several repetitions for the same probability distribution.

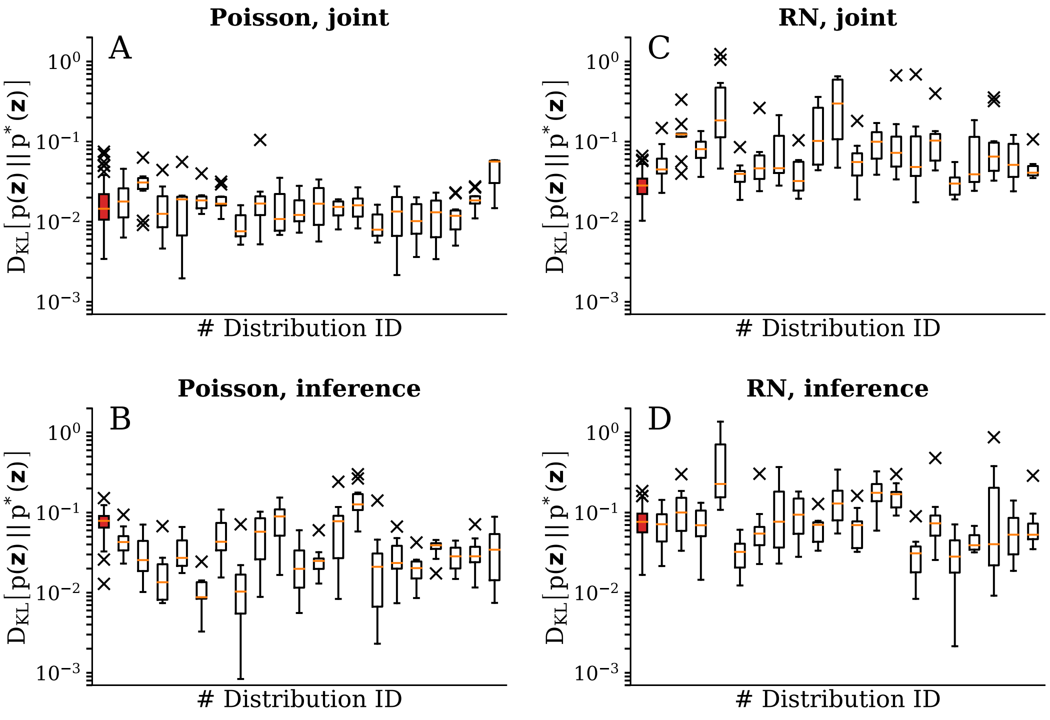


Figure 3.10: Emulated SSNs sampling from different target Boltzmann distributions. The figure shows the results of experiments for 20 different target distributions with 10 repetitions for each sample. The experiments followed the same setup as in section 3.4.1. We show the $D_{\mathrm{KL}}(p \| p^*)$ of the test-run after training for **(A)** the joint distributions with Poisson noise, **(B)** the inference experiment with Poisson noise, **(C)** the joint distributions with a random background network and **(C)** the inference experiment with a random background network. The data is plotted following the traditional box-and-whiskers scheme: the orange line represents the median, the box represents the interquartile range, the whiskers represent the full data range and the $\times$ represent the far outliers. In each subplot the leftmost data (highlighted in red) corresponds to the distribution shown in figure 3.9. Figure taken from Kungl et al. [2019].

3.4.2 Representing high-dimensional datasets

To be able to learn models of data, we trained hierarchical SSNs inspired by restricted Boltzmann machines. We used the reduced version of two standard datasets to test and demonstrate the capabilities of our implementation: the MNIST [LeCun et al., 1998] and the fashion MNIST [Xiao et al., 2017] datasets. The datasets were first preprocessed: We reduced the size of the images using nearest-neighbor resampling (`misc.imresize` function in the SciPy library [Jones et al., 2001–]) and we binarized the originally grayscale images. Resizing the images was necessary due to the limited maximum size of the network on the hardware. From both datasets, we used all images (approximately 6000 images per class, but note that MNIST is not balanced) using 4 classes (0,1,4,7) for the reduced MNIST (rMNIST) and 3 classes (**T**-shirts, **Tr**ousers, **S**neakers) for the reduced fashion MNIST (rFMNIST, see figure 3.11 A-B). The hierarchical network consisted of 3 layers: a visible (144 neurons), a hidden (60 hidden) and a label layer (3 neurons for rFMNIST and 4 neurons for rMNIST).

First, a restricted Boltzmann machine was trained on the respective dataset using the CAST [Salakhutdinov, 2010] algorithm; the resulting network was mapped to the hardware using an empirical factor based on the average activation functions (figure 3.8 C) to convert the weights and biases into hardware parameters. In our experience, the exact numerical value of this empirical factor did not have a perceivable effect on the quality of the results. This initial training procedure provided a baseline for performance comparison, a staring point for the in-the-loop algorithm and better generative properties due to the CAST algorithm. The translation to hardware domain resulted in a significant drop in classification performance, especially with the rMNIST dataset (figure 3.11 C-D). After mapping, we trained the network using the wake-sleep algorithm with the in-the-loop training procedure (section 3.2.1). An analytic calculation of the "data" phase was not possible anymore, hence — like in CD — the visible and label layers were clamped to the correct pixels via external spike-trains. To ensure proper clamping, we cut the synaptic connections from the hidden to the visible and label layers during (and only during) the "data" phase of the learning.

We tested the network in discriminative and generative tasks during and after learning. In the classification task we presented an image to the visible layer by clamping the neurons to the $z_k = 0$ and $z_k = 1$, respectively. Each image was presented for 500 ms in biological equivalent time, which corresponds to 50 µs wall-clock time. Clamping with external spike trains made it possible to send in data in batches, reducing the need of time-costly hardware initializations. The label neuron with the highest activity was considered as the winner. The in-the-loop training restored the classification performance for both datasets (figure 3.11 C-D): The implementation on BSS-1 reached an error of $4.45^{+0.12}_{-0.36}\%$ on rMNIST and $3.32^{+0.27}_{-0.04}\%$ on rFMNIST, which is close to the error of $3.89^{+0.10}_{-0.02}\%$ on rMNIST and $2.645^{+0.002}_{-0.010}\%$ on rFMNIST with an RBM sampled with Gibbs sampling. The results are shown as the median value with interquartile range.

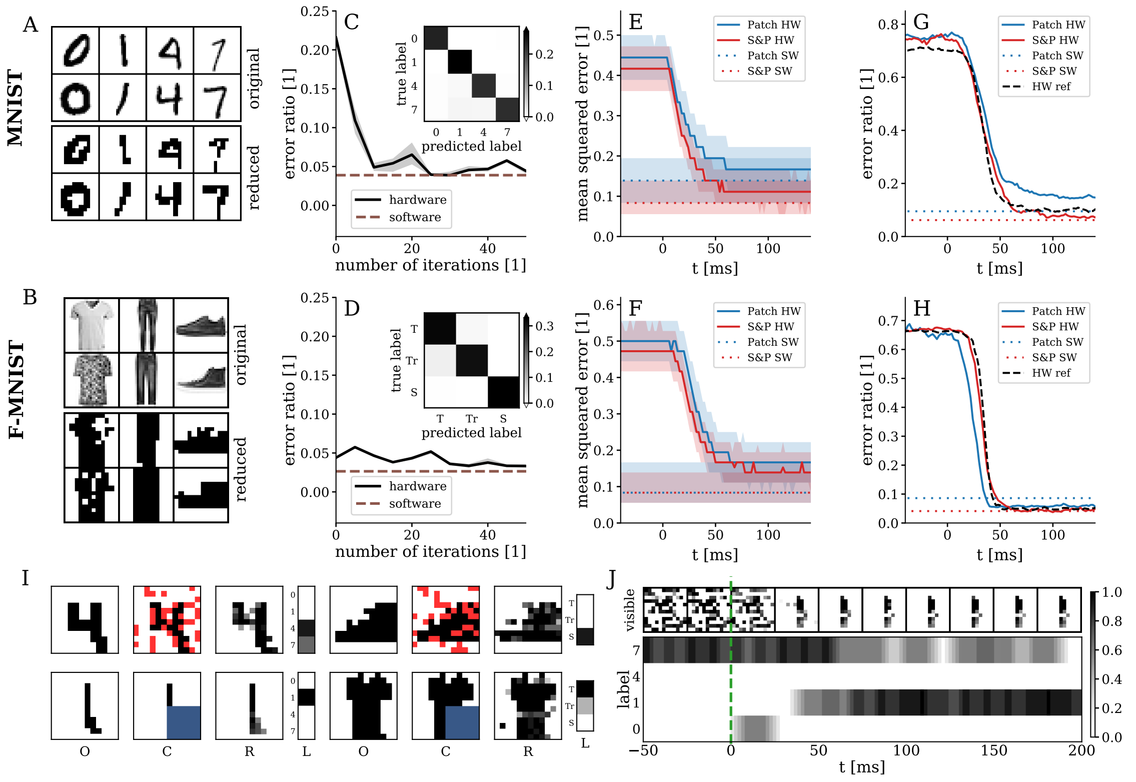


Figure 3.11: Behavior of hierarchical SSNs trained on data. Top row: rMNIST; middle row: rFMNIST; bottom row: exemplary setups for the partial occlusion scenarios. **(A-B)** Exemplary images from the rMNIST (A) and rFMNIST (B) datasets used for training and comparison to their MNIST and FMNIST originals. **(C-D)** Training with the hardware in the loop after translation of pre-trained parameters. Confusion matrices after training shown as insets. Performance of the reference RBMs shown as dashed brown lines. Results are given as median and interquartile values over 10 test runs. **(E-F)** Pattern completion and **(G-H)** error ratio of the inferred label for partially occluded images (blue: patch; red: salt&pepper). Solid lines represent median values and shaded areas show interquartile ranges over 250 test images per class. Performance of the reference RBMs shown as dashed lines. As a reference, we also show the error ratio of the SNNs on unoccluded images in (G) and (H). **(I)** Snapshots of the pattern completion experiments: O - original image, C - clamped image (red and blue pixels are occluded), R - response of the visible layer, L - response of the label layer. **(J)** Exemplary temporal evolution of a pattern completion experiment with patch occlusion. For better visualization of the activity in the visible layer in (J) and (I), we smoothed out its discretized response to obtain grayscale pixel values, by convolving its state vector with a box filter of 10 ms width. Figure taken from Kungl et al. [2019].

The classification benefited from the speed-up of the hardware. Classifying 4125 images from the rMNIST dataset took 10 s wall-clock time or 2.4 ms per image with a speed-up of 210 compared to biological equivalent time. While the classification of 3000 fMNIST images took 9.4 s wall clock time, that is 3.1 ms per image with a speed-up of 160. The measured time is the full turnover, including translation from the PyNN-based network description to hardware parameters, I/O processes, initialization of the hardware and gathering the experiment results. In our experiments the turnover time is dominated by the overhead, the pure physical emulation time of 1.5 ms for the rFMNIST and 2.0 ms for the rMNIST only minimally contribute to the total turnover-time.

We tested the generative properties of the trained network in two distinct task. In the first, the pattern completion task we presented the network 250 images per class with 25 % occlusion (not presented pixels). The occlusion was once done in a salt&pepper (upper row in figure 3.11 I), where randomly chosen pixels are occluded; and in a patch-wise (lower row in figure 3.11 I) manner, where a coherent domain is occluded. Each image was presented for 500 ms, during which the neurons of the visible layer not receiving external clamping evolved freely, according to the internal dynamics of the SSN. To remove any bias effects stemming from the effect of the consecutive pictures on each other, we inserted random clamping to the visible layer between two images. The reconstruction accuracy was evaluated based on the mean squared error (MSE) between the original picture and the network response:

$$\mathrm{MSE} = \frac{1}{N_{\mathrm{pixels}}} \sum_{k=1}^{N_{\mathrm{pixels}}} \left(z_k^{\mathrm{data}} - z_k^{\mathrm{recon}} \right)^2 , \tag{3.30}$$

where z_k^{data} is the reference data value, z_k^{recon} is the model reconstruction and the sum goes over the N_{pixels} pixels to be reconstructed by the SSN. In our case with binarized images, the MSE directly corresponds to the ratio of falsely reconstructed pixels. Simultaneously, with the reconstruction properties, we also followed the classification of the occluded images. Both the reconstruction and the classification converged to their final value after approximately 50 ms corresponding to 5 spikes per neuron when counting with the average refractory time of 10 ms (figure 3.11 E-F). The temporal evolutions of both quantities move together in all of the experiments. The performance of the SSN follows closely that of a restricted Boltzmann machine simulated on a CPU; and the classification performance only slightly deteriorated compared to classification with non-occluded images. Example images of the pattern completion tasks are shown in figure 3.11 I-J for both datasets and for both occlusion versions.

Finally, we tested the network in a guided dreaming scenario, where the network ran to sample images from the learned distribution ("dreaming"). The visible and the hidden layers ran freely, while the label layer was periodically clamped into different one-hot coding conformations in order to nudge the network to sample from the different classes. Ideally, this nudging should be enough to guide the network through the diverse classes. In practice, we introduced a random clamping input of 100 ms when changing the clamped label in order to facilitate the mixing between the learned classes. The SSN could produce recognizable pictures from all the classes as far as the low resolution of pictures allowed it (figure 3.12). For rMNIST, all the classes appeared among the generated images approximately equally. But for rFMNIST the SSN failed to mix into the class "Sneakers", suggesting that there is a great energy barrier between this class and the other two because the "Sneakers" class is too different form the other two. The energy barrier then reduced mixing even with the random input between the clamping periods in the label layer.

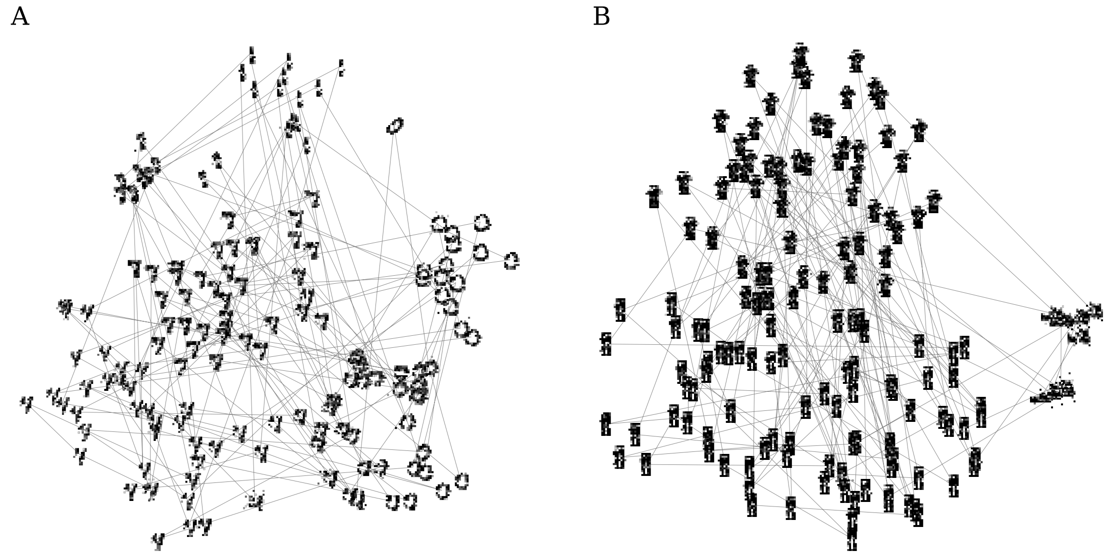


Figure 3.12: Generated images during guided dreaming. The visible state space, along with the position of the generated images within it, was projected to two dimensions using t-SNE [Maaten and Hinton, 2008]. The thin lines connect consecutive samples. **(A)** rMNIST; **(B)** rFMNIST. Figure taken from Kungl et al. [2019].

3.5 Discussion and conclusion

In this project, we demonstrated the first scalable implementation of probabilistic inference in an accelerated spiking neuromorphic hardware. The implementation could both cope with the challenges inherent to (analog) hardware and benefit form the accelerated nature of the substrate. We verified our implementation by successfully training fully connected SSNs driven by Poisson noise and RN to a target distribution (section 3.4.1) and we showed that the implementation can be used in discriminative and generative tasks (section 3.4.2). These experiments were possible in spite of the hardware imperfections and inherent constraints. The training was faster than it would be possible with hardware systems operating in biological real-time. By co-embedding the RN on the same substrate as the sampling network, we circumvented the band-width limitations and achieved a fully autonomous probabilistic machine. It allowed a runtime scaling of $\mathcal{O}(1)$ with the size of the network making use of this inherent scaling property of physical emulations.

3.5.1 Limitations of the study

We consider the limited size of the implemented network as the main limitation of our work. The reasons for this limitation stem from the ongoing commissioning of the BSS-1 system: 1) the software displays limited flexibility, 2) the system assembly is not mature and results in a reduced substrate yield, and 3) the usable wafer area patchy and non-contiguous. The combination of these factors reduces the size of the implementable networks. In order to restrict the resulting synapse loss below approximately 2 %, we maximized the size of the SSN and of the RN. With the ongoing commissioning of the system, improvements in the software and in the assembly procedure, we expect that emulation of larger networks and hence tackling more complex applications will be available in the future. Our setup of implementing spike-based inference scales naturally to larger network sizes.

The sampling accuracy was also affected by the precision of the parameters. At the start of any trial, the limited precision of the analog parameters leads to heterogeneous neuron parameters, which is known to lead to reduction in the sampling quality [Probst et al., 2015]. Most of this is compensated by the wake-sleep training, but the 4-bit resolution of the synaptic weights (and in this implementation of the biases) ultimately limits the capabilities of the network to adapt to a target distribution, reducing the potential quality of sampling. We can observe the effect of the limited resolution in the jumping behavior of the $D_{\mathrm{KL}}(p\|p^*)$ in the late stages of the learning (figure 3.9 A). In small networks, the limited weight resolution impairs the performance of the network [Petrovici et al., 2017b], but this performance penalty decreases for larger networks with large hidden layers, both in spiking and non-spiking models [Courbariaux et al., 2015, Petrovici et al., 2017a]. The next generation of the BrainScaleS system will be equipped with 6-bit resolution weights [BrainScaleS-2, Aamir et al., 2016, Friedmann et al., 2017].

In our setup we used a single bias neuron shared by all the sampling neurons with individual synaptic connections to generate the bias value. One would expect that this introduces cross-correlations between the sampling neurons. But in our experience this effect is neither apparent nor limiting due to several reasons. First, the high firing rate of the biases neurons and the long (in comparison to the inter-spike intervals) synaptic time constant of the sampling neurons leads to an approximately constant bias current. Second, spike time jitter and the heterogeneous synaptic delays on the neuromorphic hardware reduce the correlations. Third, the limitation introduced by the shared bias neuron is overshadowed by other more important limitations such as the size of the network and limited parameter precision. Finally, the in-the-loop training procedure explicitly compensates for the remaining cross-correlations in the background noise [Bytschok et al., 2017, Dold et al., 2019].

We observed limited mixing abilities in our implementation: In the guided dreaming task one of the classes was much harder to generate probably due to its dissimilarity to other classes learned in the same dataset (figure 3.12 B). Slow mixing in restricted Boltzmann machines — and also more general in generative models — in probability landscapes with high peaks as a result of learning from data is a known and studied problem in machine learning. Most models use some form of costly annealing techniques to accelerate mixing [Salakhutdinov, 2010, Desjardins et al., 2010, Bengio et al., 2013]. The fully commissioned BSS-1 system will also feature short-term plasticity [Schemmel et al., 2010] which can be used to facilitate mixing in spiking stochastic networks [Leng et al., 2018].

Currently, most of the full turnover time of the experiments is spent on I/O processes and setting the hardware parameters. This reduces the classification speed (section 3.4.2) from the 50 µs per image pure runtime to 2.4 ms to 3.9 ms gross classification time. We expect that by improvements in the software layer this discrepancy can be reduced.

The applied learning rule was local, but we still required the host computer to turn the measured spike trains into states and to calculate the parameter updates. This procedure slowed down the learning process due to the repeated stopping, evaluation and re-starting of the experiment. Similarly, as in case of classification the spent wall-clock time was dominated by the overhead and not by the pure hardware emulation time that is accelerated by a factor of 10^4. By using the on-chip plasticity in the BrainScaleS-2 successor system [Friedmann et al., 2017, Wunderlich et al., 2019], the iterative procedure could become obsolete resulting in significant speed-up of the learning. The event-based Contrastive Divergence theory [Neftci et al., 2014] is a promising candidate model for the synaptic learning rule, but its compatibility with the BrainScaleS-2 system is yet to be studied.

3.5.2 Relation to other works

Relation to previous works leading to this study

This study builds on a series of experimental and theoretical works studying probabilistic sampling models using neurons. Buesing et al. [2011] showed analytically exact sampling from Boltzmann machines while including the refractory mechanism. The framework was then extended and its dynamics were formally described in Petrovici et al. [2013] and Petrovici et al. [2016] to networks of LIF neurons with actual time-continuous dynamics. Inspired by studies on the asynchronous firing state of random networks [Brunel, 2000], Tetzlaff et al. [2012] and Pfeil et al. [2016] studied the decorrelation effect of inhibitory connections in random spiking networks in theory and in experiments, respectively. The random network was first used to generate stochasticity on-chip by Pfeil et al. [2016] on hardware. Jordan et al. [2019] have shown that an inhibition-dominated random (but dynamically deterministic) spiking neural network can provide sufficient stochasticity to SSNs. Finally, Bytschok et al. [2017] and Dold et al. [2019] discussed the compensation of cross-correlations in the background noise via learning.

Sampling with neurons on hardware

Previous small-scale and partial implementations of sampling on spiking analog hardware can be found, e.g. in Petrovici et al. [2015, 2017b,a]. Pedroni et al. [2016] showed an implementation of neural sampling (closer to Buesing et al. [2011]) on the TrueNorth fully digital neuromorphic system [Merolla et al., 2014]. We point out three main differences between their implementation [Pedroni et al., 2016] and ours. First, the nature of the neuromorphic substrate is different. The TrueNorth is a fully digital system that calculates the dynamics of the single neurons instead of building actual dynamic circuits to emulate it; meaning especially that the neuron parameters and the dynamics are precise on the TrueNorth and variability (fixed-pattern and trial-to-trial) is not an issue. However, the TrueNorth system runs at real-time by design [Merolla et al., 2014, Akopyan et al., 2015], which is 10^4 times slower than the BSS-1. Second, the neurons used in Pedroni et al. [2016] are intrinsically stochastic because the TrueNorth hardware also features stochastic neuron models. Hence, their implementation stays much closer to neural sampling from Buesing et al. [2011]. In contrast, our study considers additional aspects of both the biological inspiration and of analog systems (exponential kernels, leaky membrane, stochasticity through background noise, deterministic firing mechanism, shared correlations etc.). Finally, our approach features a more efficient usage of the hardware real-estate: While in Pedroni et al. [2016] several neuron units are required to build a single sampling unit, in our case a single neuron was sufficient to represent a single sampling unit.

3.5.3 Conclusion

In this project we demonstrated how Bayesian inference can be implemented with spiking neurons on an analog neuromorphic substrate. The implementation was able to cope with the challenges inherent to analog hardware such as the fixed-pattern noise, imprecise parameters, limited control of the parameters and limited bandwidth between the chip and the host computer; while it still benefited from the 10^4 speed-up of the hardware. The entire setup was realized on the hardware including the stochasticity-providing random network, hence we realized a self-contained sampling machine. External communication was only used for representing input data and acquiring response spikes from parts of the network, such as the label neurons. Note, that such a system is also a plausible model for sampling-based inference in the cortex [Jordan et al., 2019, Dold et al., 2019] when considering the deterministic behavior of neurons *in vivo* [Mainen and Sejnowski, 1995, Reinagel and Reid, 2002, Toups et al., 2012, Masquelier, 2013].

We demonstrated sampling from arbitrary Boltzmann distributions (section 3.4.1); and we showed that the framework can be applied to form hierarchical structures to learn from standard datasets and to solve discriminative and generative tasks (section 3.4.2). As previously shown [Probst et al., 2015], the framework can be extended to sample from arbitrary distributions over binary variables. We stress that in the case of hierarchical networks, the model solves the classification and the pattern completion task simultaneously by sampling from the conditional distribution with its dynamics. Both, when learning a target distribution and in inference tasks, the model could make use of the acceleration of the hardware reaching a net speedup of 100 to 210 compared to biological real-time.

We view our project as a contribution to the rapidly expanding (although not yet competitive) field of biologically inspired physical model systems. We demonstrated the feasibility of the approach to tackle machine learning problems, and (implicitly) to study biological phenomena. Our results implicitly show the robustness of the application and its ability to harness the advantages of the underlying substrate, in this case the speed-up. The introduced architecture scales by design to neuromorphic hardware with more available neuronal and synaptic resources. The underlying Boltzmann Machine can be mapped to other problems where a Bayesian approach is beneficial, such as prediction of temporal sequences [Sutskever and Hinton, 2007], solving constraint satisfaction problems [Jonke et al., 2016, Fonseca Guerra and Furber, 2017], movement planning [Taylor and Hinton, 2009, Alemi et al., 2015], quantum many-body problems [Carleo and Troyer, 2017, Czischek et al., 2018, Carleo et al., 2019b, Melko et al., 2019] and simulation of solid state systems [Edwards and Anderson, 1975]. Since the publication of this project, sampling with LIF neurons has been successfully implemented on a prototype chip of the BSS-2 system [Billaudelle et al., 2019b, Stradmann, 2019].

4 Pilot study on the BrainScaleS-2 prototype chip — demonstrating advantages of neuromorphic computation

The content of this chapter was published in Wunderlich et al. [2019] in close collaboration mainly, but not exclusively, with Timo Wunderlich. Here, we follow the publication but we give a more detailed description of the project for the sake of clarity and completeness.

Neuromorphic devices represent an attempt to create novel non-Turing computers relying strongly on inspiration from neural networks in biology [Mead, 1990]. This approach aims at capturing the advantages of the mammalian nervous system to create fast and robust hardware with low-power and/or energy consumption (section 2.3). This endeavor abstracts away details of the brain's neural networks while trying to keep only the central aspects which provide the desired advantages. The holy grail of neuromorphic engineering, the correct level of abstraction and the most important aspects of biological neural networks, remains elusive; but several approaches appeared in search of it (section 2.3.2). A common central approach is that models of neurons, which are considered the basic computational units of the brain, are emulated using specialized digital and/or analog systems (section 2.3.2). Although the idea of neuromorphic engineering reaches back to the 1980s, publications quantifying the performance of neuromorphic hardware on practical tasks are scarce [Merolla et al., 2014, Davies, 2019, Rhodes et al., 2019].

In this project, we demonstrate and quantify the advantages of neuromorphic computation in terms of speed, energy consumption and robustness in a pilot-study. To this end, we use the example of learning a simplified version of the Pong video game with reward-modulated spike-time-dependent plasticity (R-STDP)-based reinforcement learning on the BrainScaleS-2 system High Input-Count Analog Neural Network Digital Learning System v2 (HICANN-DLSv2) neuromorphic prototype chip.

BSS-2 is a neuromorphic hardware system consisting of CMOS-based ASIC [Friedmann et al., 2017, Aamir et al., 2018] implementing physical models of neurons and synapses in mixed-signal circuits. Several features make BSS-2 unique among neuromorphic approaches (section 2.3): 1) the emulation runs with an acceleration of 10^3 compared to biological real-time (compare to BSS-1 in section 3.1) due to the supra-threshold working point of the analog circuits, 2) an on-chip processor enables the flexible implementation of a broad range of plasticity rules and in our study the environment simulation, finally 3) built-in correlation sensors in the synapses and digital spike counters serve as easily accessible observables for plasticity calculations.

In reinforcement learning, a behaving agent interacts with its environment and aims at optimizing its parameters in a way to maximize the obtained reward from the environment (Sutton and Barto [2018] and section 2.1.3). It is a relevant learning paradigm for neuromorphic systems for applications where autonomous systems are required, e.g. in robotics. In recent years, reinforcement learning systems have reached remarkable, often superhuman results in playing board and video games [Mnih et al., 2013, Silver et al., 2017, Vinyals et al., 2019]. Unfortunately, mapping these state-of-the art solutions to neuromorphic hardware is a far-from-obvious task, because they often use mechanisms that are not directly found in neuromorphic systems, e.g. non-local learning rules or tree search. Hence, for more compatible models of reinforcement learning we turn to computational neuroscience. Since the discovery that the neuromodulator dopamine corresponds to the reward-prediction error in the brain [Schultz et al., 1997, Niv, 2009], new mechanistic models, the so-called three factor learning rules, have been developed for neural reinforcement learning [Frémaux et al., 2013, Frémaux and Gerstner, 2015]. In these models the plasticity rule depends on a Hebbian term containing pre- and post-synaptic contributions, and on a third global neuromodulator, which behaves similarly to dopamine in the brain. These learning rules are good candidates for implementation on neuromorphic hardware because they roughly respect the constraints of the substrate. The R-STDP plasticity rule is a simple member of this family of learning rules.

Our experiments take place fully autonomously on the BSS-2 chip using the on-chip plasticity processing unit both for plasticity calculations and for environment simulation. Measurements of time of emulation and energy consumption are compared to a reference implementation using the Nest simulator [Peyser et al., 2017] on an off-the-shelf CPU. The neuromorphic chip achieved an order of magnitude faster emulation and three orders of magnitude lower energy consumption than the simulator on CPU. The learning converged on the neuromorphic chip within seconds. Further, we show that the learning is robust against imprecision of calibration of the neuron parameters and compensates against fixed-pattern noise on the analog parts of the chip. Finally, meta-parameters optimized for one BSS-2 chip can be transferred without significant performance loss to another BSS-2 chip.

4.1 Materials and methods

4.1.1 The BrainScaleS-2 DLSv2 prototype chip

The HICANN Digital Learning System (DLS) v2 neuromorphic chip [Friedmann et al., 2017, Aamir et al., 2018] is a prototype chip of the envisioned BSS-2 large-scale accelerated network emulation platform. Similarly to its predecessor BSS-1 (Schemmel et al. [2010]; section 3.1); it features spiking analog neurons that communicate via all-or-nothing logical spike events transmitted digitally. Unlike BSS-1, it is produced in a 65 nm CMOS process and features the Plasticity Processing Unit (PPU) as on-chip processor. Future versions are envisioned as large systems using wafer-scale technology [Schemmel et al., 2010, Zoschke et al., 2017] to enable the emulation of large-scale networks consisting of several thousands of neurons.

Experimental Setup

BSS-2 contains the neuromorphic chip mounted on a prototyping board (figure 4.1 A). The chip can be accessed and configured either from a host computer through a Xilinx Spartan-6 FPGA or from the embedded processor on the neuromorphic chip. The FPGA in-turn is accessed through Universal Serial Bus (USB) 2.0 from the host computer. The FPGA provides experiment setup, experiment control and hard real-time playback of input and recording of output data. Neuron-to-neuron connections are realized via routing through the FPGA as used in other experiments [Aamir et al., 2018, Stradmann, 2019, Cramer et al., 2019a, Billaudelle et al., 2019b].

The experiments are described in a container-based user interface that provides access to the neuron and synapse parameters and the functional units on the chip. The experiment is then transformed to the DRAM attached to the FPGA. Once the experiment has started, the FPGA starts playing back the experiment data (e.g. input spikes) through a sequencer logic to the neuromorphic chip and at the same time the output from the chip is recorded into the DRAM. After the experiment has been completed, the host computer downloads the recorded data from the FPGA.

The analog neuromorphic core

The BSS-2 continues the "physical modeling" approach as its predecessor (Schemmel et al. [2010]; section 3.1): Instead of calculating the dynamics of the neurons and the synapses, equivalent analog circuits are implemented on the chip mimicking the dynamics of biological neurons. More precisely, BSS-2 [Aamir et al., 2016, 2018] implements LIF neurons with CUBA synapses using exponential kernels (section 2.2.2). Additionally, each neuron is equipped with a 10-bit spike counter, which serves as an observable for plasticity calculations and can be accessed and reset by the embedded processor [Friedmann et al., 2017].

Similarly to BSS-1, BSS-2 emulates the network dynamics at an accelerated speed, but slower with an acceleration factor of 10^3; that is 1 ms emulated biological time

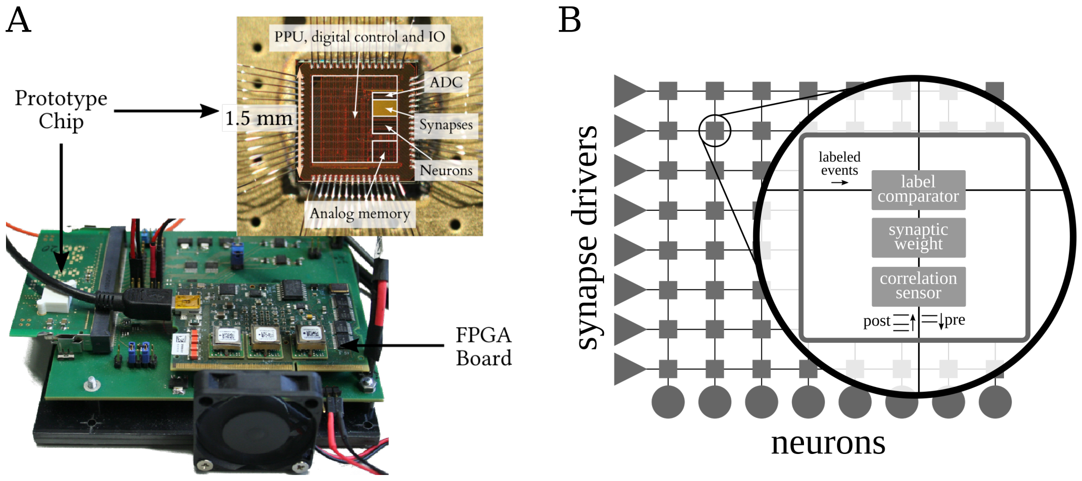


Figure 4.1: The BrainScaleS-2 DLSv2 prototype chip. **(A)** Upper right: Photo of the BSS-2 prototype chip with the different areas labeled. Note that unlike on the HICANN chip (section 3.1), the most chip area is occupied by the Plasticity Processing Unit (PPU) and other digital circuitry. Adapted from Aamir et al. [2018]. **(B)** Schematics of the circuits emulating neurons and synapses. The analog core is highly similar to the one in the predecessor chip (section 3.1). The 32 analog neurons are ordered in an array with the synapse matrix above it ordered into 32 rows. Synapse drivers inject spike events row-wise into the synapse matrix. Each synapse locally stores a 6-bit label to identify the incoming event, a 6-bit weight and an analog coincidence detector to measure the temporal correlation of the pre- and post-synaptic spike events. Figure adapted from Wunderlich et al. [2019].

corresponds to 1 µs on the neuromorphic chip. The slowdown was necessary, such that 1) the digital communication on-chip and to the external components does not pose a bandwidth bottleneck (compare to the bandwidth limitations in section 3.1.1) and 2) the digital plasticity calculations on the PPU can be realistically carried out during runtime of the emulation. The emulation speed is independent of the network size by nature of the emulation. In this project, we refer to all time-constants in the hardware time-scale (wall-clock) in the order of microseconds.[1]

The 1024 synapses are arranged in a 32-by-32 matrix such that each neuron can receive input from 32 synapses from the corresponding column (figure 4.1 B). Each row of synapses is served by a synapse driver that can inject either excitatory or inhibitory (exclusive) spike events into the synapses row. At each synapse the 6-bit label of the spike-event is compared to the label stored in the synapse. If they match then a spike pulse weighted by the 6-bit synaptic weight is generated and sent to the post-synaptic neuron. At the post-synaptic neuron the synaptic circuit generates a PSC injected into the membrane of the neuron. Because the PSP-shaping conversion is completed at the neuron, the largest fixed-pattern noise is present between the neurons and not between the synapse drivers as is for example the case on the similar, but older, Spikey neuromorphic chip [Fehre, 2017].

Post-synaptic spikes generated at the neuron are sent back to the synapses. This way a dedicated coincidence detector circuit can measure the correlation between the pre- and post-synaptic spiketrains by measuring the time elapsed between consecutive spikes. The causal (pre-before-post) and anticausal (post-before-pre) correlations are exponentially weighted and accumulated in two distinct circuits. The PPU can access the accumulators via an Analog-to-Digital Converter (ADC) and it can also reset them. Hence, the correlation measurements serve as an observable for plasticity calculations.

Control and calibration of the analog parameters

The analog neuron parameters are stored in on-chip analog capacitive memory (capmem) cells with 10 bit resolution [Hock et al., 2013]. The capmem cells have less trial-to-trial variability and faster writing speed than the FGs on BSS-1 (section 3.1). All six parameters of the ideal LIF model with CUBA synapses can be calibrated on the BSS-2: membrane time-constant τ_mem, synaptic time-constants $\tau_\mathrm{syn}^\mathrm{exc}$ and $\tau_\mathrm{syn}^\mathrm{inh}$, refractory time τ_ref, resting or leak potential E_leak, threshold potential V_thresh as well as the reset potential V_reset. The inhibitory time-constant $\tau_\mathrm{syn}^\mathrm{inh}$ is not used in this study. In summary, the neuromorphic implementation has 18 tunable analog parameters, but 12 of them are only used to set the circuits to the correct working point [Aamir et al., 2018], hence they only have to be calibrated once for

[1]Note that, while publications using the BSS-1 refer to time-scales in the equivalent biological time, publications using BSS-2 stick to the wall-clock time reflecting a change in the publication policy of the research group. I deemed compatibility to the published literature more important than consistency in this thesis, which is composed of three loosely connected projects.

a given chip. The other 6 parameters directly control one of the model parameters of the current-based leaky integrate-and-fire (CUBA-LIF) equations.

Fixed-pattern noise at manufacturing introduces inhomogeneities among the neuron circuits, leading to differences in the realized neuron parameters. This is most apparent in the case of the time-constants $(\tau_{\mathrm{mem}}; \tau_{\mathrm{syn}}^{\mathrm{exc}}; \tau_{\mathrm{ref}})$. Instead of embracing the fixed-pattern noise as other systems [Neckar et al., 2018], the BSS-2 aims at reducing them via calibration (like the BSS-1, section 3.1). The neurons are calibrated individually to help the mapping from the user-defined LIF parameters to the parameters set on the neuromorphic chip. The parameters can be typically calibrated up to a precision of approximately 5 % [Aamir et al., 2018]. However, the precision of the parameters also depends on the target value of the parameters. Long time-constants close to the available range have lower absolute precision than short time-constants. A detailed description of the calibration procedure is given in Stradmann [2016].

The Plasticity Processing Unit

The main innovation of the BSS-2 is the Plasticity Processing Unit (PPU), which is a 32-bit general-purpose processor implementing the PowerPC-ISA 2.06 instruction set and additional custom vector instructions [Friedmann et al., 2017]. The availability of vector instructions enhances the speed of plasticity calculations with the idea that plasticity calculations include several identical and individually simple calculations. In the DLSv2 prototype chip, the PPU has a clock frequency of 98 MHz and 16 KiB on-chip memory. The 128-bit-wide vector registers can be progressed in eight 16-bit or sixteen 8-bit slices. There is a loose coupling between the vector-instruction part and the general-purpose part of the processor: Fetched vector instructions are fed into dedicated command queue and from there they are passed to the vector unit.

The PPU can access and modify the synapse and neuron parameters row-wise, allowing for row-wise parallel plasticity calculations by vector registers. The PPU can change the connectivity of the network by modifying the stored labels in the label comparator of the synapses — e.g. also in Cramer et al. [2019a], Billaudelle et al. [2019a,b] — during operation of the neural network. The user can program the PPU using higher level programming languages such as C or C++, but programming directly via *Assembler* is also possible. PPU-specific complier support is provided by a customized `gcc` [Electronic Vision(s), 2017, Stallman and GCC Developer Community, 2018]. The software in this work was written in C using the vectorized plasticity processing instructions.

Coincidence measurements at the synapses

At each synapse two analog units record and measure the nearest-neighbor correlation between pre- and post-sypnatic spikes. For each pair of spikes, two distinct circuits measure causal (pre-before-post) and anti-causal (post-before-pre) correlations of the spike trains. The correlations are modeled with decaying exponential kernels [Friedmann et al., 2017], inspired by the causal and anti-causal branches of the STDP mechanism found in biology (Markram et al. [1997], Bi and Poo [1998] and section 2.2.3). The measured values are accumulated on two capacitors per synapse until a reset by the PPU. The idealized models of the accumulated coincidence signals are, for the causal branch

$$a^{+} = \sum_{\text{pre-post}} \eta_{+} \exp\left(-\frac{t_{\text{post}} - t_{\text{pre}}}{\tau_{+}}\right) \quad , \tag{4.1}$$

and for the anti-causal part

$$a^{-} = \sum_{\text{post-pre}} \eta_{-} \exp\left(-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_{-}}\right) \quad ; \tag{4.2}$$

with the kernel decay time-constants τ_{+} and τ_{-} as well as the scaling factors η_{+} and η_{-} for the causal and anti-causal branches respectively. The summation corresponds to a flat (non-decaying) eligibility trace. There is leakage from the accumulators due to the imperfection of the manufacturing on a very long time-scale, but not by design intention. The PPU can read the accumulators using a column-wise 8-bit ADC, allowing row-wise parallel readout. As usual on analog circuits, there is variability among the correlation units due to fixed-pattern noise (figure 4.2 A). In this project, only the causal branch a_{+} of the coincidence detection was used.

Temporal and fixed-pattern noise on the BSS-2 neuromorphic chip

Similar to all analog systems, there are severals sources of parameter variability and noise on BSS-2. We have already introduced fixed-pattern noise and trial-to-trial variability in previous chapters (sections 2.3.2 and 3.1), but for the sake of completeness and because they play a prominent role in the presented experiments, we emphasize them again.

Fixed-pattern noise refers to the systematic deviation of parameters (e.g. transistor parameters) from the designed values due to imperfections in the manufacturing process. It is inevitable because it stems from stochastic variations in the process parameters, which is significant in microelectronics. Deviations caused by fixed-pattern noise are constant in time and they manifests themselves in heterogeneous neuron and synapse parameters The magnitude of the caused heterogeneity magnitude can be reduced via calibration. The effect of fixed-pattern noise on the membrane time-constant is shown in figure 4.2 B.

Temporal noise is the collective name of all the effects that influence the dynamics on the chip on the relevant time-scales of the hardware usage, that is approximately from 1 ns to 1 day. The source of temporal noise is diverse: thermal noise, fluctuations in the ambient temperature, instability of the analog storage, crosstalk and fluctuations in the power supply are among the possible causes. Effectively, temporal noise can change the response of a neuron trial to trial even if the input is constant. An example of the effect of temporal noise is shown in figure 4.2 C-D.

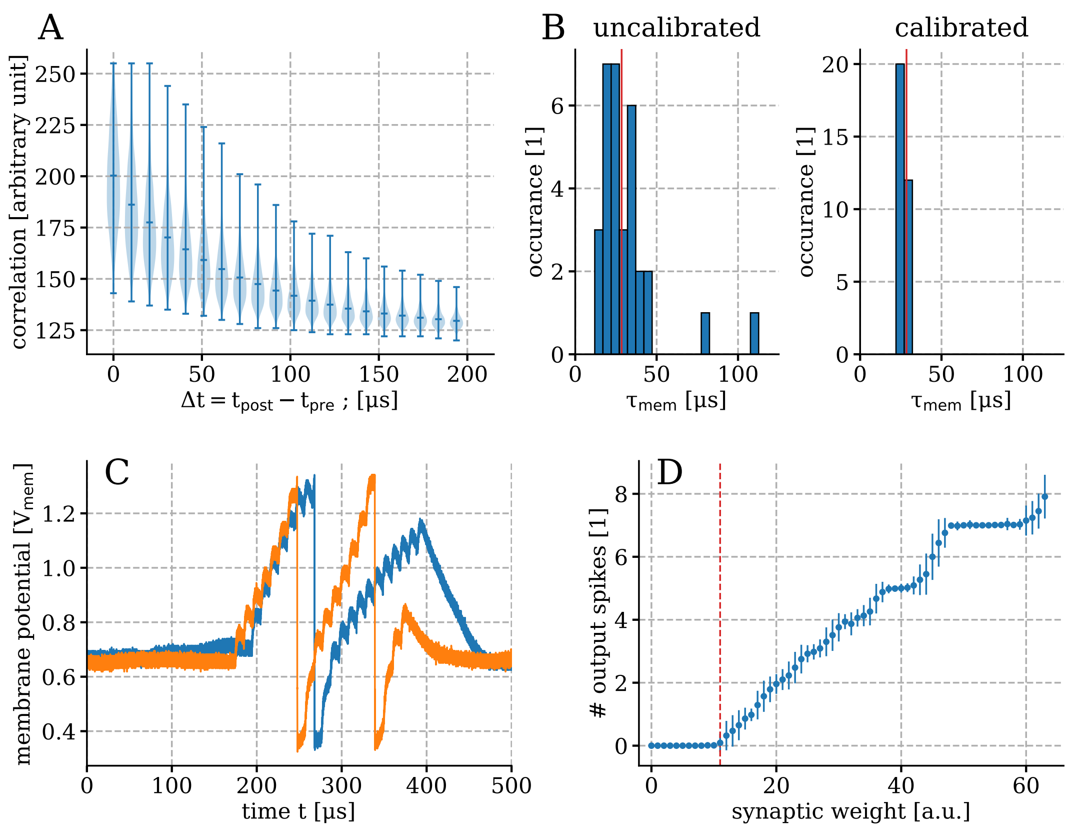


Figure 4.2: Types of noise on BrainScaleS-2 HICANN-DLSv2. (A) Correlation measurement on the causal part of the coincidence detector on the sample chip #1 as a function of time between the pre- and post-synaptic spike event. The violin plot shows the mean, the range and the distribution of measurements. **(B)** Distribution of the realized τ_{mem} membrane time constants before (*left*) and after (*right*) calibration. The target value $\tau_{\text{mem}}^{\text{trg}} = 28.6$ µs is indicated by a vertical line. **(C)** Two traces with identical setups in two different runs. The regular spike train of 20 spikes with 10 µs spacing causes different number of post-synaptic spikes due to temporal noise. **(D)** Response function of a single example neuron excited with the same protocol as in **(C)** as a function of the synaptic weight averaged over 100 trials for each weight. We identify as threshold (*red line*) the lowest weight where spikes are elicited with a probability larger than 5 %. Figure adapted from Wunderlich et al. [2019].

Due to the temporal noise, the neuron elicits a different number of spikes as a reaction to the same spike input in distinct experiments.

Note the difference between the BSS-1 and BSS-2. On BSS-1, the main source of noise is the limited writing accuracy of the analog storage parameters, which introduced a large trial-to-trial variability between experiments (section 3.1). This implicitly also deteriorated the ability of the calibration to compensate for the fixed-pattern noise. On BSS-2, the trial-to-trial variability on the stored analog parameters is much lower, and it is not a limiting factor in experiments. Temporal noise and residual (after calibration) fixed-pattern noise are the main sources of noise, while a clear order of importance is not apparent.

4.1.2 Reinforcement learning with reward modulated STDP

In reinforcement learning, a behaving agent faces the task to maximize its accumulated reward over time by interacting with its environment (Sutton and Barto [2018] and section 2.1.3). In the recent years reinforcement learning — supported by deep learning — has reached remarkable achievements in succeeding in more and more complex simulated environments [Mnih et al., 2013, Silver et al., 2017, Vinyals et al., 2019]. Unfortunately, most of the applied techniques behind these remarkable results apply to neither spiking neural networks nor to time-continuous systems. Mostly they use non-local and complex learning rules — such as backpropagation and experience replay — that are not trivial to implement on neuromorphic hardware. Spike-based reinforcement learning clearly takes place in biology [Guttman, 1953, Fetz and Baker, 1973, Moritz and Fetz, 2011]. It remains an open challenge to create reinforcement learning rules that respect known biological constraints such as locality of information for the plasticity and time-continuous dynamics. This means at the same time that models readily available for implementation on analog spiking neuromorphic hardware are scarce.

The R-STDP [Farries and Fairhall, 2007, Izhikevich, 2007a, Frémaux et al., 2010] is a simple and well-studied three-factor learning rule that combines the causal branch of the classical STDP mechanism and a reward signal as a modulator. The mechanism behind the R-STDP is inspired by biological findings. The phasic activity of dopaminergic neurons (at least of some of them) encodes the reward prediction error [Schultz et al., 1997, Hollerman and Schultz, 1998, Bayer and Glimcher, 2005, Schultz, 2016, Cai et al., 2020] and the dopamine concentration modulates synaptic changes in STDP [Pawlak and Kerr, 2008, Edelmann and Lessmann, 2011, Brzosko et al., 2015]. R-STDP and closely related Hebbian learning rules with reward modulation have been applied to a variety of tasks in simulation-based studies, such as learning periodic activities in recurrent networks [Hoerzer et al., 2014], reproducing biofeedback experiments as in [Fetz and Baker, 1973, Legenstein et al., 2008], simulational reproduction of classical conditioning [Izhikevich, 2007a] and reproducing temporal spike patterns [Farries and Fairhall, 2007, Vasilaki et al., 2009, Frémaux et al., 2010]. A combination of classic unsupervised STDP and R-STDP was sufficient to train deep spiking networks in a heuristic bottom-up approach [Mozafari et al., 2018b] and R-STDP managed to extract more task-relevant features from the input than unsupervised STDP [Mozafari et al., 2018a]. According to our best knowledge; R-STDP has not yet been implemented on analog neuromorphic hardware, but the Pavlovian conditioning experiments found in Izhikevich [2007a] were replicated on the SpiNNaker neuromorphic simulator [Mikaitis et al., 2018].

The R-STDP rule is not derived top-down from first principles but rather motivated heuristically bottom-up [Frémaux and Gerstner, 2015], with the idea that the causal branch of the STDP gathers causality measurements which are then modulated by an obtained reward to elicit a change in the synaptic weights. We postulate the plasticity rule of the form

$$\Delta w_{ij} = \beta (R - b) e_{ij} \quad , \tag{4.3}$$

with the learning rate β, the obtained reward R, the eligibility trace of the causal branch of the STDP curve e_{ij} and a baseline b. The choice of b is in principle arbitrary similarly as in policy gradient methods [Sutton and Barto, 2018], but the choice is crucial because an inappropriate b can introduce reward-independent unsupervised contribution, which is in general detrimental for learning [Frémaux et al., 2010]. By choosing the expected reward $\langle R \rangle$, where the expectation is taken over the probability distribution of the current policy, the weight updates capture the correlation between the obtained reward and the synaptic activity [Frémaux and Gerstner, 2015]

$$\langle \Delta w_{ij} \rangle = \langle \beta \left(R - \langle R \rangle \right) e_{ij}) \rangle = \beta \left(\langle R e_{ij} \rangle - \langle R \rangle \langle e_{ij} \rangle \right) = \beta \, \mathrm{Cov}(R, e_{ij}) \quad . \tag{4.4}$$

With this choice of baseline, R-STDP becomes a statistical learning rule in the sense that it captures the relationship between the synaptic activity and the reward in a statistical manner. In practice, the expected reward $\langle R \rangle$ is not calculated as an expected value but rather estimated with a moving average of the previously experienced rewards.

4.2 Experimental setup

4.2.1 The simplified Pong game and its simulation

We simulated the environment dynamics, the evaluation of the followed policy, the monitoring of the learning progress and the plasticity rule on the PPU. The environment consists of the playing field, the ball and the player's paddle (figure 4.3 A). Three of the four walls of the playing field (top, left, right) are rigid and reflect the ball perfectly elastically. The ball moves with a constant velocity v_b; it starts from the middle of the top wall in a random direction at the start of the game or if the player's paddle missed the ball. The player's paddle can either stay on its current position or move according to the current policy in a given direction with a constant velocity of v_p.

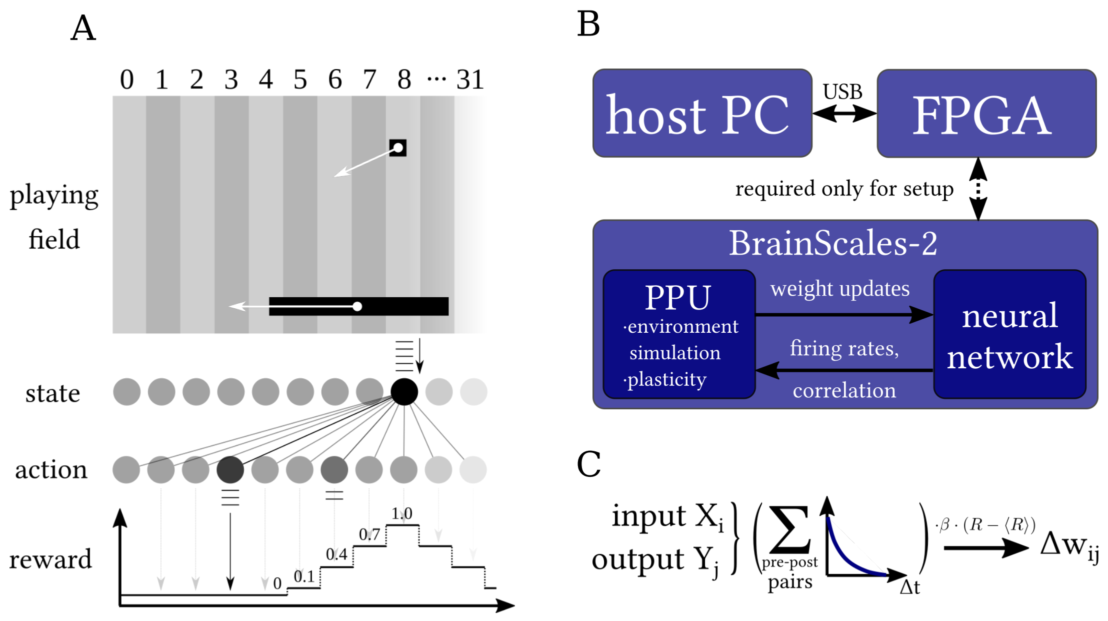


Figure 4.3: Experimental setup. (A) The playing field constitutes of three reflecting walls, a ball and a paddle. In the panel: the ball is currently in the 8-th column, hence the 8-th state sends regular spikes to all action neurons. The neuron at index 3 reacts with the most spikes, hence the paddle moves towards the 3rd vertical position. Because the index 3 lies far away from the 8-th column, the network receives zero reward (equation (4.5)). **(B)** The experiment execution is entirely contained on the neuromorphic chip. The PPU handles the environment simulation, the generation of input spikes and the plasticity. The FPGA (connected to the host via USB) is only used for the initial setup. **(C)** Schematics of the learning rule. The causal (pre-before-post) spike pairs of the input X_i and output unit Y_i are exponentially weighted with the temporal difference $\propto \exp\left(-\Delta t/\tau_+\right)$ and summed into a flat (non-decaying) eligibility trace. This sum is then modulated by the learning rate β and the reward-prediction error $R - \langle R \rangle$ to obtain the synaptic update Δw_{ij}. Figure adapted from Wunderlich et al. [2019].

The paddle is controlled by a two-layer fully-connected feed-forward actor-network with 32×32 neurons in the layers and with purely excitatory synapses. The playing field is discretized into 32 equal segments along the bottom wall giving rise to 32 horizontal positions or states u_i with $i \in [0, 31]$. The input layer of the actor-network represents the 32 possible positions of the ball, while the output layer represents the target positions of the paddle. The input layer is virtual; that is the input neurons are not implemented by physical neuron circuits, they merely act as spike sources. The output neurons are implemented in the neuromorphic core of the BSS-2. If the ball is in section u_i, the input i provides a finite regular spike train to all its post-synaptic neurons. The winner is determined as the action neuron with the highest spike count ρ_i after the injections of the spike train, that is $j = \text{argmax}_i\,(\rho_i)$. If the target column is the same as the current position of the paddle's center then the paddle does not move. If there are several action neurons with equal number of spikes, then the winner action is chosen randomly.

The player obtains reward based on the aiming accuracy with the paddle:

$$R = \begin{cases} 1 - |j-k| \cdot 0.3 & \text{if } |j-k| \leq 3, \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where j is the position of the ball and k is the target position for the paddle. The player receives more reward for accurate aiming than for slightly off aiming. The seven positions width of the reward window matches the size of the paddle. The graded reward scheme is a heuristic with the aim of rewarding the network for good enough aiming but at the same time encouraging aiming to the middle of the paddle.

4.2.2 The implemented plasticity rule

The experiment runs in an iterative way: 1) The positions of the ball and of the paddle are updated, 2) the new position of the ball is accessed and the actor network receives input accordingly, 3) the actor network determines the target position of the paddle and we calculate the reward, 4) we update the synapses according to the plasticity rule, 5) the mean expected reward is updated. If the player loses the ongoing game (ball touches the lower wall), the position of the ball is reset and the ball starts in a random direction. A flowchart of the game-loop is shown in figure 4.4.

For each potential position of the ball, we calculate and save the task-specific (position-specific) mean expected reward $\bar{R}_k; k \in [0, 31]$ and estimate it effectively using an exponentially-weighted moving average:

$$\bar{R}_k \to \bar{R}_\text{k} + \gamma(R - \bar{R}_k) \quad , \tag{4.6}$$

where γ is the discount factor for the moving average. We use a task specific reward for the 32 tasks in our setup because task specificity is required for learning multiple tasks [Frémaux et al., 2010]. In the somewhat surprising terminology of reinforcement learning the 32 states of the ball are considered as 32 distinct

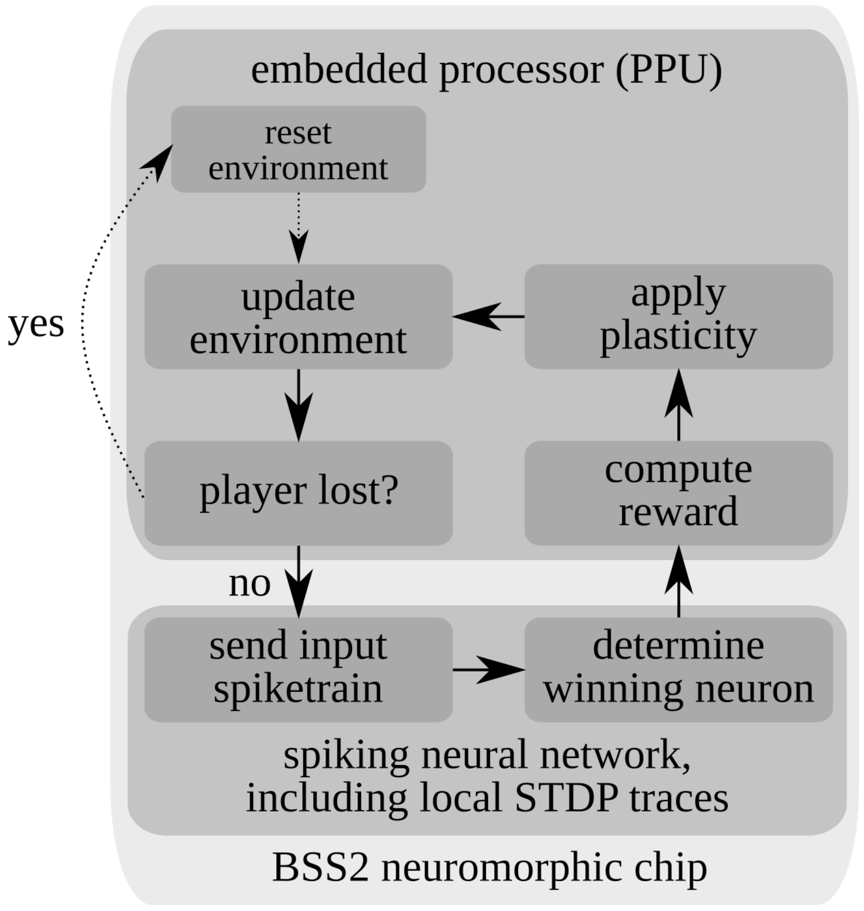


Figure 4.4: Flowchart of the implementation. The experiment runs in loop iterating between the PPU and the neuromorphic core. The game is started/reset by setting the ball in the middle of the playing field and releasing it in a random direction. Based on the current position, the environment sends a regular spike train to the neuromorphic core through the corresponding state; the winning neuron is determined as the most active neuron. The reward is determined and based on the distance between the ball state and the target position and the plasticity is applied (equation (4.3)). Finally, the PPU updates the environment: the ball moves on and the paddle moves towards the target position; and the loops starts again. If the player looses the game (the ball drops to the lower wall), the environment is reset. Figure taken from Wunderlich et al. [2019].

(multi-armed bandit) tasks, as e.g. in Sutton and Barto [2018]. The weights are initialized with a Gaussian distribution and we update all the 1024 synapses based on the R-STDP learning rule (equation (4.3)),

$$\Delta w_{ij} = \eta(R - \bar{R}_k)A^+_{mn} \quad , \tag{4.7}$$

where A^+_{mn} is a processed version of the accumulator value a^+_{mn} (equation (4.1)) on the causal branch. The accumulator value is digitized to 8-bit resolution, calibrated against the offset value, and bit-shifted to the right, meaning that the often noisy least-significant bit is thrown away. Note, that we update all the synapses on the array although we would expect that only synapses in a single row should experience any updates, the others should have exactly zero update because the STDP trace is zero. This holds for this specific application, but we want to use the results of the experiments as a pilot study for large-scale experiments where the distinction between used synapses and silent synapses is not obvious. Further, even the unused synapses could contain nonzero values on the coincidence accumulators, for example through a malfunctioning reset. By updating all the synapses, thus refrained from using expert knowledge, we preserve the generality of our findings for larger networks. This will also be important, when we compare the implementation on neuromorphic hardware to computer simulations. By not including expert knowledge, we keep the network setup scalable (no elaborate blacklisting of faulty synapses) and hence we preserve the generality of the results.

Monitoring the learning progress

We monitor the learning with two observables. The accumulated mean expected reward is defined as

$$\langle R \rangle = \frac{1}{32}\sum_{i=0}^{31} \bar{R}_i \, , \tag{4.8}$$

which is the average expected reward over all the 32 positions (inputs) at a given iteration. Because of the graded reward scheme over the length of the paddle, the mean expected reward is only a proxy for the ability of the player to catch the ball; even a slightly off paddle can catch the ball. To access the playing capability more precisely, we introduce the measure of *performance*

$$P = \frac{1}{32}\sum_{i=0}^{31} \lceil R_i \rceil \, , \tag{4.9}$$

where $\lceil \cdot \rceil$ is the ceiling operator and R_i is the last reward received by state i. The performance reflects the ratio of states in which the aiming is accurate enough such that the paddle can reflect the ball. The used parameters of the environment simulation and the plasticity are give in table 4.1.

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 µs
w	mean of initial weights (digital value)	14
σ_w	standard deviation of initial weights	2
L	length and width of quadratic playing field	1 m
$\|\|v_p\|\|_1$	L1-norm of ball velocity	0.025 m per iteration
v_p	velocity of paddle controlled by BSS2	0.05 m per iteration
r_b	radius of ball	0.02 m
r_p	length of paddle	0.20 m
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
$\tau_{\text{syn}}^{\text{exc}}$	LIF excitatory synaptic time-constant	1.8 µs	2.4 µs	1.4 µs
E_{leak}	LIF leak potential	0.62 V	0.56 V	0.87 V
V_{reset}	LIF reset potential	0.36 V	0.36 V	0.30 V
V_{thresh}	LIF threshold potential	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 4.1: Parameters used in the experiment. Abbreviations in the table: LIF: leaky integrate-and-fire; ISI: inter-spike interval. The three parameter sets for the three chips are the results of the meta-parameter optimization. We describe quantities of the playing field, such as the length of the paddle, in meters to give them a dimension and to distinguish them from dimensionless quantities. If not mentioned otherwise, the experiments were carried out on chip #1. Table adapted from Wunderlich et al. [2019].

Meta-parameter optimization

We performed meta-parameter optimization of the time-constants $(\tau_{\mathrm{mem}}; \tau_{\mathrm{syn}}^{\mathrm{exc}}; \tau_{\mathrm{ref}})$, the neuron potentials $(E_{\mathrm{l}}; V_{\mathrm{reset}}; V_{\mathrm{thresh}})$ and the amplitude of the coincidence detectors $(\tau_{+}; \eta_{+})$. For the optimization we used the decision-tree-based optimization algorithm FOREST_MINIMIZE from the SCIKIT-OPTIMIZE [Head et al., 2018] software package with default settings (extra trees regressor model, 10^5 acquisition function samples, maximizing expected improvement, target improvement 0.01). The meta-parameter optimization serves several ends: 1) it helps to explore the parameter space and sets the parameters of the model to a good working point, 2) it ensures that the results can be compared between hardware and simulation results 3) finally, meta-parameter optimization builds the basis of our transfer experiments, where we study if results obtained on one BSS-2 chip can be applied on another chip of the same generation.

4.2.3 Reference software simulation using the Nest simulator

To establish a comparison for our results on the BSS-2, we implemented the same actor network using the Nest v2.14.00 spiking neural network simulator [Peyser et al., 2017]. We used the *iaf_psc_exp* model from the neuron model library of Nest, a LIF neuron model with CUBA synapses and exponential synaptic kernel. Fixed-pattern noise was omitted from simulation, meaning that all the simulated neurons had identical parameters. The LIF parameters in simulation were equal to the target values of the LIF parameters on the BSS-2. The weights were scaled to a dimensionless quantity, discretized and matched via their effect in terms of the response function to match the weights on the BSS-2. We chose the STDP parameters η_{+} and τ_{+} to match the mean values on the neuromorphic hardware. To include noise into the Nest-based simulations, we used Nest's *noise_generator* to inject Gaussian current onto the neuron's membrane. Learning rate and the parameters of the game dynamics were the same as in the hardware implementation.

The plasticity updates were not calculated in Nest, because at the time of this project there was no official (off-the-shelf available in the used version) R-STDP model implemented in Nest. For fair speed and energy comparison we implemented the calculation of the a^{+} values in a custom Python code based on the spike-timings from Nest, and we excluded the time and energy spent on plasticity calculations to achieve a conservative comparison between neuromorphic hardware and off-the-shelf simulator. In contrast to the hardware implementation, in each iteration only synapses were updated that transmitted spike events. In a software simulation we can guarantee that the a^{+} values equal zero for the silent synapses. This choice also reduced the time and energy spent on the plasticity calculations.

4.3 Results

4.3.1 Learning performance

We monitored the progress of learning both on BSS-2 (figure 4.5 A) and using Nest without (figure 4.5 B) and with additional temporal noise (figure 4.5 C) for 5×10^5 iterations both. Both measures, the mean expected reward (equation (4.8)) and the playing performance (equation (4.9)), grow as the learning progresses for both the neuromorphic implementations and the software simulation with noise. The learning was repeated 10 times and the results are reproducible with some expected stochastic deviations between the trials.

The perfect solution of the simplified Pong task is a one-to-one mapping from the ball position to the same target position for the paddle. Hence, we expect that starting from randomly initialized weights, the learning should result in a diagonal-dominant weight matrix; and this is what we observe in the neuromorphic implementation (figure 4.6 A). The obtained final weight matrix is diagonal dominant, but there are deviations from an envisioned perfect solutions. The salt&pepper-like randomness of the weights is a combined result of the random initialization, the stochastic nature of learning and the fact that the learning does not aim at reaching a perfect weightmatrix, but at accumulating reward. The first, second and third diagonals are also strengthened due to the graded reward scheme. They correspond to the "reward classes" 0.7, 0.4 and 0.1. Finally the vertical stripes are an implicit result of the adaptation to the fixed-pattern noise. The fixed-pattern noise is most apparent on the BSS-2 in the neuron-to-neuron deviations because the PSP-shaping of the spikes happens at the neurons (section 4.1.1).

Temporal noise as a resource of exploration

On the BSS-2, the exploration of actions is provided solely by the temporal noise on the system, and this noise is sufficient to enable learning (figure 4.5 A).

In contrast, the software simulation without noise is unable to learn and both measures get stuck at approximately chance level (figure 4.5 B). Chance level for the mean expected reward is

$$\langle R \rangle_{\text{chance}} = \frac{1}{32} \left(1 \cdot 1 + 2 \cdot 0.7 + 2 \cdot 0.4 + 2 \cdot 0.1\right) \approx 0.1 \quad , \tag{4.10}$$

and for the performance

$$P_{\text{chance}} = \frac{1}{32} \left(1 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1\right) \approx 0.22 \quad , \tag{4.11}$$

assuming equal chance to win for all the action neurons. There are only two sources of stochasticity in the system. First, the random initialization of the weights. Second, if two or more action neurons elicit the same number of spikes, then the action is chosen randomly among them. But they are not sufficient for learning.

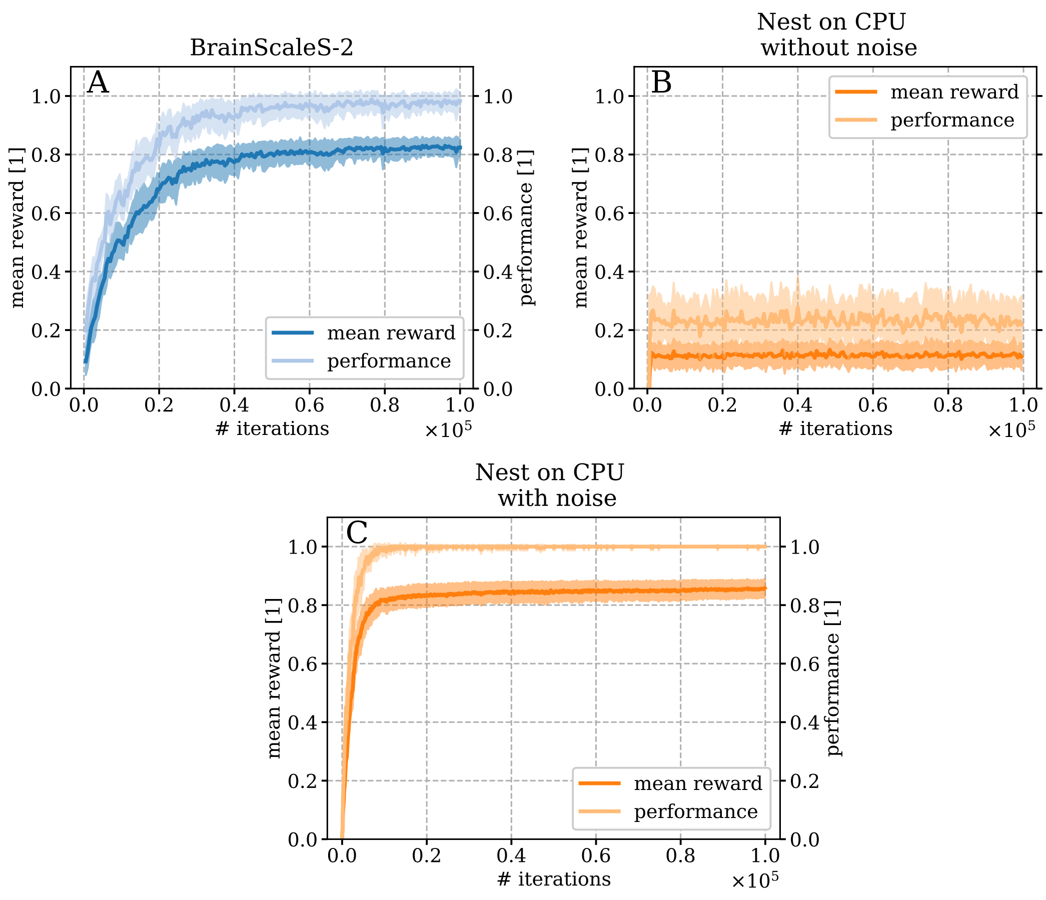


Figure 4.5: Learning on BSS-2 and with Nest on a CPU. The plots show the mean expected reward (equation (4.8)) and the performance (equation (4.9)) for both cases with the mean and the standard deviation over 10 experiments. **(A)** The BSS-2 can learn even without explicit noise, because the internal temporal noise sufficiently facilitates exploration. **(B)** The simulation is unable to learn beyond chance level without additional noise. **(C)** When we add a Gaussian current based noise with a standard deviation of $\sigma = 100\,\text{pA}$, exploration sets in and learning becomes possible. The simulation converges faster in terms of number of iterations than on BSS-2 due to the lack of fixed-pattern noise. Figure adapted from Wunderlich et al. [2019].

If we inject a Gaussian noise to the network with a standard deviation of $\sigma = 100\,\mathrm{pA}$, then exploration becomes possible and the learning sets in (figure 4.5 C). With Gaussian noise, the network in software simulation learns faster and converges to higher values than on the BSS-2. The higher performance and mean reward stem from the lack of fixed-pattern noise in simulation: the network starts out from a balanced weight matrix (random initialization but no fixed-pattern noise) and does not have to spend synaptic updates and learning iterations to compensate for the heterogeneity of the neurons. Further, the accumulator circuits also show deviations introduced by fixed-pattern noise (figure 4.2 A). In turn, these also lead to deviations in the weight updates. By comparing the simulations with and without Gaussian noise, we also implicitly verify that the exploration and hence the learning on BSS-2 is indeed a result of the temporal noise and not of the simpleness of the task and the network.

Our experiments show that temporal noise on neuromorphic hardware, which is usually regarded as nuisance, can be exploited as useful resource. Certainly, we could replace the temporal-noise-driven exploration by an explicit exploration e.g. ϵ-greedy policy, but this would require additional resources either on the neuromorphic core or on the PPU.

4.3.2 Learning is calibration

Learning means changing the own parameters in a way to improve the capability of solving a given task. Learning on an imperfect substrate also implicitly includes adaptation to fixed-pattern noise-induced variations (figure 4.2 B) giving rise to the notion that learning is calibration. But this neat and useful attribute of learning is elusive in experiments, as it, by definition, only appears as a side-effect modulating the central challenge, namely learning the task. In the following we show in two observations how learning compensates for the inhomogeneities of neural excitability on BSS-2.

The compensation for the fixed-pattern noise is apparent in the evolution of the unrewarded synapses. Each input neuron projects with 32 synapses to 32 action neurons. 7 of these synaptic connections are rewarded with non-zero reward such that we expect that they will be strengthened to some extent if the corresponding action neuron is the winner. These 7 rewarded synapses form the dominant diagonal of the final weight matrix (figure 4.6 A). The other 25 of the synapses are unrewarded and should be pushed below the spiking threshold to prevent the corresponding action neuron from winning. Indeed, we find that the final value of the non-rewarded synapses falls below the firing threshold (figure 4.6 B). The threshold weight of the neurons correlates with the final weight-value of the unrewarded synapses. This is caused by the fixed-pattern noise: due to the fixed-pattern noise, the firing threshold is different for each neuron, and hence the learning stops at different weight values.

In a next experiment, we demonstrate the adaptation by shuffling the assignment of logical neurons to the neuron circuits on the chip after an initial learning. The logical weight matrix is kept intact, that is each logical neuron sees the same

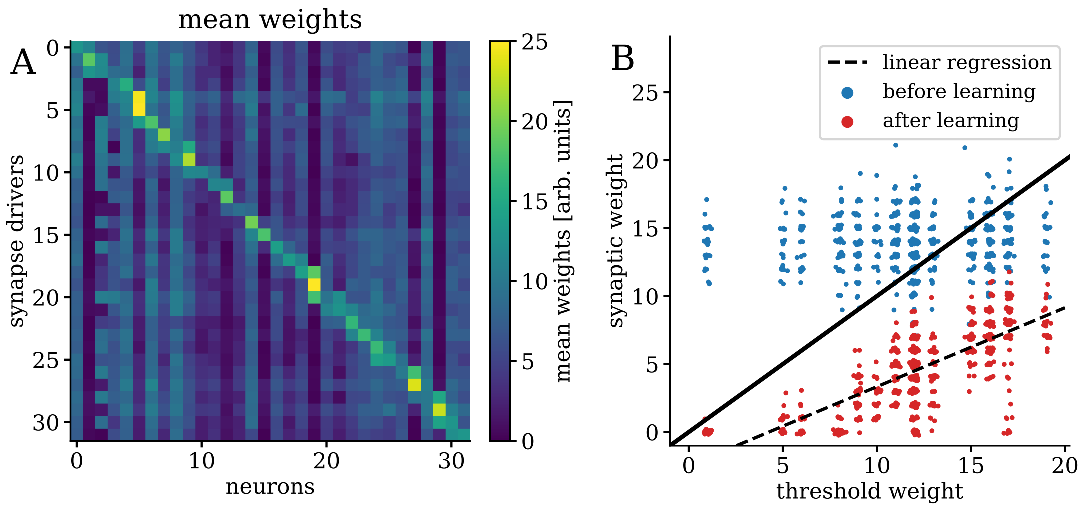


Figure 4.6: Learning is calibration. **(A)** Final weight matrix after learning on BSS-2 averaged over 10 experiments from figure 4.5 A. The noticeable vertical stripes are the indirect result of the fixed-pattern noise. The fixed-pattern noise causes the largest differences neuron-to-neuron, because the PSC is generated at the neurons [Fehre, 2017]. Because the weights adapt to the excitability of the neurons, the vertical stripes appear on the weight matrix after learning. The plasticity calibrates for this fixed-pattern noise, which is in-turn apparent on the final weight matrix. **(B)** The compensation of the fixed-pattern noise is most apparent on the unrewarded synapses, as they are systematically pushed below the spiking threshold. This leads to correlation between the learned weights and the spiking threshold with a Pearson's $r = 0.76$ and $p < 0.001$ using two-sided Wald test with t-distribution (SCIPY.STATS.LINREGRESS function from the scipy library [Jones et al., 2001–]). The weights are plotted with jitter for better visibility. Figure adapted from Wunderlich et al. [2019].

input weights in bit values, but not in their effect. After learning, the synapses are adapted to the excitability of the post-synaptic neuron. By shuffling the assignment we disturbed this relationship and the synapses are maladapted to their new target neurons. If learning is indeed calibration, then further learning should restore the performance of the network.

To establish a baseline, we trained the network for 5×10^4 iterations and measured the final reward distribution over 100 distinct experiments (figure 4.7 A — top panel). The "reward distribution" refers to the most recent obtained reward by the 32 input states at the end of the learning. In the baseline measurement, the network reached a mean expected reward $\langle R \rangle = 0.73 \pm 0.09$ and a performance of $P = 0.85 \pm 0.06$, respectively.

Afterwards we shuffled the assignment of the logical neurons to the hardware neuron circuits. We measured the reward distribution as in the baseline case but with learning turned off (figure 4.7 A — middle panel). The reached mean expected reward dropped to $\langle R \rangle = 0.37 \pm 0.09$ and the reached performance to $P = 0.47 \pm 0.11$.

Finally, starting from the same weight matrix, we trained the network (with enabled learning) for 5×10^4 iterations. The reward distribution was restored close to the baseline value (figure 4.7 A — bottom panel), reaching a mean expected reward of $\langle R \rangle = 0.67 \pm 0.07$ and a performance of $P = 0.81 \pm 0.09$.

Our experiments demonstrate that the learning can adapt to the inhomogeneities of the neurons and can implicitly compensate for them. This implies that learning on the chip can reduce the requirements on the calibration and on the manufacturing quality.

4.3.3 Robustness of learning

When using analog neuromorphic hardware, or in general any physical model system, an important question is the robustness of the implemented network and learning rule (compare to chapter 3). The potential advantages in terms of speed and energy consumption come with a price-tag: The implementation has to cope with the limited controllability, precision and range of the parameters on the hardware.

As before, one concern is the fixed-pattern noise. It can be reduced by calibration [Stradmann, 2016, Aamir et al., 2018], but only at a price. A precise calibration would require the tedious measurement, mapping and storage of a high-dimensional parameter space from the targeted LIF values to the hardware parameters. A simpler calibration that assumes independence between each parameters would be faster but would introduce systematic deviations due to the neglected interactions. The calibration still has to be done for each chip. Even the possibility of calibration requires chip area, because it needs large tunable neurons. We study the effect of missing calibration on the results.

Furthermore, results (network architectures, good parameters etc.) found on one chip should apply to other realizations of the same chip version as well to enable a broad and scalable usage of neuromorphic systems. But remaining fixed-pattern

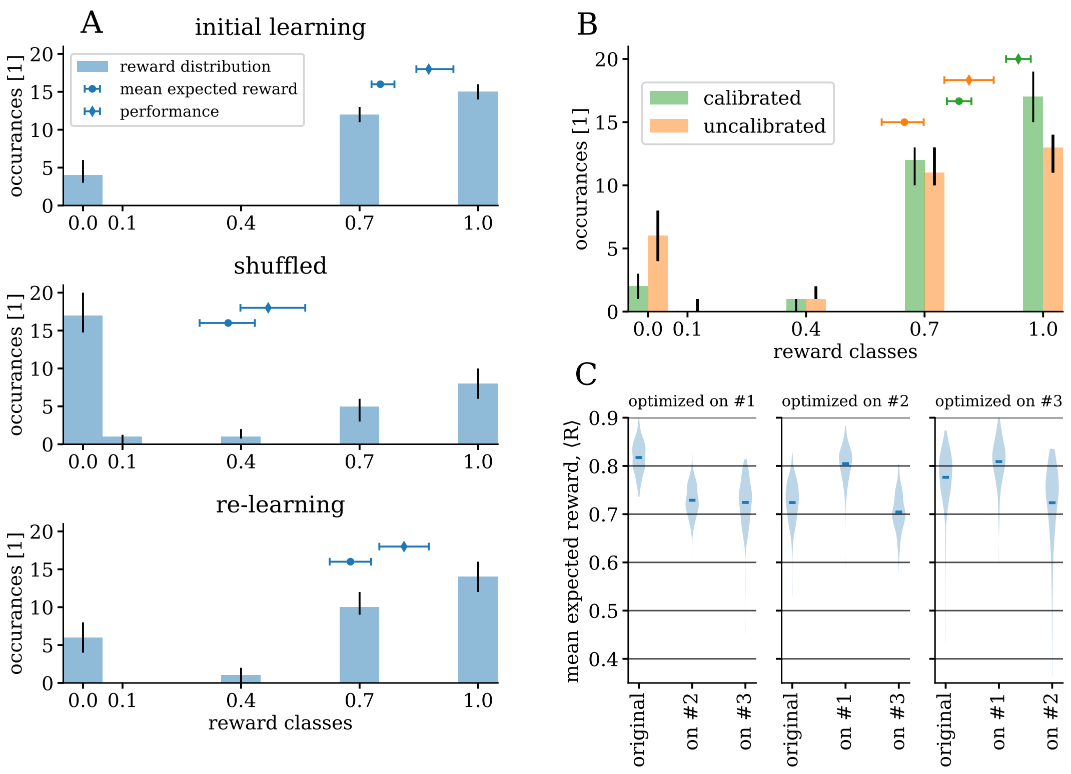


Figure 4.7: Robustness of learning. The learning compensates for the variation of individual neuron parameters as demonstrated in the following experiments. **(A)** *Top:* Distribution of the received reward over 100 experiments after learning. *Middle:* Same distribution after randomly shuffling the mapping of the abstract neurons to the neuronal circuits. The mean reward dropped since the learning adapted the individual neuron circuits, and with the shuffling this adaption is not correct any more. *Bottom:* After additional 5×10^4 learning iterations following shuffling, the network adapted to the new mapping and the rewards are largely restored. The experiments were carried out with uncalibrated neurons. The mean expected reward and performance values are shown with median and interquartile range. **(B)** Reward distribution in 100 experiments after 5×10^4 learning iterations with calibrated and uncalibrated neurons. The reached mean expected reward (cross symbol with interquartile range) and performance (diamond symbol with interquartile range) are similar showing that learning largely compensates for the missing calibration. **(C)** Violin plots of the mean expected reward with mean. We performed the meta-parameter optimization on the indicated chip and then tested the obtained meta-parameters on the two other. The obtained mean expected rewards largely agree, there is no systematic drop in the reward after transfer to other chips. We considered a system as a combination of the chip and the corresponding calibration. Figure adapted from Wunderlich et al. [2019].

noise could affect the transferability of the results. We study the transferability of results on the example of meta-parameter learning.

All experiments in the following were carried out with 5×10^4 learning iterations each.

4.3.4 Impact of missing calibration

Until this point, we used the calibration of the LIF time-constants $(\tau_{\mathrm{mem}}; \tau_{\mathrm{ref}}; \tau_{\mathrm{syn}}^{\mathrm{exc}})$ which reduced the deviation between realized and target LIF parameters (section 4.1.1). On the BSS-2 the calibration can typically reduce the standard deviation of the realized parameters by an order of magnitude [Stradmann, 2016, Aamir et al., 2018].

Next, we studied the effect of uncalibrated LIF time-constants on the learning. To preserve a good working point, we defined the uncalibrated hardware parameter as the average of the calibrated ones given a specific target value. A comparison of the realized time-constants is given in figure 4.2 B.

We measured the distribution of the rewards with and without calibration over 100 experiments each (figure 4.7 B). Learning was possible even in the absence of calibration, however the reached mean expected reward suffered a loss of approximately 17 %. With calibration the learning reached a mean expected reward of $\langle R \rangle = 0.79 \pm 0.05$ and a performance of $P = 0.93 \pm 0.05$, while without it reached only $\langle R \rangle = 0.65 \pm 0.08$ and $P = 0.80 \pm 0.09$.

We conclude that the implemented model is robust against fixed-pattern noise typical to the BSS-2 in the sense that learning is still possible without calibration and the performance suffer some loss, but without a dramatic drop to chance level.

4.3.5 Transferability of the results between chips

Until now, we conducted all the presented experiments on the chip #1 using parameters found in meta-parameter optimization (table 4.1). Now we would like to see if the results found in the meta-parameter optimization on one chip can be applied on another chip. Note, that this procedure is not only about inference on different chips. On each chip, we perform learning to enable adaptation to the chip-specific fixed-pattern noise, which is central on analog hardware (section 4.3.2).

To study the transferability of the results, we took three different BSS-2 chips of the same generation. For each potential transition, we first applied meta-parameter optimization on the original chip and tested the resulting parameters in 200 experiments with learning for 5×10^4 iterations in each experiment. Then we applied the same meta-parameters and tested them on a different chip in identical (in terms of target parameters and setup) experiments. We consider as a chip the combination of the neuromorphic substrate and the corresponding calibration.

We found no apparent drop in the reached mean expected reward after transition between the chips (figure 4.7 C). There is a clear ordering between the chips as chip #1 performs better than the two other chips irrespective where the meta-parameter optimization was performed. We explain the observed differences by deviations during the manufacturing process. Still, the game could be learned successfully in each experiment, and there is no systematic drop in performance after transfer to another chip.

We conclude that results achieved on a realization of BSS-2 apply to other realizations, meaning that the chips can be used as drop-in replacements for each other and that this neuromorphic chip could be used in applications where scaling and performance consistency over the several chips is an issue.

4.3.6 Computation speed and energy consumption

The central attribute of the BSS-2 system is its accelerated emulation of neural activity, which is 10^3 times faster than the biological equivalent time. While speed is the primary advantage, the accelerated emulation on analog substrate also leads to low energy consumption. We stress that unlike other systems that aim at low power consumption [Qiao et al., 2015, Moradi et al., 2017, Neckar et al., 2018], the BSS-2 chip features low energy consumption. The BSS-2 system performs the calculations with at a higher power-consumption but at the same time in shorter time.

To put this into perspective, we compared the emulation on BSS-2 to computer simulation running the Nest simulator on an Intel i7-4771 CPU using one core (more cores would not result in any perceivable speed-up due to the small size of the network) in terms of speed and energy consumption.

Speed measurements

A full iteration in software simulation took 50 ms, but most of it was spent on the custom-code-based plasticity calculations, which we disregard to obtain a fair and conservative comparison. The pure network simulation (Nest's *Simulate* routine) of 200 ms network activity took 4.3 ms with noise (Nest's *noise_generator*) and 1.2 ms without it (figure 4.8). A small negligible time was spent on the calculations for the environment, but we also disregard them for the same reason as in case of the plasticity rule. In contrast, a full iteration on the BSS-2 took 0.4 ms measured using the time stamp register on the PPU. This time was approximately equally divided between the emulation of the network activity and the combination of plasticity and environment calculations.

Even in this conservative comparison, the emulation of the model was at least three times faster, 0.4 ms on the BSS-2 against 1.2 ms in software simulation using Nest. The total time of a full experiment with 5×10^4 iterations took 25 s on BSS-2 (including 5 s overhead for applying the calibration and chip setup) and 40 min in software simulation. We remark, that including the plasticity calculation into the comparison would increase the advantage of BSS-2 further even if we had used a Nest native plasticity model. In software simulations, the time spent on the eligibility trace calculations roughly scales linearly with the number of synapses, while on BSS-2 it happens parallel with the network emulation.

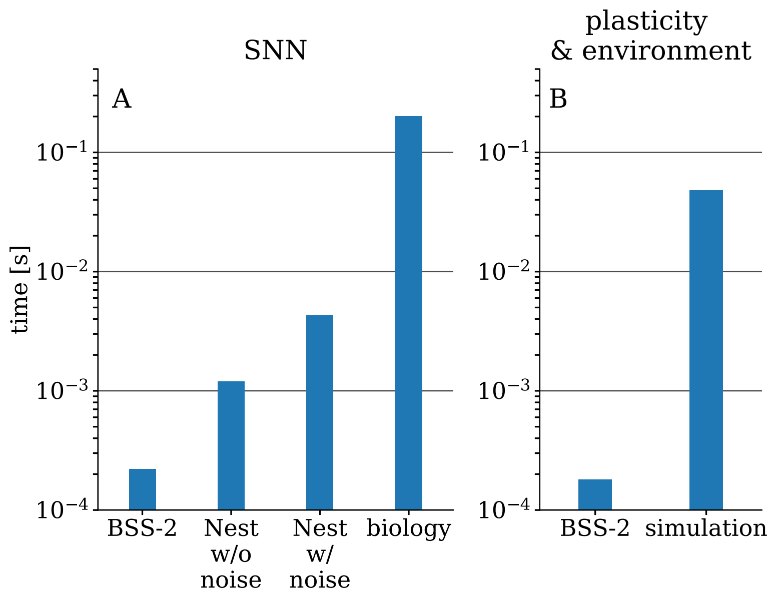


Figure 4.8: Speed comparison between BSS-2 and Nest. (A) A single experiment, that is an iteration of the loop (figure 4.4), on the BSS-2 takes 400 µs with 220 µs spent on the network emulation and 180 µs on the plasticity rule. The network was emulated for corresponding 200 ms biological time. In software simulation, the state-propagation of the network activity took 4.3 ms with noise and 1.2 ms without noise. **(B)** The environment and the plasticity calculation took 50 ms with Nest and 180 µs on BSS-2. Figure adapted from Wunderlich et al. [2019].

Energy consumption measurements

We measured the power consumption of the CPU as a difference measurement between the power consumption during simulation and when idling using the Entry-Level Power Supply (EPS) 12 V power supply cable. The EPS cable connects to the CPU socket of the motherboard and powers the CPU. We found that a lower bound for the power consumption is 24 W without simulating noise and 25 W when using noise. Using that a single iteration took 1.2 ms without noise, we arrive at an energy consumption of 29 mJ. With noise (4.3 ms simulation time) we obtain an energy consumption of 106 mJ.

We accessed the energy consumption on the BSS-2 by measuring the current drawn during the experiment. We measured a power consumption of 57 mW, which is consistent with previous power consumption measurements of the same chip generation [Aamir et al., 2018]. This excluded the energy consumption of the FPGA which is only used during the initial configuration. With 0.4 ms per iteration, this leads to an energy consumption of 23 µJ per iteration on the chip.

We conclude that the energy consumption of the emulation on the BSS-2 chip is at least three orders of magnitude smaller than that of a software simulation on a CPU.

4.4 Discussion

In this project we demonstrated and analyzed the advantages of neuromorphic computation on the example of an implementation of R-STDP on the BSS-2 HICANN-DLSv2 prototype chip in terms of emulation speed, energy consumption and robustness of the results. Further, we showed how learning can compensate for the inevitable distorting effects of fixed-pattern noise on analog hardware, hence reducing the precision requirement of the calibration.

We found that temporal variations on the chip can be exploited as a computational resource. This temporal noise is not present on the chip by design, and in general designers aim at reducing its amplitude. Still, it could be of interest for the general public, that they can be leveraged as computational resource. The successor of the HICANN-DLSv2 prototype chip will feature on-chip stochastic spike train generators to enable the injections of Poisson-like spike trains to the neurons in a controllable way [Billaudelle et al., 2019b, Müller et al., 2020a, Schemmel et al., 2020][2]. With these noise generators it will be possible to enable the emulation large-scale stochastic networks similar to the Stochastic Sampling Network (SSN) shown in chapter 3.

[2]The next generation chip is described in detail in [Schemmel et al., 2020] and it has been used for experiments as described in [Billaudelle et al., 2019b]. Müller et al. [2020a] describe the software stack of BSS-2 system.

4.4.1 Limitations of the study

The main limitation of the study is the size of the prototype chip which only allowed the emulation of small simple networks and learning tasks. In the world of large-scale simulations with at least several thousands of neurons [Merolla et al., 2014, Jordan et al., 2018, van Albada et al., 2018], emulating 32 neurons is rather small and hence limits the validity of our results.

In terms of speed and energy efficiency, we expect that the advantages would become more significant with larger network sizes. One of the main features of physical modeling is the independence of emulation time of the network size. In our case, the relative swiftness of the software simulation stems from the small size of the network. With the physical computing approach, the emulation of the neuron dynamics and the accumulation of the eligibility traces happens simultaneously at the 10^3 times accelerated speed independent of the size of the emulated network. In the next generation, two PPU units will be placed per 512 neurons, making the plasticity calculations similarly fast as in this project. As long as the calculations stay local to the PPUs the $\mathcal{O}(1)$ scaling property is preserved. In contrast, state-of-the-art large-scale software simulation requires minutes to simulate one biological second [Jordan et al., 2018]. Even on a specialized digital neuromorphic simulator, the best results of reported simulations are at real-time [van Albada et al., 2018, Rhodes et al., 2019]. A potential pitfall of this prediction appears if the emulations become dominated by the overhead, such as the mapping procedure or the setup of the system parameters (van Albada et al. [2018], Kungl et al. [2019]; chapter 3).

The low number of chips limits the validity of the transferability results. Because the used neuromorphic BSS-2 prototype chips are not the result of mass-production but of ongoing research, the commissioning of each chip is a tedious process. We consider our experiments sufficient for a pilot study.

The accelerated emulation is a merit when we want to emulate experiments as fast as possible, but it is a drawback if we consider interfacing with robots. Naively, robots interact and act on a time-scale similar to biological neurons, that is on the order of milliseconds. One would expect that neuromorphic systems running at real-time are better suited for robotics. But there are applications where the time-scale is on the order of microseconds, for example in adaptive beam shaping in radar systems. Also for a sampling-based neural network (e.g. in chapter 3), the acceleration could be beneficial: the network has sufficient time to explore the state-space and infer an optimal reaction to the current state of the environment.

The simplicity of the applied R-STDP model was not only a limitation but also a merit for our project. This well-studied and easy-to-implement model both fitted the resources of the prototype chip and at the same time its simplicity allowed us to focus on the different aspects of learning and to easily interpret the results. We expect that further larger chips of the BSS-2 generation will, in principle, enable the emulation of more complex networks. However, further theoretical work is required since most state-of-the art models, such as AlphaGo Zero [Silver et al., 2017] or AlphaStar [Vinyals et al., 2019], are far away from a direct implementation to a spiking neuromorphic substrate. In computational neuroscience, there are models solving reinforcement learning problems with sparse rewards, which would be good candidates for neuromorphic implementation. For example, the TD-STDP (temporal difference STDP) [Frémaux et al., 2013] implements actor-critic learning in spiking neural networks, and it already matches the designed capabilities of a larger BSS-2 chip.

The complexity of the environment simulation is strongly limited by the capabilities of the PPU. Clearly, the PPU was not designed to be an elaborate processor for complex environment simulations but rather for fast local plasticity calculations. The simplicity of the environment, like in case of the plasticity rule, was a merit and allowed us to focus on and interpret aspects of the learning. A more complex environment could be simulated on a separate system, for example on an FPGA, which then only communicates with the neuromorphic chip via input (state, reward) and output (actions) spike-trains and events.

4.4.2 Outlook

Our work not only demonstrated and quantified the advantages of the accelerated analog neuromorphic approach, but also laid a groundwork for implementing reinforcement learning in a spiking neuromorphic network. We firmly believe that based on this groundwork future studies using more neuromorphic resources will show more elaborate agents acting in complex environments guided by reinforcement learning, such as the insect navigation task in Billaudelle et al. [2019b] and Schreiber et al. [2020].

5 Biologically plausible deep reinforcement learning in a time-continuous framework

The project in this section was done in close collaboration with Walter Senn, Dominik Dold, Oskar Riedler and Mihai Petrovici. At the time of writing, it is being prepared for publication.

A central question in neuroscience is how the brain is capable of learning and building memories. This question concerns several sub-disciplines of neuroscience and it spans over orders of magnitude both in time and space: from molecular models of synaptic plasticity to changes in behavior over decades. Despite the myriads of proposed models and experimental findings (compare to section 2.2.3), it remains an open issue what the basic coding scheme(s) of the brain is and which learning rule(s) it realizes.

The recent success of deep learning [LeCun et al., 2015] put the question whether deep learning is realized in the mammalian brain back into the focus. Particularly interesting is deep reinforcement learning due to its success in machine learning: it has reached often super-human performance in playing board- and video-games [Mnih et al., 2013, Silver et al., 2017, Vinyals et al., 2019]. This success is based on three key components: the backpropagation algorithm [Rumelhart et al., 1986], the availability of large labeled datasets and the availability of cheap and powerful GPUs. Here, we focus on mechanistic modeling of the backpropagation algorithm and disregard the two other factors.

For a long time, backpropagation in the brain had been considered impossible [Richards et al., 2019]. It seemed implausible that an error signal is propagated backwards over several layers of neurons, tailored individually for each synapse and still obeying biological constraints like locality of interactions and the heterogeneity of the neuro-synaptic parameters. In the last couple of years, several studies either relaxed implausible assumptions [Lillicrap et al., 2016] or even suggested models for the backpropagation mechanism [O'Reilly, 1996, Xie and Seung, 2003, Roelfsema and Ooyen, 2005, Rombouts et al., 2015, Scellier and Bengio, 2017, Whittington and Bogacz, 2017, Amit, 2019, Mesnard et al., 2019, Marblestone et al., 2016, Pozzi et al., 2018, Whittington and Bogacz, 2019, Richards et al., 2019]. A common feature of all these works is that they usually consider supervised and unsupervised learning only, although reinforcement learning is clearly present in the brain [Niv, 2009]. Deep reinforcement learning is only considered in a few of these studies [Rombouts et al., 2015, Pozzi et al., 2018], but they lack other biological constraints such as time-continuous dynamics.

A large amount of literature has been published about models of reinforcement learning in the brain, see for example the review from Niv [2009]. However these publications are more focused on the biological aspects of reinforcement learning and less on the learning capabilities of the underlying model; and often they only consider shallow learning architectures, for example in Farries and Fairhall [2007], Izhikevich [2007a], Frémaux et al. [2010, 2013], Deperrois et al. [2019]. This restriction to shallow learning limits the capabilities of these models in terms of task complexity, but animals and humans can clearly solve more complex tasks than these shallow models allow.

In this project, we extend the model of deep supervised learning in a time-continuous framework based on the principle of least action [Senn et al., in preparation, Dold, 2020] to include reinforcement learning. In their work, Senn et al. [in preparation] derive neural dynamics and plasticity rules from first principles and present a framework of time-continuous deep learning using only local interactions and plasticity. The authors propose a model where stereotypical microcircuits and a predictive firing mechanism of the neurons enable learning via backpropagation at any time without separate phases. We amended the model with a lateral interaction among the neurons representing the actions and with a global neuromodulator based on the reward-prediction error. The lateral interaction is closely related to winner-takes-all (WTA) circuits, while the reward-prediction error measures the deviation of the received reward from the expectation [Schultz, 2016, Sutton and Barto, 2018]. We show that the proposed model approximates policy-gradient learning and, by that, maximizes the expected reward.

The proposed model is tested on a reduced version of the popular MNIST dataset [LeCun et al., 1998]. Furthermore, we verify the robustness of the model against both fixed and random temporal reward delays as well as against fixed-pattern noise (heterogeneous parameters) in the lateral circuit. Finally, we briefly sketch and test two alternative forms of reward maximizing interactions in the action layer.

Our work contributes to the pursuit for mechanistic models of biologically plausible deep reinforcement learning. This model could be the basis of more elaborate biological reinforcement learning models, for example based on actor-critic architectures; or it could inspire experiments to explore hallmarks of deep reinforcement learning in the brain.

5.1 Materials and Methods

The presented work extends and builds on the framework of supervised learning in the principle of least action model [Senn et al., in preparation, Dold, 2020]. Here, we give a summary of the model, so that the connection between the model and its extension becomes apparent. Moreover, we describe policy-gradient in a deep neural network, which we will use as a standard comparison for the presented learning rules.

5.1.1 Deep supervised learning in the principle of least action framework

The content of this section is in preparation for publication in Senn et al. [in preparation] and in the PhD thesis of Dominik Dold [Dold, 2020]. In this section, we give a summary of the theory, because the reinforcement learning model in this chapter is a direct extension of it.

The principle of least action framework aims to provide an energy-based description of time-continuous neural dynamics and the synaptic plasticity rules based on first principles. The neural dynamics result from the energy function via variational calculus inspired by the Lagrange formalism found in physics (see for example Landau and Lifshitz [2013]). The plasticity rule shapes the energy landscape in order to minimize the deviation to a targeted neuronal behavior. The resulting time-continuous neuro-synaptic dynamics enable learning at any time while approximating backpropagation if used for a layered network. Additionally, the resulting dynamics can be interpreted as cortical microcircuits, giving it a mechanistic model interpretation.

In the following, we describe the theory for a layered network both to emphasize the backpropagation feature and because we will extend it in a layered network setup. The framework can be extended to arbitrary network architectures [Senn et al., in preparation].

Energy, prediction error and cost

The total energy L of the system is composed of two terms: a prediction error E and a cost function C:

$$L = E + \beta C = \sum_{i=1}^{N} \frac{1}{2} \|u_i - W_i \bar{r}_{i-1}\|^2 + \beta \frac{1}{2} \left\| u_N - u_N^{(\mathrm{trg})} \right\|^2 \quad , \tag{5.1}$$

where the sum goes over the N layers, u_i is the vector of membrane potentials of the neurons in the i-th layer, W_i is the synaptic matrix projecting from layer $i-1$ to layer i, $\bar{r}_{i-1}$ is the low-pass filtered activation of the neurons in layer $i-1$ and $u_N^{(\mathrm{trg})}$ is target membrane potential of the neurons in the last (output) layer (figure 5.1 A). The $\|\cdot\|$ denotes the standard Euclidean norm of a vector. The term C is the *cost*: it measures the difference between the membrane potential of the output neurons and the desired target potential $u_N^{(\mathrm{trg})}$. The cost function C corresponds to the loss function in supervised learning (for example section 2.1.1), $u_N^{(\mathrm{trg})}$ could be the desired label of a classified image using one-hot coding. The term E is called the *prediction error*, because it represents the difference between the realized somatic membrane potential u_i and the corresponding dendritic prediction $W_i\bar{r}_{i-1}$. In this sense the framework is based on predictive coding, meaning that each neuron aims at reducing this dendritic prediction error. The β factor weights the cost relative to the prediction error; and, as we will see, it weights the amplitude of the error backpropagation direction compared to the inference direction. In the following, we show that, under certain prerequisites, minimizing the prediction error E corresponds to minimizing the cost C. The calculations are shown in detail in section 5.1.2.

The neural dynamics

To derive the neural dynamics we introduce the concept of future discounted voltage:

$$\tilde{u}(t) = \frac{1}{\tau}\int_t^\infty u(\hat{t})\exp\left(-\frac{\hat{t}-t}{\tau}\right)\mathrm{d}\hat{t} \quad , \tag{5.2}$$

where τ is the membrane-time constant of the neuron, which will become immediately clear after the derivation of the neural dynamics. The future discounted voltage $\tilde{u}$ will act as generalized neural coordinates: we use $(\tilde{u}, \dot{\tilde{u}})$ as the canonic variables to execute the variational calculus, implicitly using the total energy in terms of $L = L(\tilde{u}, \dot{\tilde{u}})$. Using equation (5.2), the membrane potential can be reconstructed from $\tilde{u}$ with a formula resembling a look-back (section 5.1.2)

$$u = \tilde{u} - \tau\dot{\tilde{u}} \quad . \tag{5.3}$$

We require the membrane potential in terms of $\tilde{u}$ to follow a trajectory stationary to the action $A = \int L(\tilde{u}, \dot{\tilde{u}})\mathrm{d}t$, or $\delta A = 0$. This leads to the Euler-Lagrange equations:

$$\frac{\partial L}{\partial \tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\tilde{u}}} = 0 \quad . \tag{5.4}$$

From equation (5.3) we can derive the relation between the partial differentials,

$$\begin{aligned} \frac{\partial u}{\partial \tilde{u}} &= 1 \quad , \\ \frac{\partial u}{\partial \dot{\tilde{u}}} &= -\tau \quad , \end{aligned} \tag{5.5}$$

yielding the dynamic equations for the neurons:

$$\left(1 + \tau\frac{\mathrm{d}}{\mathrm{d}t}\right)\frac{\partial L}{\partial u} = 0 \quad , \tag{5.6}$$

or explicitly:

$$\begin{aligned} \tau\dot{u}_i &= W_i r_{i-1} - u_i + e_i \quad , \\ r_i = \bar{r}_i + \tau\dot{\bar{r}}_i \; ; &\quad e_i = \bar{e}_i + \tau\dot{\bar{e}}_i \quad , \\ \bar{e}_i &= \bar{r}_i \odot W_{i+1}^T\left(u_{i+1} - W_{i+1}\bar{r}_i\right) \quad , \\ \bar{e}_N &= u_N^{\mathrm{trg}} - u_N \quad . \end{aligned} \tag{5.7}$$

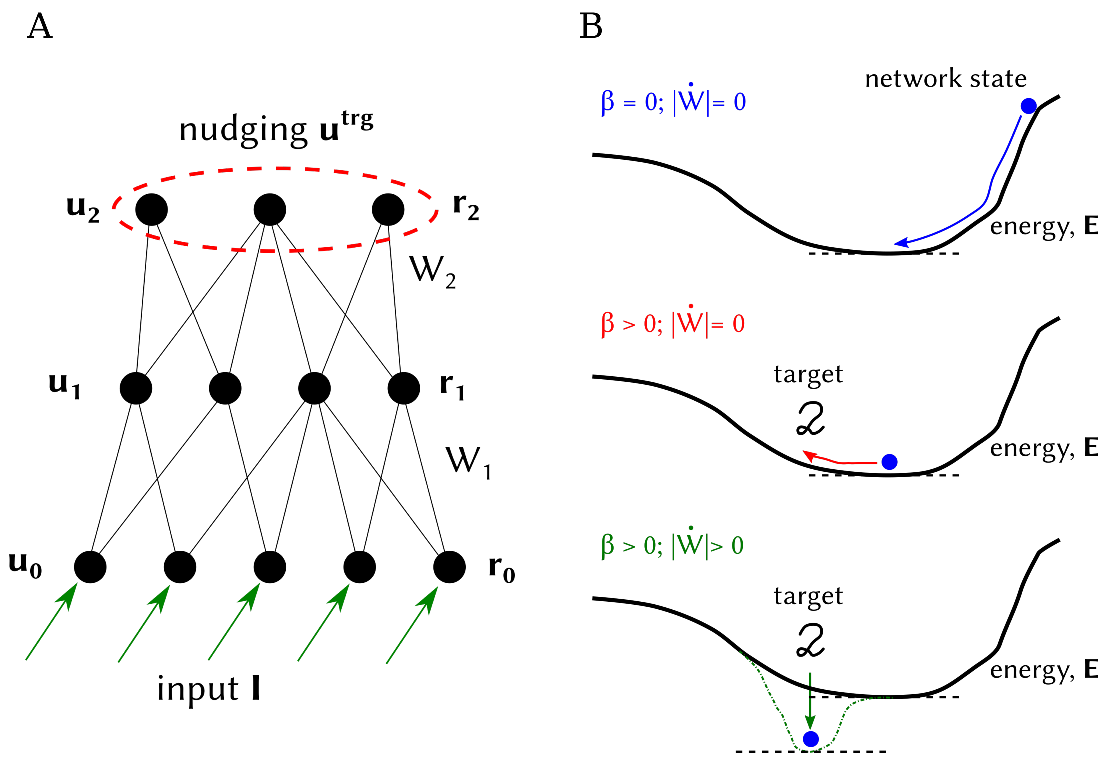


Figure 5.1: Sketch of the setup and dynamics of the principle of least action framework. (A) Sketch of the setup on the example of a layered feed-forward network. The setup is similar to feed-forward networks of artificial neurons (section 2.1.1), but the neurons have internal rate-based dynamics and a subset of the neurons (output, red dashed line) obtains nudging from a target voltage. The cost function C is defined over these output neurons. **(B)** Intuitive mechanism of the framework. *Top:* Without learning or nudging the network dynamics drive the network towards the self-predicting state where the prediction error E is minimized. *Middle:* The nudging pulls the network towards the desired target state, where the membrane potential of the output neurons u_N is closer to the desired target $u_N^{(\mathrm{trg})}$. In this case the prediction error E is not minimal anymore, but the total energy L is still minimized. *Bottom:* The learning rule adjusts the energy landscape of E such that the output neurons get closer to their target value. The minimum of E and the minimum of C will fall closer to each other due to the plasticity mechanism. Figure adapted from Dold [2020].

Here, $\odot$ is the element-wise product of two vectors. Note, that the neurons do not fire with $\bar{r}_i$ but with the instantaneous firing rate $r_i = \bar{r}_i + \tau \bar{r}'_i \odot \dot{u}_i$, which not only depends on the membrane potential u_i but also on the first time derivative $\dot{u}_i$. In a simplified picture, we say that the neuron fires with a predictive rate by using the first time derivative of the membrane potential to modulate their firing. To distinguish the two quantities we call $\bar{r}_i$ the steady-state firing rate, which does not depend on $\dot{u}_i$. $\bar{r}_i(u_i)$ is the firing rate of the neuron if the membrane potential u_i is constant in time. Henceforth, we refer to $\bar{r}_i$ as the activation function of the neuron. We call r_i the instantaneous firing rate, that is the firing rate of the neuron if it is exposed to a time-varying stimulus.

Note that the operator $(1 + \tau \frac{\mathrm{d}}{\mathrm{d}t})$ cancels the low-pass filtering operation $\bar{x}(t) = \frac{1}{\tau} \int_{-\infty}^{t} x(\hat{t}) \exp\left(-\frac{t-\hat{t}}{\tau}\right) \mathrm{d}\hat{t}$, that is $(1 + \tau \frac{\mathrm{d}}{\mathrm{d}t})\bar{x} = x$ (section 5.1.2). This is the rationale behind the mechanism how the neurons can at the same time integrate up the input current I and still keep the feed-forward information and feed-back error in phase. The leaky integrator behavior yields a low-pass filtering $u \propto \bar{I}$ of the input I, but the look-ahead firing cancels the low-pass filter. Without the look-ahead mechanism, each layer in the network would introduce a τ time-lag due to leaky integrator property. The backpropagated error would lag behind the feed-forward information. To create the error signal at a given synapse, the input signal first has to propagate to the output neurons and then back to the synapse of interest. At each layer, the leaky integrator dynamics would add another τ delay to the signal. Hence, the bottom-up signal and the top-down error signal would not correspond to each other, and learning would break down.

The synaptic plasticity

We design the plasticity rule with the aim to minimize the cost C. For that we use the fact that for small β local changes in the synaptic strength that minimize E will in turn also minimize the cost C (details in section 5.1.2). The mechinism is sketched in figure 5.1 B. Hence, we define the plasticity rule as

$$\dot{W}_i \propto -\frac{\mathrm{d}L}{\mathrm{d}W_i} = -\sum_{l=0}^{N} \frac{\partial L}{\partial u_l} \frac{\mathrm{d}u_l}{\mathrm{d}W_i} - \frac{\partial L}{\partial W_i} = -\frac{\partial L}{\partial W_i} = (u_i - W_i \bar{r}_{i-1}) \bar{r}_{i-1}^T = \bar{e}_i \bar{r}_{i-1}^T \quad . \tag{5.8}$$

In the derivation we can set $\frac{\partial L}{\partial u_l} = 0$ because according to neural dynamics (equation (5.6)) it decays exponentially to zero with the time-constant τ. Several τ time after initialization the term can be neglected. The learning rule itself is now a local plasticity rule with the mismatch $(u_i - W_i \bar{r}_{i-1})$ between the somatic voltage and the dendritic prediction on the post-synaptic side and with the pre-synaptic firing rate $\bar{r}_{i-1}$. Equation (5.8) implements gradient descent on the cost function and can be interpreted as a predictive dendritic learning rule in the sense of Urbanczik and Senn [2014].

In the case of weak nudging, the plasticity rule implements a gradient descent on the cost function (detailed calculation in section 5.1.2):

$$
\begin{aligned}
-\frac{\mathrm{d}}{\mathrm{d}W_i}C = -\frac{\mathrm{d}}{\mathrm{d}W_i}\frac{\partial}{\partial\beta}L\bigg|_{\beta=0} &= -\frac{\mathrm{d}}{\mathrm{d}\beta}\frac{\partial}{\partial W_i}L\bigg|_{\beta=0} = \\
= \lim_{\beta\to 0}(u_i - W_i\bar{r}_{i-1})\bar{r}_{i-1}^T &\approx \frac{1}{\beta}(u_i - W_i\bar{r}_{i-1})\bar{r}_{i-1}^T \quad ,
\end{aligned}
\tag{5.9}
$$

where the membrane potential and the low-passed firing rate are meant in the case when nudging is turned on ($\beta > 0$). Intuitively, the plasticity forms the energy landscape E in a way that output membrane potentials will get closer to their target values even without additional nudging, hence the cost function C is reduced.

The plasticity rule can be related directly to the well-known backpropagation algorithm (section 2.1.1, Rumelhart et al. [1986]). We use the fact that far away from the initialization, we can say that the partial derivative of L according to u_i vanishes. And hence writing out $\frac{\partial L}{\partial u_i} = 0$, we obtain:

$$
\begin{aligned}
u_i &= W_i\bar{r}_{i-1} + \bar{e}_i \quad \text{, with} \\
\bar{e}_i = \bar{r}_i' \odot W_{i+1}^T\left(u_{i+1} - W_{i+1}\bar{r}_i\right) &= \bar{r}_i' \odot \left[W_{i+1}^T\bar{e}_{i+1}\right] \quad \text{, and} \\
\bar{e}_N &= u_N^{\text{trg}} - u_N \quad .
\end{aligned}
\tag{5.10}
$$

Here we made use of $\frac{\partial L}{\partial u_i} = 0$ implying $\bar{e}_{i+1} = u_{i+1} - W_{i+1}\bar{r}_i$ to establish the recursive formula for the errors. In the last (output) layer of the network, the nudging gives rise to the error $\bar{e}_N = u_N^{\text{trg}} - u_N$, which propagates through the network back to the first layer via the recursive formula (equation (5.10)). Because the imposed neural dynamics (equation (5.7)) keep the network at $\frac{\partial L}{\partial u_i} = 0$, learning along the gradient of backpropagation is possible at any time. This is true as long as the input to the network is smooth enough, such that the look-ahead firing is informative about the future development of the system. This requirement is violated in case of non-differentiable input. Intuitively, the look-ahead firing $r_i = \bar{r}_i + \tau\bar{r}_i' \odot \dot{u}_i$ uses the current derivative of the membrane potential to predict its future course. For non-differentiable input, the time-derivative is not informative about the future and therefore the predictive firing property breaks down.

Interpretation as cortical microcircuits

The neuro-synaptic dynamic equations (equation (5.7)) give rise to a physiological interpretation with excitatory pyramidal neurons and inter-neurons forming stereotypical microcircuits (figure 5.2 and equation (5.8)). In a feed-forward network (can also be extended to arbitrary architectures) the pyramidal neurons form the inference pathway similar to the neurons in an abstract neural network (section 2.1.1), but the pyramidal neurons also show temporal leaky integrator dynamics (section 2.2.2).

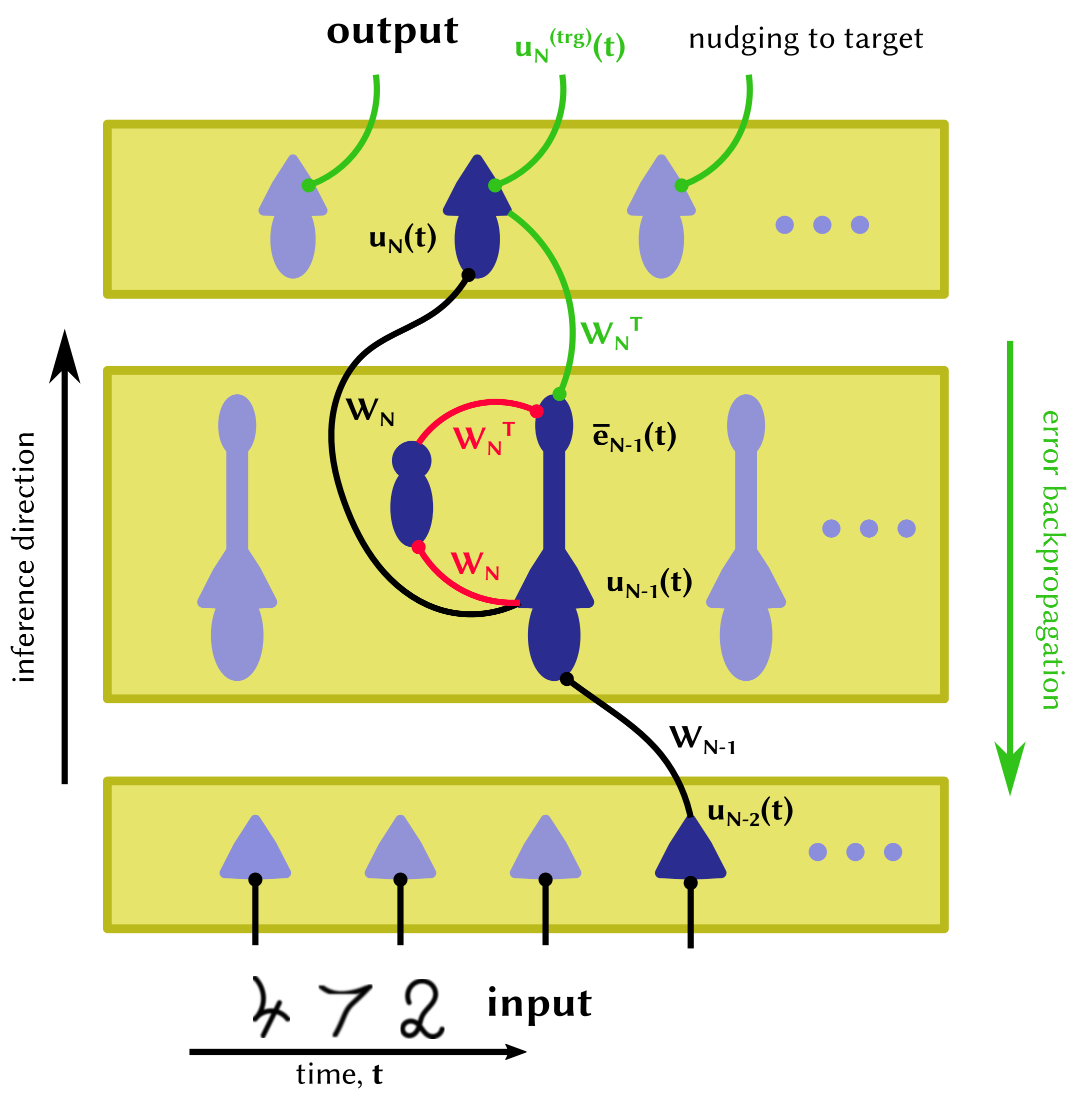


Figure 5.2: Physiological interpretation of the model equations. In the feed-forward direction (from bottom to top), the pyramidal neurons act similarly to abstract neurons in a conventional neural network (section 2.1.1) but the pyramidal neurons additionally feature leaky integrator dynamics. In the error backpropagation direction (from top to bottom), the output neurons are nudged towards the target value, hence they deviate from the activity predicted from below. In each hidden layer, stereotypical microcircuits of inter-neurons and pyramidal neurons project back the prediction error. Importantly, the network is recurrent in its connection due to the feedback connections, but feed-forward in its function. Figure adapted from Dold [2020].

In the error-backpropagation path the error-vector is generated in the output neurons via nudging towards the target value $\beta\left(u_N^{(\mathrm{trg})} - u_N\right)$. Note that this nudging is both weak and conductance-based. We assumed a weak nudging by requiring a small β in equation (5.9). This implies that the dynamics of the network is mainly driven by the bottom-up input, the backpropagation of the error only modulates this activity slightly. The conductive nudging means, that the nudging current linearly depends on the distance between realized and target membrane potential, resembling conductance based synaptic connections (section 2.2.2). The advantage of conductive nudging is that the supervisor turns itself automatically off if the output is correct, hence it does not disturb an already learned correct output.

The generated error is communicated back to the deeper layers via stereotypical microcircuits. Each microcircuit consists of a pyramidal neuron and an inter-neuron. The naming stems from their biological resemblance (see the following subsection). The inter-neuron shows an activity that the pyramidal neuron would show if it did not experience any top-down error nudging. Both the inter-neuron and the pyramidal neuron project to the apical dendrite of the pyramidal neuron in the layer below, but the inter-neuron does so with a negative sign. Hence, they compute the error $\bar{e}_i$ at the apical dendrite of the pyramidal neuron that nudges the somatic voltage. In the error term we can identify the top-down pathway and the inter-neuron circuit as

$$\bar{e}_i = \underbrace{\bar{r}'_i}_{\text{modulator}} \odot \left(\underbrace{W_{i+1}^T u_{i+1}}_{\text{top-down}} - \underbrace{W_{i+1}^T W_{i+1} \bar{r}_i}_{\text{via inter-neuron}} \right) . \tag{5.11}$$

Senn et al. [in preparation] further show that the requirement of weight sharing among the physically distinct synapses can be relaxed by using the arguments from feedback alignment [Lillicrap et al., 2016] and by learning the lateral weights. In this project, we stick to a formulation with weight sharing because the simulation of the model is computationally demanding even with shared weights, and we focus on the aspects of reinforcement learning.

Biological inspiration and application to the MNIST dataset

Apart from the existence of pyramidal- and inter-neurons, the presented framework relies on three main biological inspirations (and postdictions).

In the principle of least action framework, the neurons fire with $r = \bar{r} + \tau\dot{\bar{r}}$ instead of the traditionally assumed activation function $\bar{r}$ that only depends on the instantaneous membrane potential (figure 5.3 A-B). Köndgen et al. [2008] have shown, in *in vitro* experiments, that neurons can tract sinusoidal current input up to 200 Hz surprisingly fast. The firing rate is sometimes even phase-advanced compared to the current input in spite of the leaky integrator nature of neurons (figure 5.3 C). Senn et al. [in preparation] argue that similar look-ahead dynamics can be derived in the Hodgkin-Huxley model [Hodgkin and Huxley, 1952].

Second, the dynamic equations (equation (5.7)) and their microcircuit interpretation (figure 5.2) require that not the firing rate but the membrane potential is transmitted from the pyramidal neuron in layer $l + 1$ to the apical dendrite of the pyramidal neuron in layer l. Similarly, the membrane potential is transmitted from the inter-neurons to the apical dendrite of the pyramidal neurons. This can be achieved by assuming linear activation functions, but this is not favorable because it poses a strong restriction on the realizations. Alternatively, we can achieve it by using the filtering property of short-term depression. Inspired by the work of Pfister et al. [2010], Senn et al. [in preparation] show that short-term depression can filter the pre-synaptic membrane voltage, and hence a synapse with an appropriate short-term depression can transmit the pre-synaptic membrane potential to the post-synaptic neuron. Essentially, the non-linearity of the activation function and the non-linearity introduced by the short-term plasticity cancel out each other. If we assume that filtering via short-term depression is present on the apical dendrite of the pyramidal neurons, we obtain the necessary membrane potential transfer without restrictions on the activation function.

Third, the two types of neurons in the microcircuits received their names due to their similarity to neuron types found in the mammalian brain. Pyramidal neurons are the most abundant type of neuron in the neocortex [Spruston, 2008]. They are usually excitatory in their effect, and they show a typical morphology with a pyramid-shaped soma, and large tree-like apical and basal dendrite. Pyramidal neurons receive input from large distances, also from outside of the neocortex. Spike-generation takes place at the soma, but electric effects of action-potentials propagate back to the dendrites as well. This so-called backpropagating action potentials justify the idea that both somatic and dendritic quantities could be treated as local information for plasticity [Urbanczik and Senn, 2014].

Inter-neurons constitute a more diverse family of neurons [Markram et al., 2004]. In spite of their variability, most inter-neurons are inhibitory and they preferably form local intra-cortical synaptic connections, which makes them good candidates for forming the stereotypical microcircuits.

Senn et al. [in preparation] apply the theory to the MNIST dataset [LeCun et al., 1998] and show that learning is indeed possible over several hidden layers both with the standard model (figure 5.3 D) and with the relaxed weight requirements on the weight sharing. The learning speed is on par with standard backpropagation in terms of improvement per iteration. Note that equating iterations between a time-continuous system and a discrete system necessarily introduces simplifications, and is therefore only approximative.

Finally, note that the presented network is feed-forward from the point of view of its function but it is recurrent in its structure. This paradox is resolved by the following reasoning. If we consider the synaptic connections among the neurons as a directed graph[1], the network is recurrent. There are cycles between the pyramidal neurons of the consecutive layers and between the pyramidal neurons and the corresponding inter-neurons. This is how a researcher from neuroscience

[1]For a textbook on graph theory see Grimaldi [2003].

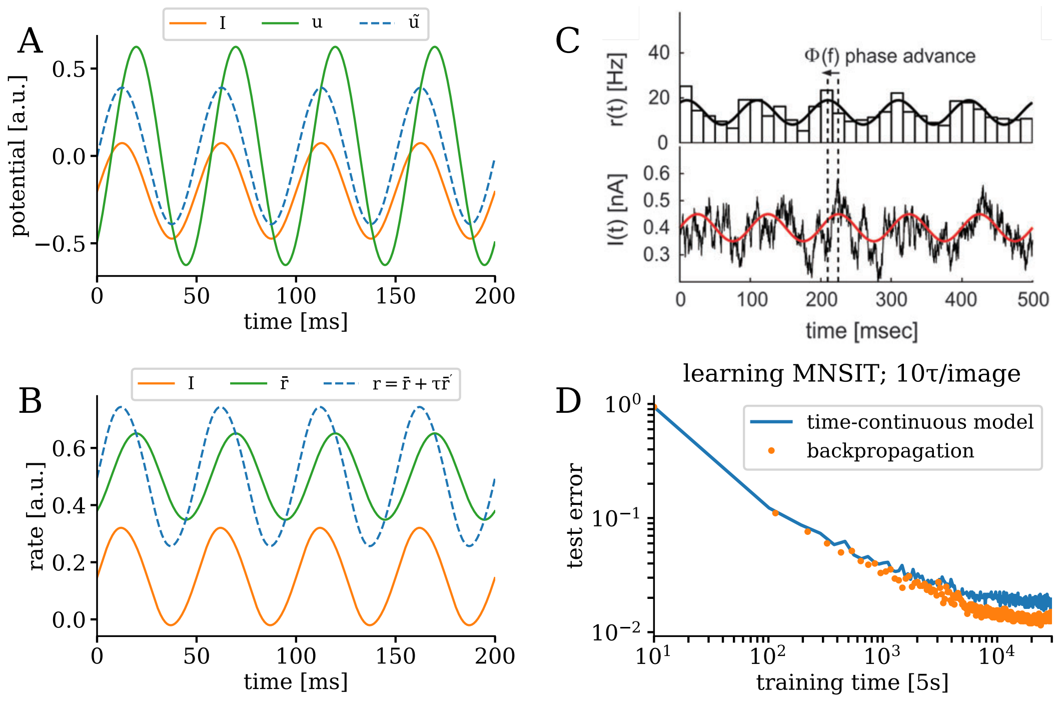


Figure 5.3: Motivation and application of the principle of least action framework. (A) Upon a sinusoidal input, the realized membrane potential u follows the input I with a τ lag. The future discounted voltage is phase-advanced compared to the membrane potential $\tilde{u}$ and is in phase with the modulating input. Figure adapted from Dold [2020]. **(B)** Similarly, the low-passed membrane potential $\bar{r}$ lags behind the input I. By the look-ahead firing mechanism, the neuron fires with r in phase with the input. Figure adapted from Dold [2020]. **(C)** Köndgen et al. [2008] have found experimental evidence that neurons can follow closely the input with their firing pattern over a broad range of frequencies. In some cases, they even show phase-advanced firing compared to the input. Figure taken from Köndgen et al. [2008]. **(D)** Supervised learning with the time-continuous model is on par with learning with backpropagation (data from Dold [2020]). Here, a membrane time-constant $\tau = 10\,\text{ms}$ was used, and each image was shown for 10τ. Figure adapted from Dold [2020].

would think about the network. However, the information processing goes only into the direction from input to output without cycles. The connections from the inter-neurons to the pyramidal neurons and from the higher layers to lower layers propagate only the errors by design. Furthermore, the magnitude of the backpropagated signal is by design β-times smaller than the forward propagated signal. The cyclic connections could, in principle, lead to significant changes as in chaotic dynamical systems, but we did not observe such behavior in the simulations. Hence, the network processes the input in a feed-forward manner similarly to feed-forward ANNs, and they do not use long-term memory as, for example, recurrent neural networks in machine learning [Hochreiter and Schmidhuber, 1997]. Finally, this observation only holds for the feed-forward example in this thesis.

5.1.2 Detailed calculations of the principle of least action framework

Properties of the future discounted voltage

In the following, we give a definition and main properties of the future discounted voltage from equations (5.3) and (5.5). Although, the calculations are not particularly complex, the rigorous separation of the steps gives a better overview of the model.

Definition 1 (Future discounted voltage) *Given a membrane potential $u : \mathbb{R} \to \mathbb{R}$ with $t \mapsto u(t)$, the future discounted voltage $\tilde{u}(t)$ with the time-constant τ is defined as the look-ahead of $u(t)$:*

$$\tilde{u}(t) := \frac{1}{\tau} \int_t^\infty u(\hat{t}) \exp\left(-\frac{\hat{t}-t}{\tau}\right) \mathrm{d}\hat{t} \quad . \tag{5.12}$$

First important property of the future discounted voltage is the look-back relation to the membrane potential.

Property 1 (Look-back property) *The membrane potential u can be reconstructed via a look-back from the future discounted voltage,*

$$u = \tilde{u} - \tau \dot{\tilde{u}} \quad . \tag{5.13}$$

Proof. First we observe from the definition, that

$$\begin{aligned} \dot{\tilde{u}} = \frac{\mathrm{d}}{\mathrm{d}t} \left[\exp\left(\frac{t}{\tau}\right) \frac{1}{\tau} \int_t^\infty u(\hat{t}) \exp\left(-\frac{\hat{t}}{\tau}\right) \mathrm{d}\hat{t} \right] = \\ = \frac{1}{\tau} \exp\left(\frac{t}{\tau}\right) \frac{1}{\tau} \int_t^\infty u(\hat{t}) \exp\left(-\frac{\hat{t}}{\tau}\right) \mathrm{d}\hat{t} - \exp\left(\frac{t}{\tau}\right) \frac{1}{\tau} u(t) \exp\left(-\frac{t}{\tau}\right) = \\ = \frac{1}{\tau}(\tilde{u} - u) \quad . \end{aligned} \tag{5.14}$$

Using the result we obtain:

$$\tilde{u} - \tau\dot{\tilde{u}} = \tilde{u} - \tau\frac{1}{\tau}(\tilde{u} - u) = u \quad . \tag{5.15}$$

□

The second property summarizes the derivatives of u according to $\tilde{u}$ and $\dot{\tilde{u}}$ (equation (5.5)), which allowed us to formulate the Euler-Lagrange equations in terms of the variable u.

Property 2 (Partial derivative relationship) *Between u and $\tilde{u}$ we have the following derivatives:*

$$\begin{aligned} \frac{\partial u}{\partial \tilde{u}} &= 1 \quad , \\ \frac{\partial u}{\partial \dot{\tilde{u}}} &= -\tau \quad . \end{aligned} \tag{5.16}$$

Proof. Using $u(\tilde{u}, \dot{\tilde{u}}) = \tilde{u} - \tau\dot{\tilde{u}}$ (property 1) first we can immediately take the partial $\frac{\partial u}{\partial \tilde{u}}$:

$$\frac{\partial u}{\partial \tilde{u}} = 1 \quad . \tag{5.17}$$

Similarly by taking $\frac{\partial u}{\partial \dot{\tilde{u}}}$:

$$\frac{\partial u}{\partial \dot{\tilde{u}}} = -\tau \quad . \tag{5.18}$$

□

Inverting the low-pass filtering

For the sake of clarity we define the already introduced low-pass filtering operation and the inverse of it.

Definition 2 (Low-pass filer) *Given a function $x(t) : \mathbb{R} \to \mathbb{R}, t \mapsto x(t)$, the low-pass filter $\bar{x}(t)$ with the time-constant τ is defined as:*

$$\bar{x}(t) := \frac{1}{\tau}\int_{-\infty}^{t} x(\hat{t}) \exp\left(-\frac{t - \hat{t}}{\tau}\right) \mathrm{d}\hat{t} \quad . \tag{5.19}$$

Definition 3 (The look-ahead operator) *The look-ahead operation with time-constant τ for a function $x(t) : \mathbb{R} \to \mathbb{R}, t \mapsto x(t)$ is defined as:*

$$(1 + \tau\frac{\mathrm{d}}{\mathrm{d}t})x(t) \tag{5.20}$$

Lemma 1 (Inverting the low-pass filter) *The low-pass filter operation and the look-ahead operation are inverse of each other.*

Proof.

I. First, we show $(1+\tau\frac{\mathrm{d}}{\mathrm{d}t})\bar{x} = x$
First we look at:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\bar{x}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{1}{\tau}\int_{-\infty}^{t} x(\hat{t})\exp\left(-\frac{t-\hat{t}}{\tau}\right)\mathrm{d}\hat{t}\right] = \\ = -\frac{1}{\tau}\exp\left(-\frac{t}{\tau}\right)\frac{1}{\tau}\int_{-\infty}^{t} x(\hat{t})\exp\left(\frac{\hat{t}}{\tau}\right)\mathrm{d}\hat{t} + \exp\left(-\frac{t}{\tau}\right)\frac{1}{\tau}x(t)\exp\left(\frac{t}{\tau}\right) = \\ = -\frac{1}{\tau}\bar{x}(t) + \frac{1}{\tau}x(t) \quad .\end{aligned} \tag{5.21}$$

Using this, we calculate:

$$(1+\tau\frac{\mathrm{d}}{\mathrm{d}t})\bar{x}(t) = \bar{x}(t) + \tau\frac{\mathrm{d}}{\mathrm{d}t}\bar{x}(t) = \bar{x}(t) - \tau\left(-\frac{1}{\tau}\bar{x}(t) + \frac{1}{\tau}x(t)\right) = x(t) \quad . \tag{5.22}$$

II. Second, we show $\overline{(1+\tau\frac{\mathrm{d}}{\mathrm{d}t})x} = x$
First, by integration by parts we calculate:

$$\begin{aligned}\overline{\tau\frac{\mathrm{d}}{\mathrm{d}t}x} = \tau\frac{1}{\tau}\int_{-\infty}^{t}\left(\left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{\hat{t}}\right)\exp\left(-\frac{t-\hat{t}}{\tau}\right)\mathrm{d}\hat{t} = \exp\left(-\frac{t}{\tau}\right)\int_{-\infty}^{t}\left(\left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{\hat{t}}\right)\exp\left(\frac{\hat{t}}{\tau}\right)\mathrm{d}\hat{t} = \\ = \exp\left(-\frac{t}{\tau}\right)\left[x(\hat{t})\exp\left(\frac{\hat{t}}{\tau}\right)\Big|_{\hat{t}=-\infty}^{\hat{t}=0} - \int_{-\infty}^{t} x(\hat{t})\exp\left(\frac{\hat{t}}{\tau}\right)\frac{1}{\tau}\right] = \\ = x(t) - \bar{x}(t) \quad .\end{aligned} \tag{5.23}$$

Using this, we calculate:

$$\overline{(1+\tau\frac{\mathrm{d}}{\mathrm{d}t})x} = \bar{x} + x - \bar{x} = x \quad . \tag{5.24}$$

□

Properties of the neural and synaptic dynamics

In the following we give the calculations of the neural and synaptic dynamics.

Definition 4 (Neural dynamics) *Given the total energy $L = E + C$, the neural dynamics is postulated to follow the Euler-Lagrange equation with the variables $(\tilde{u}, \dot{\tilde{u}})$:*

$$\frac{\partial L}{\partial \tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\tilde{u}}} = 0 \quad . \tag{5.25}$$

Lemma 2 (Neural dynamics) *Given the total energy $L = E + C$ the neural dynamics is by governed by the equation:*

$$\left(1+\tau\frac{\mathrm{d}}{\mathrm{d}t}\right)\frac{\partial L}{\partial u} = 0 \quad . \tag{5.26}$$

Proof. The proof results by substituting the operator relations from property 2 in definition 4:

$$0 = \frac{\partial L}{\partial \tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\tilde{u}}} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial \tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial u}\frac{\partial u}{\partial \dot{\tilde{u}}} = \frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial u}(-\tau) = \left(1 + \tau\frac{\mathrm{d}}{\mathrm{d}t}\right)\frac{\partial L}{\partial u} \quad . \tag{5.27}$$

□

First, we show that the partial derivative $\frac{\partial L}{\partial u}$ decays exponentially with the ongoing dynamics.

Lemma 3 (Steady-state neural dynamics) *The partial derivative $\frac{\partial L}{\partial u}$ decays exponentially, and disappears for $t \to \infty$ as:*

$$\frac{\partial L}{\partial u} = \left.\frac{\partial L}{\partial u}\right|_{t_0} \exp\left(-\frac{t - t_0}{\tau}\right) \tag{5.28}$$

with $\left.\frac{\partial L}{\partial u}\right|_{t_0}$ the initial value at t_0.

Proof. Proof is given by the solution of the dynamic equation defined in lemma 2.

□

Remark 1. The steady-state neural dynamics are made possible by the look-ahead firing property of the neurons. With the look-ahead, the neurons are able to follow the input as long as the input is differentiable and does not contain any discontinuities (jumps), hence the system stays in its steady-state while at the same time it is controlled by the dynamic equations. If the input contains jumps, then the network drops out of the steady-state dynamics but it relaxes back to it with the time-constant of τ. Loosely speaking: The aim of the introduced dynamics is to do away with the leaky integrator dynamics and make the system closely follow the input.

Lemma 4 (Interchangeability of partial and total derivatives with respect to W) *Given the dynamics in lemma 2, far away from the initialization the total and the partial derivatives according to W can be exchanged:*

$$\frac{\mathrm{d}L}{\mathrm{d}W} = \frac{\partial L}{\partial W} \quad . \tag{5.29}$$

Proof. For this proof we will proceed similarly as in the case of the derivation of the Euler-Lagrange equations. We look at a time interval $[t_1; t_2]$ that is sufficiently large, that is $t_2 - t_1 \gg \tau$. We consider an arbitrary variation $\delta W(t)$ such that it disappears at the edges of the interval, $\delta W(t_1) = \delta W(t_2) = 0$. The variation $\delta W(t)$ will also lead to the variations $\delta \tilde{u}(t)$ and $\delta \dot{\tilde{u}}(t)$ of the solutions of the Euler-Lagrange equation. Furthermore, we consider the effect of the infinitesimal variation on the action L with the variational parameter ϵ. At any given point in time we can express the variation with the variational parameter:

$$\frac{\mathrm{d}L}{\mathrm{d}W}\delta W = \frac{\mathrm{d}}{\mathrm{d}\epsilon}L(\tilde{u}+\epsilon\delta\tilde{u}, \dot{\tilde{u}}+\epsilon\delta\dot{\tilde{u}}, W+\epsilon\delta W)\bigg|_{\epsilon=0} = \frac{\partial L}{\partial\tilde{u}}\delta\tilde{u} + \frac{\partial L}{\partial\dot{\tilde{u}}}\delta\dot{\tilde{u}} + \frac{\partial L}{\partial W}\delta W \tag{5.30}$$

Using this we calculate:

$$\int_{t_1}^{t_2}\frac{\mathrm{d}L}{\mathrm{d}W}\delta W\mathrm{d}t = \int_{t_1}^{t_2}\frac{\partial L}{\partial\tilde{u}}\delta\tilde{u} + \frac{\partial L}{\partial\dot{\tilde{u}}}\delta\dot{\tilde{u}} + \frac{\partial L}{\partial W}\delta W\mathrm{d}t \quad . \tag{5.31}$$

Here, we use integration by parts:

$$\int_{t_1}^{t_2}\frac{\partial L}{\partial\dot{\tilde{u}}}\delta\dot{\tilde{u}}\mathrm{d}t = \frac{\partial L}{\partial\dot{\tilde{u}}}\delta\tilde{u}\bigg|_{t_1}^{t_2} - \int_{t_1}^{t_2}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial\dot{\tilde{u}}}\right)\delta\tilde{u}\mathrm{d}t \quad . \tag{5.32}$$

Plugging this into equation (5.31) we obtain:

$$\begin{aligned}\int_{t_1}^{t_2}\frac{\mathrm{d}L}{\mathrm{d}W}\delta W\mathrm{d}t &= \int_{t_1}^{t_2}\delta\tilde{u}\underbrace{\left(\frac{\partial L}{\partial\tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\tilde{u}}}\right)}_{=0}\mathrm{d}t - \frac{\partial L}{\partial\dot{\tilde{u}}}\delta\dot{\tilde{u}}\bigg|_{t_1}^{t_2} + \int_{t_1}^{t_2}\frac{\partial L}{\partial W}\delta W\mathrm{d}t = \\ &= -\frac{\partial L}{\partial\dot{\tilde{u}}}\delta\dot{\tilde{u}}\bigg|_{t_1}^{t_2} + \int_{t_1}^{t_2}\frac{\partial L}{\partial W}\delta W\mathrm{d}t \quad ,\end{aligned} \tag{5.33}$$

where we used that the system follows the dynamics $\frac{\partial L}{\partial\tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\tilde{u}}} = 0$. Rearranging the terms and using the operational relation (property 2) yields:

$$\int_{t_1}^{t_2}\left(\frac{\partial L}{\partial W} - \frac{\mathrm{d}L}{\mathrm{d}W}\right)\delta W\mathrm{d}t = -\tau\frac{\partial L}{\partial u}\delta\dot{\tilde{u}}\bigg|_{t_1}^{t_2} \quad . \tag{5.34}$$

On the right-hand side the term evaluated at t_1 disappears because the variation is zero $\delta\dot{\tilde{u}}(t_1) = 0$. At t_2 we know that $\frac{\partial L}{\partial u}\Big|_{t_2} \propto \exp\left(-\frac{t_2-t_1}{\tau}\right)$ due to lemma 3. Assuming a sufficiently large time interval and a sufficiently well-behaved variation δW leading to a sufficiently well-behaved (non-exploding) $\delta\tilde{u}$, we can neglect the term at t_2 as well, leading to:

$$\int_{t_1}^{t_2}\left(\frac{\partial L}{\partial W} - \frac{\mathrm{d}L}{\mathrm{d}W}\right)\delta W\mathrm{d}t = 0 \quad . \tag{5.35}$$

Finally, because of the (almost) arbitrary choice of δW, we conclude that the partial and total derivatives can be exchanged:

$$\frac{\partial L}{\partial W} = \frac{\mathrm{d}L}{\mathrm{d}W} \quad . \tag{5.36}$$

□

Remark 2. We note again, that the proof relies on the assumption of a well-behaved variation of $\delta W(t)$, which does not destroy the system by introducing exploding behavior.

Lemma 5 (Interchangeability of partial and total derivatives with respect to β) *Given the dynamics in lemma 2, far away from the initialization the total and the partial derivatives according to β can be exchanged:*

$$\frac{\mathrm{d}L}{\mathrm{d}\beta} = \frac{\partial L}{\partial \beta} \quad . \tag{5.37}$$

Proof. The proof goes exactly as the proof of lemma 4 but instead we consider the variation $\delta\beta(t)$. □

After all this preparation we can state the complete theorem.

Theorem 1 (Learning in the principle of least action framework) *Consider a total energy function L composed of a prediction error E and a cost function term C:*

$$L = E + \beta C = \sum_{i=1}^{N} \frac{1}{2} \left\| u_i - W_i \bar{r}_{i-1} \right\|^2 + \beta \frac{1}{2} \left\| u_N - u_N^{(\mathrm{trg})} \right\|^2 \tag{5.38}$$

with small $\beta \ll 1$, and formulated both with the membrane potential u and with the future discounted voltage $\tilde{u}(t) := \frac{1}{\tau} \int_t^{\infty} u(\hat{t}) \exp\left(-\frac{\hat{t}-t}{\tau}\right) \mathrm{d}\hat{t}$. Furthermore, consider the imposed neural dynamics that keep the action $A = \int L(\tilde{u}, \dot{\tilde{u}})\mathrm{d}t$ extremal,

$$\frac{\partial L}{\partial \tilde{u}} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L}{\partial \dot{\tilde{u}}} = 0 \quad . \tag{5.39}$$

Third, consider the postulated plasticity rule:

$$\dot{W}_i \propto -\frac{\partial E}{\partial W_i} \quad \forall i \in \{1, \ldots, N\} \quad . \tag{5.40}$$

Then, the plasticity rule performs gradient a descent on the cost function C.

Proof. The proof appears as a corollary of the calculations done in this section. Starting our from the gradient descent on the cost function we calculate.

$$\begin{aligned} -\frac{\mathrm{d}C}{\mathrm{d}W_i} &\overset{\text{Form of L; equation (5.38)}}{=} -\frac{\mathrm{d}}{\mathrm{d}W_i} \frac{\partial}{\partial \beta} L \bigg|_{\beta=0} \overset{\text{lemma 5}}{=} -\frac{\mathrm{d}}{\mathrm{d}W_i} \frac{\mathrm{d}}{\mathrm{d}\beta} L \bigg|_{\beta=0} = \\ &= -\frac{\mathrm{d}}{\mathrm{d}\beta} \frac{\mathrm{d}}{\mathrm{d}W_i} L \bigg|_{\beta=0} \overset{\text{lemma 4}}{=} -\frac{\mathrm{d}}{\mathrm{d}\beta} \frac{\partial}{\partial W_i} L \bigg|_{\beta=0} = \lim_{\beta \to 0} (u_i - W_i \bar{r}_{i-1}) \bar{r}_{i-1}^T \approx \\ &\approx \frac{1}{\beta} (u_i - W_i \bar{r}_{i-1}) \bar{r}_{i-1}^T \propto -\frac{\partial E}{\partial W_i} \end{aligned} \tag{5.41}$$

where the quantities u_i, $\bar{r}_i$ and $\bar{r}_{i-1}$ are meant with $\beta \neq 0$ and $\beta \ll 1$ small. □

In summary, we have shown that in this framework the plasticity rule carries out a gradient-descent learning on the cost function. In particular, in case of a layered feed-forward network, the learning corresponds to the backpropagation algorithm (section 5.1.1). Naturally, the learning rate still has to be properly chosen to ensure the convergence of gradient descent.

5.1.3 Policy gradient reinforcement learning with a deep neural network

As a baseline for comparison, we consider the standard policy gradient with a deep neural network as a model [Sutton and Barto, 2018]. The following model does not have any real notion of time unlike the time-continuous approach, this model only implements a stochastic action to a given state (input). We consider a neural network parametrized with $\boldsymbol{W}$ and with a softmax readout on the possible actions (available classes). In general, $\boldsymbol{W}$ is a placeholder for the parameters (all the weights) of the deep network. We will see that the formula falls into two parts: 1) a term interpreted as an error-vector on the action values, that is the last layer in the network; and 2) a backpropagation of this action vector through the network. This interpretation is especially important when we compare it to the mechanistic model in the least action framework. With the softmax action selection policy, the probability of an action a_i to an environment-state (input image) $\mathbf{x}$ is given by:

$$p(a_i|\mathbf{x}) = \underset{i}{\mathrm{softmax}}\,(\mathbf{a}) = \frac{\exp(q_i(\mathbf{x}))}{\sum_j \exp(q_j(\mathbf{x}))} \quad , \tag{5.42}$$

where the function $q_i(\cdot)$ denotes the action value of action a_i, which is implicitly parameterized by $\boldsymbol{W}$. In a physiological interpretation $q_i(\mathbf{x})$ is for example the firing rate of neuron i in the action layer. Following the derivation given in Sutton and Barto [2018], we apply policy gradient to the model, i.e. gradient ascent on the expected reward with respect to policy with the parameterization $\boldsymbol{W}$. The mean expected reward is given by:

$$\langle R \rangle = \sum_{\mathbf{x}} \sum_i R(a_i, \mathbf{x}) p(a_i|\mathbf{x}) \quad , \tag{5.43}$$

where $R(a_i, \mathbf{x})$ is the observed reward if action a_i is taken in response to an input state $\mathbf{x}$, and $p(a_i|\mathbf{x})$ is the probability to take action a_i given an environment state $\mathbf{x}$. Hence:

$$\begin{aligned}\partial_W \langle R \rangle &= \partial_W \sum_{\mathbf{x}} \sum_i R(a_i, \mathbf{x}) p(a_i|\mathbf{x}) = \sum_{\mathbf{x}} \sum_i R(a_i, \mathbf{x}) \partial_W p(a_i|\mathbf{x}) = \\ &= \sum_{\mathbf{x}} \sum_i R(a_i, \mathbf{x}) \partial_W \log(p(a_i|\mathbf{x})) p(a_i|\mathbf{x}) \quad .\end{aligned} \tag{5.44}$$

We apply the chain rule for the derivation with $\boldsymbol{W}$:

$$\partial_W = \sum_k \frac{\partial q_k}{\partial \boldsymbol{W}} \frac{\partial}{\partial q_k} \quad , \tag{5.45}$$

which yields:

$$\partial_{\mathbf{W}} \langle R \rangle = \sum_{\mathbf{x}} \sum_{i} R(a_i, \mathbf{x}) \sum_{k} \left[\frac{\partial q_k}{\partial \mathbf{W}} \frac{\partial}{\partial q_k} \log(p(a_i|\mathbf{x})) \right] p(a_i|\mathbf{x}) \tag{5.46}$$

Using equation (5.42), we calculate:

$$\begin{aligned} \frac{\partial}{\partial q_k} \log(p(a_i|\mathbf{x})) &= \frac{\partial}{\partial q_k} \left[q_i - \log \left(\sum_j \exp(q_j) \right) \right] = \\ &= \delta_{ki} - \frac{\exp(q_k)}{\sum_j \exp(q_j)} = \delta_{ki} - p(a_k|\mathbf{x}) \quad . \end{aligned} \tag{5.47}$$

Plugging this into equation (5.46), yields the formula for the gradient:

$$\partial_{\mathbf{W}} \langle R \rangle = \sum_{\mathbf{x}} \sum_{i} p(a_i|\mathbf{x}) R(a_i, \mathbf{x}) \sum_{k} \frac{\partial q_k}{\partial \mathbf{W}} \left[\delta_{ki} - p(a_k|\mathbf{x}) \right] \quad . \tag{5.48}$$

In this formula we can associate to each term an interpretable meaning:

1. The sum over the states $\sum_{\mathbf{x}}$ is the average over the training set, for example in a classification it would be the average over all training images. Here for the sake of simplicity, we assumed that the probability distribution of the states is flat.

2. The action-probability-weighted sum $\sum_i p(a_i|\mathbf{x})$ means a sampling from the actions taken in response to state $\mathbf{x}$.

3. $R(a_i, \mathbf{x})$ is the obtained reward. This can be understood as a modulator signal on the plasticity, similar to the third factor in three factor learning rules [Frémaux and Gerstner, 2015]. Following [Sutton and Barto, 2018], we replace $R(a_i, \mathbf{x})$ by the reward prediction error $\delta_{\mathrm{RP}} = R(a_i, \mathbf{x}) - \langle R \rangle$. We can do this because the addition of a constant baseline does not change the gradient. In practice, the $\langle R \rangle$ has to be estimated during learning via for example a moving average.

4. $\sum_k \frac{\partial q_k}{\partial \mathbf{W}}$ contains implicitly the Jacobian matrix from the output layer of the network to the single synapses. For a feed-forward neural network it could be rolled out to the backpropagation algorithm.

5. $[\delta_{ki} - p(a_k|\mathbf{x})]$ behaves like an error-vector in the output layer.

The equation results in the following algorithm with online learning/stochastic gradient ascent updates (algorithm 5.1). In this work, we use the standard policy gradient algorithm as a comparison for the introduced learning rules.

Algorithm 5.1: Standard policy gradient algorithm applied to deep reinforcement learning. The parameters γ, $N_{\text{iterations}}$ and η are metaparameters of the learning algorithm: η is the learning rate, γ is the parameter of the moving average and $N_{\text{iterations}}$ is the number of iterations to be made.

```
Initialize $\boldsymbol{W}$ randomly;
$\langle R \rangle := 0$;
for $n := 1$ to $N_{\text{iterations}}$ do
    get state/image $\mathbf{x}$ from dataset;
    obtain action values $\mathbf{q}$;
    select action $i$ with softmax $p(a_i|\mathbf{x}) = \frac{\exp(q_i)}{\sum_j \exp(q_j)}$;
    obtain reward $R := R(a_i, \mathbf{x})$;
    update parameters $\boldsymbol{W} = \boldsymbol{W} + \eta R(a_i, \mathbf{x}) \sum_k \frac{\partial q_k}{\partial \boldsymbol{W}} [\delta_{ki} - p(a_k|\mathbf{x})]$;
    update the mean reward $\langle R \rangle = \gamma R + (1 - \gamma)\langle R \rangle$;
end
```

5.2 Time-continuous deep reinforcement learning

We introduce the framework of reinforcement learning with the principle of least action framework. First, we give a general description of the working mechanism of the model and then we devote a section to the detailed calculations. Second, because the simulations of the framework are computationally expensive, we also describe an analogy of the model using artificial neural networks without temporal dynamics. This analogous model is helpful for simulations where using the full model is prohibitively expensive. Further, it gives another intuitive view on the model.

5.2.1 Theory outline

In the following, we describe the setup of the time-continuous reinforcement learning model. In the current project, we applied the framework to reinforcement-learning-based image classification. The main components of the model are: 1) the network, which is a feed-forward network in our case, 2) the differentiable input, 3) the action selection mechanism, 4) the exploration mechanism, 5) the soft winner-nudges-all (WNA) circuit that gives rise to the error-vector, and finally 6) the local plasticity rule.

Network setup, cost, energy and the network dynamics

The used network is a feed-forward network with a WNA circuit in the action layer (figure 5.4). The input is presented at the bottom of the network, in the lowest layer, as a current. The total energy is given as in the case of supervised learning:

$$L = E + \beta C_{\text{WNA}} = \sum_{i=1}^{N} \frac{1}{2} \|u_i - W_i \bar{r}_{i-1}\|^2 + \beta C_{\text{WNA}} \quad , \tag{5.49}$$

where the prediction error E is the same as in the case of the supervised learning model. We cannot give a closed form of C_{WNA} in case of reinforcement learning because the theory only allows curl-free connection patterns and the required WNA structure has a curvature. Instead, we define not C_{WNA}, but rather the error $\bar{e}_N$ (compare to equation (5.7)) and argue that $\bar{e}_N$ can be linearized for any membrane potential values and hence a corresponding $C^{\text{lin}}_{\text{WNA}}$ can be given. The detailed calculations are given in section 5.2.2. Because β is required to be small, all previous calculations apply (section 5.1.2). Hence, we postulate the neural dynamics:

$$\begin{aligned} \tau \dot{u}_i &= W_i r_{i-1} - u_i + e_i \quad , \\ r_i &= \bar{r}_i + \tau \dot{\bar{r}}_i \; ; \quad e_i = \bar{e}_i + \tau \dot{\bar{e}}_i \quad , \\ \bar{e}_i &= \bar{r}_i \odot W_{i+1}^T \left(u_{i+1} - W_{i+1} \bar{r}_i \right) \quad , \\ \bar{e}_N &= \beta M \bar{r}_N \quad , \end{aligned} \tag{5.50}$$

where M is the WNA matrix with $m_{ii} = 1$ and $m_{ij} = -\frac{1}{K-1} \; \forall i \neq j$, with K neurons in the last layer. The matrix M represents the WNA connectivity in the output layer. The neurons in the last layer project each to itself with excitatory connections and to each other with inhibitory connections, similar to the architecture used in the reinforcement learning model of Frémaux et al. [2013]. Note that the WNA circuit is multiplied by β, hence it is much weaker than the bottom-up signal. Crucially, the role of the WNA is not the action selection, but the generation of an error-vector signal, which is then propagated back to the lower layers via the microcircuits.

Input, action selection and the exploration

In the presented classification setup, each image is presented for a given T_{image} time. For a smooth transition, the input current $I(t)$ for each neuron is ramped up in the beginning and phased out at the end of the presentation in a differentiable way according to (figure 5.5):

$$I(t) = \begin{cases} \frac{I_0}{2} \left(1 - \cos \left(\frac{t}{T_{\text{ramp}}} \pi \right) \right) & \text{if } t \leq T_{\text{ramp}} \quad , \\ I_0 & \text{if } T_{\text{ramp}} < t < T_{\text{image}} - T_{\text{ramp}} \quad , \\ \frac{I_0}{2} \left(1 + \cos \left(\frac{t - (T_{\text{image}} - T_{\text{ramp}})}{T_{\text{ramp}}} \pi \right) \right) & \text{if } T_{\text{image}} - T_{\text{ramp}} < t \quad . \end{cases} \tag{5.51}$$

Here T_{ramp} is the time of the ramp-up and phase-out phase, and I_0 is the unmodulated value of the current to the given neuron. The time is meant as inside one of the image presentations. The smooth image presentation preserves the input-following property of the network. With jumps between the single images,

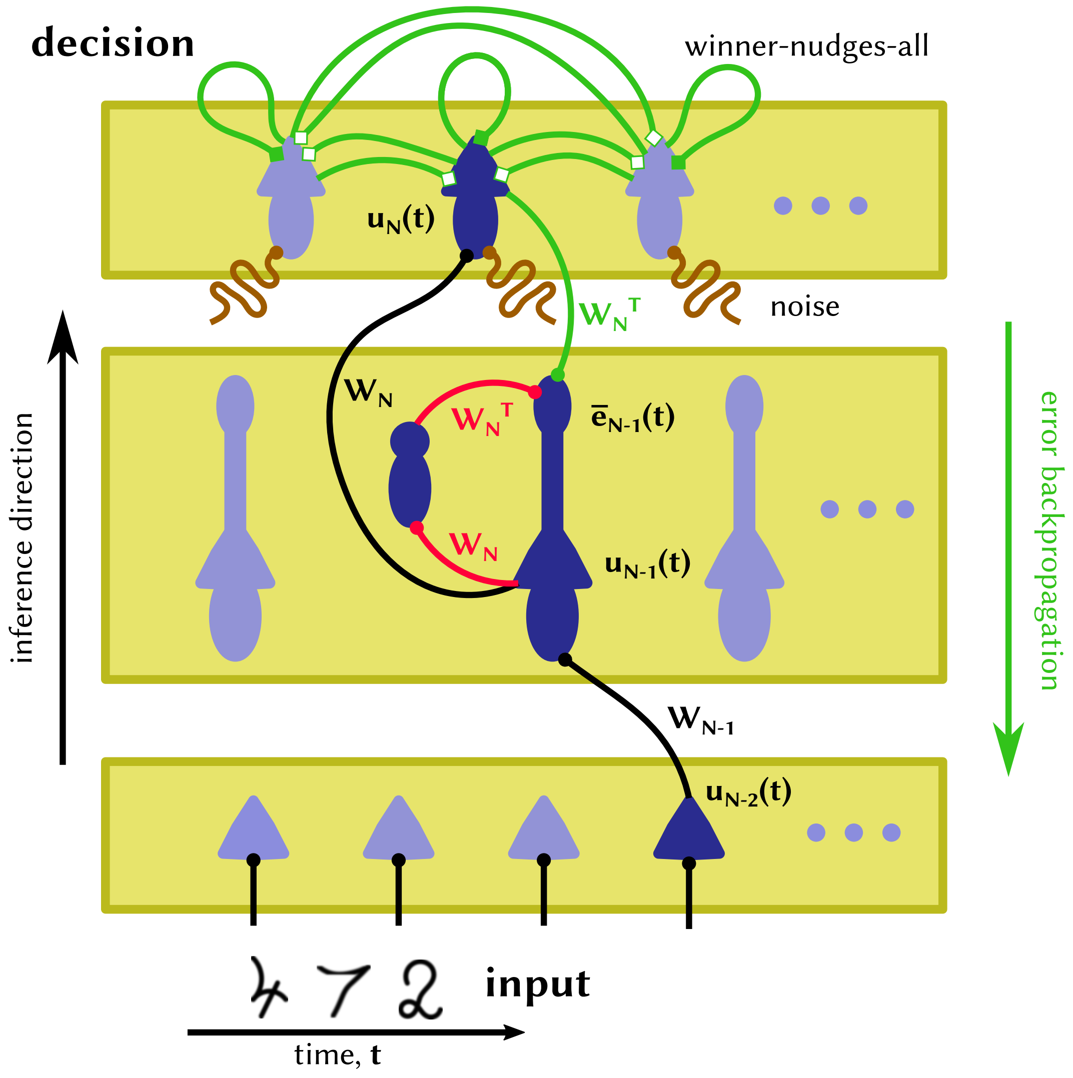


Figure 5.4: Network setup of the reinforcement learning model In the feed-forward direction (from bottom to top), the pyramidal neurons act similarly as abstract neurons in a conventional neural network, but the pyramidal neurons additionally have a leaky integrator dynamics. The output or action neurons in the last layer receive additional noise to enhance exploration. The action neuron with the highest activity represents the decision of the network. In the error backpropagation direction (from top to bottom) the output neurons nudge each other via the WNA circuit, hence their activity differs from the dendritic prediction. In each hidden layer, stereotypical microcircuits of inter-neurons and pyramidal neurons project back the prediction error. In contrast to the supervised case (figure 5.2), the error-vector is not created by an external supervisor but by the network itself. Figure adapted from Dold [2020].

the network would still learn, but the transient phases would put a lower limit on T_{image}. Furthermore, the transient phase should be smaller than the constant phase.

For the action readout, we associate each of the neurons in the output layer with one of the possible actions. In our classification setup there are discrete actions, hence we can directly identify actions with neurons. We take the neuron with the highest low-pass activity $\bar{r}$ before the phase-out of each presented image starts (figure 5.5). In the pathological case of two or more equal winner neurons, the winner is chosen randomly.

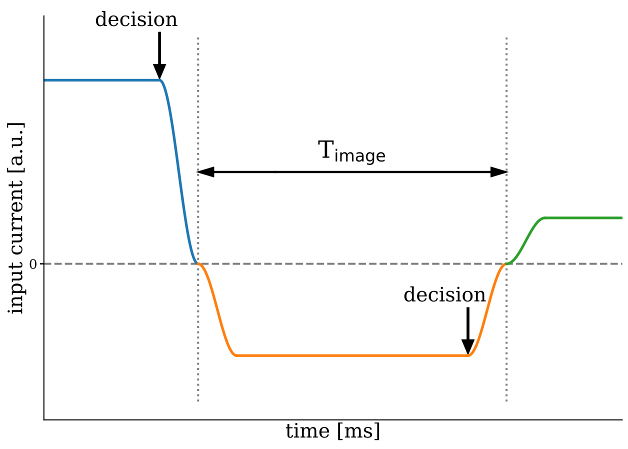


Figure 5.5: Example of the time-continuous input current. The input current to the first layer of the neurons is modulated with time-continuous and everywhere differentiable ramp up and ramp down. The differentiable input makes sure that there are no transients between the inputs where assumptions of the input tracking property are violated. There is no overlap between the inputs; the previous input is already faded out before the presentation of the next input starts. The colors indicate the distinct input images, the vertical dotted lines separate the inputs and the vertical arrows the read-out time of the decisions.

To encourage exploration we add noise to the action neurons. The noise term appears as an additional stochastic current input to the basal dendrite of the neurons. To include a realistic dynamics to the neurons, we assume that the noise ξ follows an Ornstein-Uhlenbeck [Honerkamp, 1993] process with:

$$d\xi = \frac{1}{\tau_{\text{OU}}}(\mu_{\text{OU}} - \xi)dt + \sqrt{\frac{2}{\tau_{\text{OU}}}}\sigma_{\text{OU}}d\mathcal{W} \quad , \tag{5.52}$$

where τ_{OU} is the autocorrelation time, μ_{OU} is the equilibrium point and σ_{OU} is the standard deviation of the steady-state solution. $d\mathcal{W}$ is the Wiener-process [Honerkamp, 1993] with zero mean and a variance of one. The mean μ_{OU} of the Ornstein-Uhlenbeck process is set to zero; a non-zero μ_{OU} would only mean a constant bias in the neuron activity, which would be incorporated into the learning. The resulting noise dynamics has a similar characteristic as the noise in the LIF Sampling model (section 3.2), and hence can be thought of as a result of background activity arriving at the basal dendrite of the neuron. Using the equation

$$\dot{\bar{\xi}} = \frac{\xi - \bar{\xi}}{\tau} \quad , \tag{5.53}$$

we incorporate the noise into the model by modifying the error term of the action neurons to include the noise:

$$E_N = \frac{1}{2} \left\| u_i - W_i \bar{r}_{i-1} - \bar{\xi} \right\|^2 \quad . \tag{5.54}$$

The noise is assumed to be independent and follow the same stochastic process for each action neuron.

We argue that this kind of noise is more biological than an ϵ-greedy policy found in classical machine learning. In the ϵ-greedy policy [Sutton and Barto, 2018], the most active action is chosen with a probability of $1 - \epsilon$ and a random action with a probability of $\epsilon \ll 1$. This would require an additional mechanism that stochastically turns-on equating the probability of the actions. Further, it should be in synchrony with the presentation of the image. In contrast, our exploration noise only requires a source of fluctuating input. This might be readily available from the activity of other brain areas similar to the noise mechanisms in chapter 3 and Dold et al. [2019]. Furthermore, the noise input does not interfere with the error-vector generation from the WNA network because the noise appears on the basal dendrite of the action neurons, and hence it is part of the dendritic prediction.

Note, that the expected membrane potential of the action neurons, where the expectation is taken over the realization of the exploration noise, plays the same role as the **q**-values of the actions in the policy-gradient learning rule (section 5.1.3). To emphasize this equivalent function and to simplify the notation, we refer to the expected membrane potentials as q-values: $\mathbf{q} = \langle \boldsymbol{u} \rangle_{\text{noise}}$. Here, $\langle \cdot \rangle_{\text{noise}}$ means the expected value over the possible realization of the noise.

The plasticity rule

We postulate the plasticity rule as a combination from the standard principle of least action framework [Senn et al., in preparation] and from the theory of three-factor learning rules [Frémaux and Gerstner, 2015, Gerstner et al., 2018], similar to the learning rule in chapter 4. The dendritic prediction errors are integrated into a decaying eligibility trace over time with a time-constant of τ_{elig}. When a reward signal arrives, the eligibility trace is applied as weight update modulated by the reward prediction error. Summarized, it yields:

$$\begin{aligned} \dot{W}_i &= \frac{\eta}{\tau_{\text{elig}}} \left(R - \langle R \rangle \right) \int_{-\infty}^{t} \kappa(\hat{t}) \exp\left(-\frac{t - \hat{t}}{\tau_{\text{elig}}} \right) \mathrm{d}\hat{t} \quad , \\ \kappa(t) &= \underbrace{\left(u_i - W_i \bar{r}_{i-1} \right)}_{\propto \bar{e}_i} \bar{r}_{i-1}^{T} \quad , \end{aligned} \tag{5.55}$$

with η the learning rate, R the immediate received reward and $\langle R\rangle$ the expected reward under the current policy, that is under the policy given by the current network parameters. The term $R - \langle R\rangle$ acts as the reward-prediction-error in biology [Schultz et al., 1997, Niv, 2009], in reinforcement learning models [Frémaux and Gerstner, 2015, Sutton and Barto, 2018], and in particular in the policy-gradient method (section 5.1.3). $\langle R\rangle$ implicitly takes the role of a critic module from actor-critic architectures [Sutton and Barto, 2018]. In our implementation, $\langle R\rangle$ is estimated by a moving average updated in the iterations as:

$$\langle R\rangle_n = \langle R\rangle_{n-1}\,\gamma + (1-\gamma)\,R_{n-1} \quad , \tag{5.56}$$

with γ the meta-parameter of the moving average. A small γ means that $\langle R\rangle$ tries to follow closely the immediate rewards and a large γ means that the expected reward is a moving average over many iterations. It can be shown (section 5.2.2), that the postulated learning rule implements Hill-climbing on the expected reward $\langle R\rangle$. The winner-nudges-all error $e_{\mathrm{WNA}} = \bar{e}_N = \beta M \bar{r}_N$ lies within 90 degrees of the policy-gradient error $e_{\mathrm{PG}} = \delta_{ij} - p(\text{action i})$ with j the taken action (equation (5.48)). Mathematically, the scalar product of the two error-vectors is non-negative

$$\langle e_{\mathrm{WNA}}; e_{\mathrm{PG}}\rangle \geq 0 \quad , \tag{5.57}$$

where $\langle \cdot ; \cdot \rangle$ denotes the Euclidean scalar product. The similarity and the differences are exemplified in figure 5.6.

Homeostatic plasticity, its inspiration and importance

Two types of homeostatic plasticity aid the learning. The first type, we call extreme value homeostasis and it acts in a way that it keeps the membrane potential of the neurons within limits. It is given by

$$\begin{aligned} \dot{W}_i^{\text{e-hom}} &= \frac{\eta_{\text{e-hom}}}{\tau_{\text{elig}}} \left| R - \langle R\rangle \right| \int_{-\infty}^{t} \kappa_{\text{ehom}}(\hat{t}) \exp\left(-\frac{t-\hat{t}}{\tau_{\text{elig}}}\right) \mathrm{d}\hat{t} \\ \kappa_{\text{e-hom}}(t) &= \left[\max\left(0; u_{\text{ulow}} - u_i\right) - \max\left(0; u - u_{\text{uhigh}}\right)\right] \bar{r}_{i-1}^{T} \quad . \end{aligned} \tag{5.58}$$

If the membrane potential of the neuron, and correspondingly the activity, is above the threshold value of u_{uhigh}, then all incoming synapses are reduced. Conversely, if the membrane potential is below the lower threshold u_{ulow}, then all the incoming synapses are strengthened. $\eta_{\text{e-hom}}$ is the learning rate of this type of homeostasis.

The form of the homeostasis is analogous to the form of the reward maximizing plasticity (equation (5.55)), hence we can understand it similarly as a combination of an eligibility trace of local quantities and a modulating factor. The membrane potential is compared to an expected range of valid values instead of the dendritic prediction. The homeostasis is modulated by a third factor, but importantly not by the reward prediction error, but by its absolute value. In this sense the three-factor homeostatic plasticity is similar to surprise-based or modulated learning rules, for example reviewed in Gerstner et al. [2018]. For the function, the homeostatic

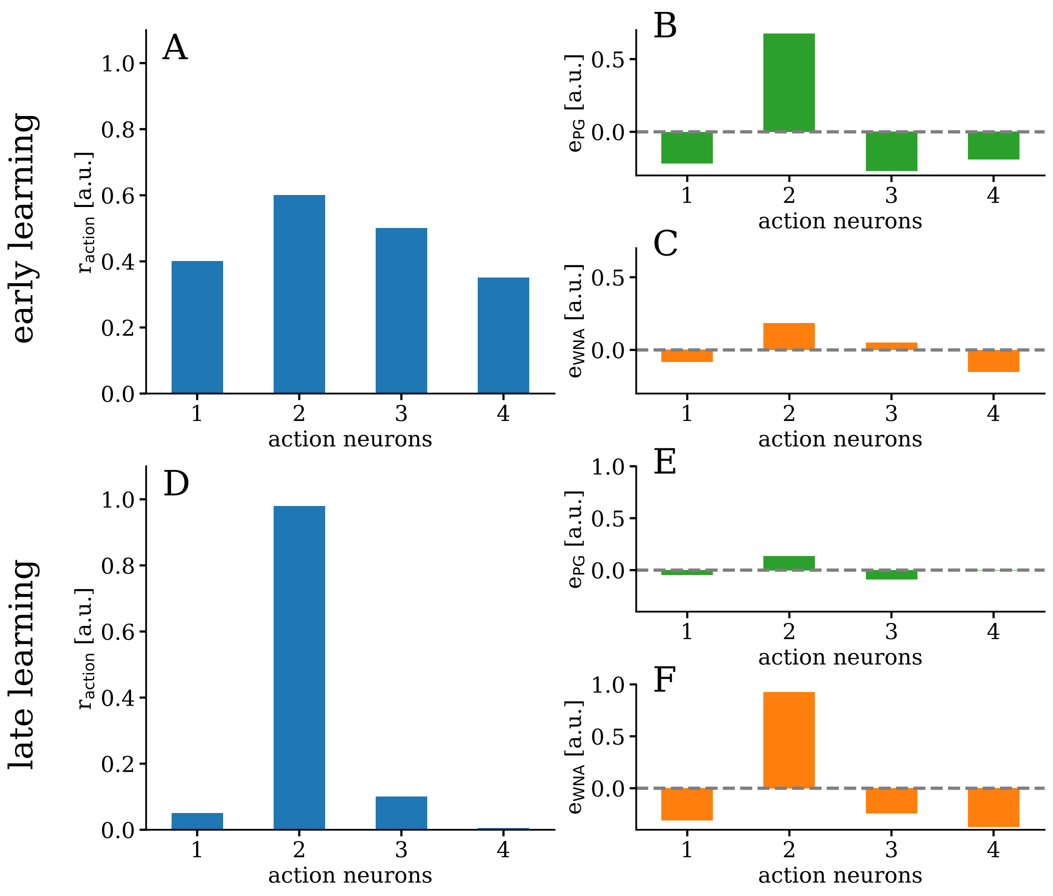


Figure 5.6: Difference between policy-gradient and winner-nudges-all error. The winner-nudges-all error e_{WNA} lies within 90 degrees of the policy-gradient error e_{PG} (equation (5.48)), that is $\langle e_{\text{WNA}}; e_{\text{PG}} \rangle \geq 0$. However the differences are apparent. **(A)** In the early phase of learning, when all the action neurons have similar activity and hence all actions have similar probability, the magnitude of e_{PG} **(B)** is larger than that of e_{WNA} **(C)**. Hence, exploration helps the learning to break the symmetry. **(D)** In the late phase of learning, when one action dominates the policy-gradient error e_{PG} disappears **(E)**, consolidating the learning. In contrast, **(F)** e_{WNA} is large in this phase, and the learning tends to explode. A homeostasis is required to keep the learning from this explosion (equation (5.58)). For the sake of simplicity, we work with dimensionless quantities, but e_{WNA} and e_{PG} take the same dimension.

plasticity should not react to the sign of the reward prediction error, because its effect is not correlated with the taken action.

Furthermore, at the late stage of learning when the obtained reward reaches a plateau and the predicted reward $\langle R \rangle$ approximates the obtained error, the homeostatic plasticity should also disappear to allow for the consolidation of the learned policy. The extreme-value homeostasis prohibits the divergence of the learning in the late phase of learning with the winner-nudges-all error (see figure 5.6 and figure 5.7). If a class is already learned correctly, while others are still learning, then the diverging winner-nudges-all error would drive the learning only towards the already learned action. This divergence would interfere with the learning of the other classes and the learning would break down. In summary, extreme value homeostasis compensates for the run-away self-strengthening effect of the already learned classes.

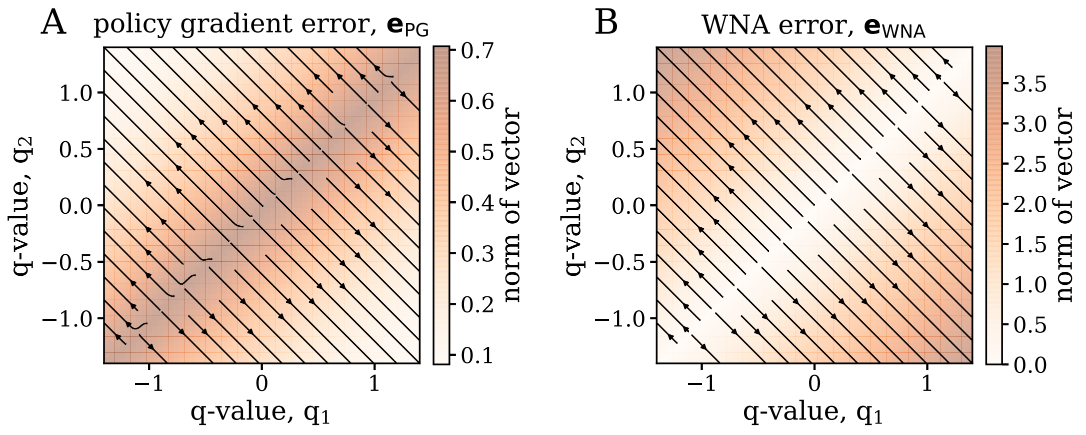


Figure 5.7: Difference between policy-gradient and winner-nudges-all error in the phase space. The figures show the resulting error-vectors in the phase-space of the **q**-values. With this choice, we simplified the plots, because we neglected the non-linearities of the activation functions, but the message stays the same. **(A)** In case of the policy-gradient $\mathbf{e}_{\mathrm{PG}}$, the magnitude of the error is large at the $q_1 = q_2$ separatrix and small further away from it. **(B)** In the case of the winner-nudges-all error $\mathbf{e}_{\mathrm{WNA}}$, we observe the opposite: the norm of the error-vector is small at the separatrix and explodes further away. The difference between both errors appears on the figures as the streamplots agree in the direction, however not in the magnitude. The extreme-value homeostatic plasticity is required to compensate for the explosion.

We call the second sort of homeostatic plasticity midpoint homeostasis. It pulls the membrane potential of the neurons to a common natural point. Effectively, it pulls the activity of the action neurons to a good starting point and further helps to avoid that a single learned action dominates the learning. However, we implement it for every synapse because we deemed a special plasticity acting only on the output neurons biologically less plausible. In formulas, the midpoint homeostasis states

$$\begin{aligned} \dot{W}_i^{\text{m-hom}} &= \frac{\eta_{\text{m-hom}}}{\tau_{\text{elig}}} \left| R - \langle R \rangle \right| \int_{-\infty}^{t} \kappa_{\text{m-hom}}(\hat{t}) \exp\left(-\frac{t-\hat{t}}{\tau_{\text{elig}}} \right) \mathrm{d}\hat{t} \quad , \\ \kappa_{\text{m-hom}}(t) &= (u_{\text{mid}} - u_i)\, \bar{r}_{i-1}^T \quad . \end{aligned} \tag{5.59}$$

with u_{mid} the target midpoint membrane potential and $\eta_{\text{m-hom}}$ the learning rate of this type of homeostasis. The rational behind the midpoint homeostasis is that exploration should be aided in higher dimensional action spaces. If, by initialization or by chance during learning, the network is unable to take the rewarded action, the midpoint homeostasis pulls the actions close to each other by changing the synaptic weights. When the action values are closer to each other than the typical magnitude of the exploration, then the network will eventually explore the rewarded action. Similarly to the extreme-value homeostasis, the midpoint homeostasis is modulated by the absolute value of the reward-prediction error and the form of the plasticity rule is analog to the reward-maximizing plasticity. The midpoint homeostasis is inspired by the homeostatic learning rule in [Habenschuss et al., 2012], where a similar rule is derived for unsupervised learning.

Note, we added both homeostatic learning rules in a bottom-up fashion inspired by other works and the described rationales. They do not have rigorous derivations from first principles.

5.2.2 Detailed calculations of the time-continuous reinforcement learning model

In the following, we give the detailed calculations regarding the reinforcement learning in the principle of least action framework. First, we show that the current form of the theory can only support curl-free synaptic connectivity from the cost function, and argue why the theory still applies via local approximations. Then, we establish the necessary mathematical groundwork and show the proof that learning in the winner-nudges-all error approximates the policy-gradient error within 90 degrees and hence it implements Hill-climbing on the expected reward.

Approximate cost function for the winner-nudges-all interaction

The WNA term in the dynamic equations (equation (5.7)) cannot be represented with a single cost function in a closed form. The reason for this is that the WNA term has in general a rotational component, while any term derived from a scalar cost function via the postulated dynamics is necessarily curl-free. Still, we argue that we can use an approximative cost function, and hence all the derived results (phase-less learning, look-ahead dynamics) apply here as well.

First, we exemplify that the WNA term has a rotation component in the general case. For simplicity, we consider the curl of the winner-nudges-all circuit in two dimensions and only on the contribution from the term $\nabla_u C$. In two dimensions, the winner-nudges-all circuit takes the form:

$$M\bar{r}_N = \begin{pmatrix} \bar{r}_N^{(1)} - \bar{r}_N^{(2)} \\ -\bar{r}_N^{(1)} + \bar{r}_N^{(2)} \end{pmatrix} \quad . \tag{5.60}$$

with $\bar{r}_N^{(1)}$, $\bar{r}_N^{(2)}$ the low-pass filtered activity of the two action neurons. The curl value is then

$$\operatorname{curl}\left(M\bar{r}_N\right) = \frac{\partial}{\partial u_1}\left(-\bar{r}_N^{(1)} + \bar{r}_N^{(2)}\right) - \frac{\partial}{\partial u_2}\left(\bar{r}_N^{(1)} - \bar{r}_N^{(2)}\right) = -\frac{\partial \bar{r}_N^{(1)}}{\partial u_1} + \frac{\partial \bar{r}_N^{(2)}}{\partial u_2} \neq 0 \quad . \tag{5.61}$$

Second, we calculate that any term F_{cost} derived from a scalar cost function in the dynamic equations (equation (5.7)) takes the from:

$$F_{\text{cost}} = \nabla_u C + \tau \frac{\mathrm{d}}{\mathrm{dt}} \nabla_u C \quad , \tag{5.62}$$

where ∇_u is the differential operator according to the membrane potential $\boldsymbol{u}$ of the action neurons. We use the definition of the operators to rewrite the second term. In the following equations $\vec{e}_j$ is the unit vector in the dimension of the j-th neuron. We calculate the second term explicitly

$$\begin{aligned} \tau \frac{\mathrm{d}}{\mathrm{d}t} \nabla_u C &= \tau \left(\sum_i \frac{\partial u_i}{\partial t} \frac{\partial}{\partial u_i} + \frac{\partial}{\partial t} \right) \left(\sum_j \vec{e}_j \frac{\partial C}{\partial u_j} \right) = \\ &= \tau \sum_j \vec{e}_j \left(\sum_i \frac{\partial^2 C}{\partial u_i \partial u_j} \frac{\partial u_i}{\partial t} + \frac{\partial}{\partial u_j} \frac{\partial C}{\partial t} \right) = \\ &= \tau \sum_j \vec{e}_j \frac{\partial}{\partial u_j} \sum_i \frac{\partial C}{\partial u_i} \frac{\partial u_i}{\partial t} + \tau \sum_j \vec{e}_j \frac{\partial}{\partial u_j} \frac{\partial C}{\partial t} = \\ &= \tau \nabla_u \left(\nabla_u C \cdot \dot{u} + \frac{\partial C}{\partial t} \right) \quad . \end{aligned} \tag{5.63}$$

Hence:

$$F_{\text{cost}} = \nabla_u C + \tau \nabla_u \left(\nabla_u C \cdot \dot{u} + \frac{\partial C}{\partial t} \right) \tag{5.64}$$

Because all three terms C, $\nabla_u C \cdot \dot{u}$ and $\frac{\partial C}{\partial t}$ are scaler fields (potential) in terms of u, they are also all curl-free. Hence:

$$\operatorname{curl}\left(F_{\text{cost}}\right) = 0 \quad . \tag{5.65}$$

Finally, despite these observations on the rotation of the terms, we argue that we can apply the principle of least action theory with a small modification, because the effect of the cost function is modulated by β which is required to be small by the theory (theorem 1). Therefore, the dynamics of the network is mainly

driven by the inference direction, and the effect of the cost function is only a slight modification of it. We approximate the term $M\bar{r}_N$ using the activity values $\bar{r}_N^{\beta=0}$
,

$$M\bar{r}_N \approx M\bar{r}_N^{\beta=0} \quad . \tag{5.66}$$

Because $M\bar{r}_N^{\beta=0}$ is a constant vector in u, it has a corresponding cost function, which we can use as an approximation in the theory.

Derivation of the Hill-climbing property

In the following, we present the calculations corresponding to deriving the Hill-climbing property. First, we lay the groundwork by defining concepts of action selection and action probabilities. *A priori* we are free to model the action selection, and we could choose from a broad range of potential mechanisms used in different publications, for example the classic ϵ-greedy [Pozzi et al., 2018], the softmax action selection [Whiteson and Stone, 2006] or direct Poissonian competition [Leimer et al., 2019]. A common property of all these mechanisms is that single neurons encode the actions and that a property of the neurons represents (a parameter of) some action value, similar to the q-values found in classical reinforcement learning (section 2.1.3, Sutton and Barto [2018]). We denote this action-value vector $\mathbf{q}$, where the single q_i represents the action-value of action i. Or in other words, the q_i is the action-value of neuron i. Obviously, if the action value q_i increases while all the others stay constant, it should increase the probability of action a_i and decrease the probability of all the others. Finally, we require that the competition among the actions should solely depend on the relative values of q_i and not on their absolute value, unlike for example in the Poissonian competition model. In the following definition, we formulate these ideas mathematically.

Definition 5 (Neurally compatible action probability) *Let there be N neurons, each holding the action values* $\mathbf{q} = (q_1, q_2, \ldots, q_N)$*; N actions* $a_1, a_2, \ldots, a_N$*; and action probabilities* $P_i := P(a_i; \mathbf{q})$*. The action probabilities* P_i *are parametrized by the vector of the action values* $\mathbf{q}$*. We call the action probabilities neurally compatible if they fulfill:*

(i) $\frac{\partial P_k}{\partial q_k} \geq 0 \quad \forall k \in \{1, 2, \ldots, N\}$,

(ii) $\frac{\partial P_k}{\partial q_i} \leq 0 \quad \forall k, i \in \{1, 2, \ldots, N\} \wedge k \neq i$,

(iii) $P(\mathbf{q}) = P(\mathbf{q} + \lambda\vec{1}) \quad \forall \lambda \in \mathbb{R}$,

with $\vec{1} = (1; 1; \ldots; 1)$ *the vector with ones as entries in each dimension.*

From the definition we can immediately derive further properties of the neurally compatible action selection.

Lemma 6 (Properties of action selection) *Let there be N neurons, each with the action values* $\mathbf{q} = (q_1, q_2, \ldots, q_N)$*; N actions* $a_1, a_2, \ldots, a_N$*; and corresponding neurally compatible action selection probabilities* $P_i := P(a_i; \mathbf{q})$*, then* *(i)* $\frac{\partial \log(P_k)}{\partial q_k} \geq 0 \quad \forall k \in \{1, 2, \ldots, N\}$,

(ii) $\frac{\partial \log(P_k)}{\partial q_i} \leq 0 \quad \forall k, i \in \{1, 2, \ldots, N\} \wedge k \neq i$,

(iii) $\frac{\partial P_k}{\partial q_j} = -\sum_{\substack{i=1 \\ i \neq j}} \frac{\partial P_k}{\partial q_i} \quad \forall k, j \in \{1, 2, \ldots, N\}$,

(iv) $\frac{\partial \log(P_k)}{\partial q_j} = -\sum_{\substack{i=1 \\ i \neq j}} \frac{\partial \log(P_k)}{\partial q_i} \quad \forall k, j \in \{1, 2, \ldots, N\}$.

Proof. $(i), (ii)$**:** Properties (i) and (ii) are trivial consequences of the definition 5 (i) and (ii) and the strict monotonicity of the logarithm function.

(iii)**:** For $k, j \in \{1, 2, \ldots, N\}$ consider the $P_k(\mathbf{y})$ with $\mathbf{y} = \mathbf{q} + \lambda \vec{1}$ as function of lambda, and take its total derivative. By definition:

$$\begin{aligned} \frac{\mathrm{d}P_k(\mathbf{q} + \lambda\vec{1})}{\mathrm{d}\lambda} &= \lim_{\epsilon \to 0} \frac{P_k(\mathbf{q} + \lambda\vec{1}) - P_k(\mathbf{q} + (\lambda + \epsilon)\vec{1})}{\epsilon} \underset{definition\ 5(iii)}{=} \\ &= \lim_{\epsilon \to 0} \frac{P_k(\mathbf{q}) - P_k(\mathbf{q})}{\epsilon} = 0 \end{aligned} \tag{5.67}$$

On the other hand, we have:

$$\frac{\mathrm{d}P_k(\mathbf{y})}{\mathrm{d}\lambda} = \sum_{i=1}^{N} \frac{\partial P_k}{\partial q_i} \underbrace{\frac{\mathrm{d}q_i}{\mathrm{d}\lambda}}_{=1} + \underbrace{\frac{\partial P_k}{\partial \lambda}}_{=0} = \sum_{i=1}^{N} \frac{\partial P_k}{\partial q_i} \quad . \tag{5.68}$$

Combining equation (5.67) and equation (5.68) yields:

$$\frac{\partial P_k}{\partial q_j} = -\sum_{\substack{i=1 \\ i \neq j}} \frac{\partial P_k}{\partial q_i} \quad .$$

(iv)**:** The last property can be shown by straightforward calculation:

$$\frac{\partial \log(P_k)}{\partial q_j} = \frac{1}{P_k} \frac{\partial P_k}{\partial q_j} \underset{\text{property } (iii)}{=} -\frac{1}{P_k} \sum_{\substack{i=1 \\ i \neq j}} \frac{\partial P_k}{\partial q_i} = -\sum_{\substack{i=1 \\ i \neq j}} \frac{1}{P_k} \frac{\partial P_k}{\partial q_i} = -\sum_{\substack{i=1 \\ i \neq j}} \frac{\partial \log(P_k)}{\partial q_i} \quad .$$

□

To get closer to the desired model (section 5.2.1), we consider the model in terms of the probability distribution of the q-values. We do not define the action selection values, but we work with the outcome probability of the output neurons, for example the distribution of the firing rates. We only require that the probability distribution of the winner neuron follows a neurally compatible action selection mechanism defined in definition 5. We extract the key approximation from the Hill climbing proof as a separate statement.

Lemma 7 (The main approximation) *Let there be N random variables $u_i \in \mathbb{R}$ with an arbitrary random distribution $\mathbf{u} \sim p(\mathbf{u} : \mathbf{q})$, parametrized by the vector $\mathbf{q}$ such that $P_i(\mathbf{q}) = P(\operatorname{argmax}_j(\mathbf{u}) = i; \mathbf{q})$ is a neurally compatible action probability with corresponding action values $\mathbf{q}$. Further let there be a strictly monotone increasing function $r : \mathbb{R} \to \mathbb{R}; x \mapsto r(x)$. Let $r(\mathbf{u})$ denote the element-wise application of the activation function to the vector of membrane potential $r(\mathbf{u}) := (r(u_1), r(u_2), \ldots, r(u_N))$. In short-hand notation $\mathbf{r} = r(\mathbf{u})$. Consider a matrix $M \in \mathbb{R}^{N\times N}$ with $m_{ij} = -\frac{1}{N-1}$ $\forall i \neq j; i,j \in \{1,\ldots,N\}$, and with $m_{ii} = 1 \quad \forall i \in \{1,\ldots,N\}$ and vectors:*

$$\mathbf{v} := Mr(\mathbf{u}) \quad ,$$
$$w_k := \frac{\partial \log(P_{\operatorname{argmax}_j(\mathbf{u})}(\mathbf{q}))}{\partial q_k} \quad .$$

Then for any realization of $\mathbf{u}$:

$$\langle \mathbf{w}, \mathbf{v} \rangle \geq 0 \quad ,$$

where $\langle \cdot ; \cdot \rangle$ denotes the Euclidean scalar product. Equality holds if only if $u_1 = u_2 = \cdots = u_N$.

The choice of $\mathbf{u}$ as the symbol for the random distribution is on purpose. Later we will associate the noisy membrane potential of the output neurons (figure 5.4) with $\mathbf{u}$ here, and $\mathbf{q}$ will correspond to the membrane potential of the action neurons if the noise was turned off. Further, $\mathbf{r}$ will become the realized activity of the neurons.

Proof. First, notice that due to the strict monotonicity of $r(\cdot)$,

$$\operatorname{argmax}(\mathbf{u}) = \operatorname{argmax}(r(\mathbf{u})) = \operatorname{argmax}(\mathbf{r}) \quad . \tag{5.69}$$

For the sake of simplicity, we denote $\mathbf{r} = r(\mathbf{u})$. We start out with direct calculations:

$$\begin{aligned}\langle \mathbf{w}, \mathbf{v} \rangle &= \sum_{k=1}^{N} \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_k} \left(r_k - \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq k}}^{N} r_j \right) = \\ &= \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_{\operatorname{argmax}(\mathbf{u})}} \left(r_{\operatorname{argmax}(\mathbf{u})} - \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq \operatorname{argmax}(\mathbf{u})}}^{N} r_j \right) + \\ &+ \sum_{\substack{k=1 \\ k\neq \operatorname{argmax}(\mathbf{u})}}^{N} \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_k} \left(r_k - \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq k}}^{N} r_j \right) \quad .\end{aligned}$$

Now we use property (ii) from lemma 6:

$$\frac{\partial \log(P_{\operatorname{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_k} \leq 0 \quad \forall k \neq \operatorname{argmax}(\mathbf{u}) \quad .$$

And by direct calculation:

$$r_{\mathrm{argmax}(\mathbf{u})} \geq r_k \quad ,$$

$$-\frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq \mathrm{argmax}(\mathbf{u})}}^{N} r_j \geq -\frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq k}}^{N} r_j \quad .$$

Using the equations above:

$$\langle \mathbf{v}, \mathbf{w} \rangle \geq \left(\frac{\partial \log(P_{\mathrm{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_{\mathrm{argmax}(\mathbf{u})}} + \sum_{\substack{k=1 \\ j\neq \mathrm{argmax}(\mathbf{u})}}^{N} \frac{\partial \log(P_{\mathrm{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_k} \right) \times$$

$$\times \left(r_{\mathrm{argmax}(\mathbf{u})} - \frac{1}{N-1} \sum_{\substack{j=1 \\ j\neq \mathrm{argmax}(\mathbf{u})}}^{N} r_j \right) \quad .$$

Using the property (iv) from lemma 6 we see that:

$$\frac{\partial \log(P_{\mathrm{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_{\mathrm{argmax}(\mathbf{u})}} + \sum_{\substack{k=1 \\ j\neq \mathrm{argmax}(\mathbf{u})}}^{N} \frac{\partial \log(P_{\mathrm{argmax}(\mathbf{u})}(\mathbf{q}))}{\partial q_k} = 0 \quad .$$

Hence,

$$\langle \mathbf{w}, \mathbf{v} \rangle \geq 0 \quad .$$

Equality holds if $r_1 = r_2 = \cdots = r_N$ or equivalently $u_1 = u_2 = \cdots = u_N$. □

Remark 3. Note that we require a neurally compatible action selection mechanism in terms of the noiseless membrane potential, but are still able to implement the action selection directly on the realized firing rates of the neurons.

With the help of these preparations, we can prove that using the winner-nudges-all error in the last layer, the network in section 5.2.1 implements Hill-climbing on the expected reward.

Lemma 8 (*Hill-climbing property*) *Consider a setup with N action neurons corresponding to actions $a_1, a_2, \ldots, a_N$. Furthermore, consider an action-selection mechanism as in lemma 7: Let there be N random variables $u_i \in \mathbb{R}$ with an arbitrary random distribution $\mathbf{u} \sim p(\mathbf{u}; \mathbf{q})$ parametrized by the action values $\mathbf{q}$, and a strictly monotone increasing function $r : \mathbb{R} \to \mathbb{R}; x \mapsto r(x)$; such that $P_i(\mathbf{q}) = P(\mathrm{argmax}\,(\mathbf{r}) = i; \mathbf{q})$ is a neurally compatible action probability. Here, $\mathbf{q}$ is also a function of the state of the environment $\mathbf{x}$. Finally, let M be a $\mathbb{R}^{N\times N}$ matrix with elements $m_{ij} = -\frac{1}{N-1}$ $\forall i \neq j; i,j \in \{1,\ldots,N\}$, and with $m_{ii} = 1 \quad \forall i \in \{1,\ldots,N\}$.*

Then for every realization of $\mathbf{u}$, the directional derivative of the expected reward in the $\mathbf{q}$-space in the $R(a_{\mathrm{argmax}(\mathbf{r})})M\mathbf{r}$ direction is non-negative:

$$\left\langle \frac{\partial \langle R \rangle}{\partial \mathbf{q}}; R(a_{\mathrm{argmax}(\mathbf{r})}) M\mathbf{r} \right\rangle \geq 0 \quad , \tag{5.70}$$

$\langle \cdot ; \cdot \rangle$ denotes the Euclidean scalar product. Equality holds only for $u_1 = u_2 = \cdots = u_N$.

Proof. First we compute the direct policy-gradient, starting form the definition of the mean expected reward:

$$\langle R \rangle = \sum_i R(a_i) P_i(\mathbf{q}) \quad . \tag{5.71}$$

We calculate the derivative of the reward according to $\mathbf{q}$ using the familiar log-derivative trick:

$$\begin{aligned} \partial_{\mathbf{q}} \langle R \rangle &= \partial_{\mathbf{q}} \sum_i R(a_i) P_i(\mathbf{q}) = \sum_i R(a_i) \partial_{\mathbf{q}} P_i(\mathbf{q}) = \\ &= \sum_i R(a_i) \left[\partial_{\mathbf{q}} \log\left(P_i(\mathbf{q}) \right) \right] P_i(\mathbf{q}) \quad . \end{aligned} \tag{5.72}$$

Notice that by construction of the action selection mechanism that:

$$P_i(\mathbf{q}(\mathbf{x})) = \int \chi_{\operatorname{argmax}(\mathbf{r})=i}(\mathbf{r}) p(\mathbf{r}; \mathbf{q}(\mathbf{x})) \mathrm{d}\mathbf{r} \quad , \tag{5.73}$$

where $\chi_{\operatorname{argmax}(r)=i}$ is the indicator function:

$$\chi_{\operatorname{argmax}(\mathbf{r})=i}(\mathbf{r}) = \begin{cases} 1 & \text{if } \operatorname{argmax}(\mathbf{r}) = i \quad , \\ 0 & \text{otherwise} \quad . \end{cases} \tag{5.74}$$

Hence, for an arbitrary function of $f(a_i)$ we know that:

$$\sum_i f(a_i) P_i(\mathbf{q}) = \int f(a_{\operatorname{argmax}(\mathbf{r})}) p(\mathbf{r}; \mathbf{q}) \mathrm{d}\mathbf{r} \quad . \tag{5.75}$$

Applying equation (5.75) to equation (5.72), we obtain the standard policy-gradient for our setup:

$$\partial_{\mathbf{q}} \langle R \rangle = \int R(a_{\operatorname{argmax}(\mathbf{r})}) \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{r})}(\mathbf{q}))}{\partial \mathbf{q}} p(\mathbf{r}, \mathbf{q}) \mathrm{d}\mathbf{r} \quad . \tag{5.76}$$

Now we can calculate the directional derivative:

$$\begin{aligned} &\left\langle \frac{\partial \langle R \rangle}{\partial \mathbf{q}} ; R(a_{\operatorname{argmax}(\mathbf{r})}) M \mathbf{r} \right\rangle = \\ &= \left\langle \int R(a_{\operatorname{argmax}(\mathbf{r})}) \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{r})}(\mathbf{q}))}{\partial \mathbf{q}} p(\mathbf{r}, \mathbf{q}) \mathrm{d}\mathbf{r} ; R(a_{\operatorname{argmax}(\mathbf{r})}) M \mathbf{r} \right\rangle = \\ &= \int R^2(a_{\operatorname{argmax}(\mathbf{r})}) \left\langle \frac{\partial \log(P_{\operatorname{argmax}(\mathbf{r})}(\mathbf{q}))}{\partial \mathbf{q}} ; M \mathbf{r} \right\rangle p(\mathbf{r}, \mathbf{q}) \mathrm{d}\mathbf{r} \quad . \end{aligned} \tag{5.77}$$

From lemma 7, we know

$$\left\langle \frac{\partial \log(P_{\mathrm{argmax}(\mathbf{r})}(\mathbf{q}))}{\partial \mathbf{q}}; M\mathbf{r} \right\rangle \geq 0 \quad , \tag{5.78}$$

and trivially

$$R^2(a_{\mathrm{argmax}(\mathbf{r})}) \geq 0 \quad , \tag{5.79}$$

therefore

$$\left\langle \frac{\partial \langle R \rangle}{\partial \mathbf{q}}; R(a_{\mathrm{argmax}(\mathbf{r})}) M\mathbf{r} \right\rangle \geq 0 \quad . \tag{5.80}$$

And equality is only possible if $u_1 = u_2 = \cdots = u_N$, which is pathological for any probability distribution $p(\mathbf{u}; \mathbf{q})$ without special peaks. □

For the application to our reinforcement learning model we have to map and interpret the proof:

1. Everywhere we use the fact that β is small, and hence the dynamics are dominantly determined by the bottom-up input; the effect of the winner-nudges-all circuit is small and only contributes to the gradient building.

2. The $\mathbf{q}$ values of the action are the membrane potentials the action neurons would take without the additional noise to their membrane.

3. The noise to the action neurons give rise to the probability distribution $p(\mathbf{u}; \mathbf{q})$. Now it is also clear — especially with the Ornstein-Uhlenbeck process based noise mechanism — why cases where $u_1 = u_2 = \cdots = u_N$ are pathological. The noise on the single action neurons is independent from each other, the resulting action selection mechanism fulfills the defining properties of a neurally compatible action selection in definition 5. The noise could originate from the background activity of other parts of the brain. The activity of other brain areas could serve as a source of stochasticity for the Ornstein-Uhlenbeck process similarly as in the mechanism in Dold et al. [2019] and in chapter 3.

4. In lemma 8 we did not mention the sum over the dataset, and hence we proved the Hill-Climbing property for online learning. The dataset would only modify the learning by summing over the dataset $\sum_{\mathbf{x}} \mathbf{q}(\mathbf{x})$ with $\mathbf{x}$ the single samples from the dataset.

5. In lemma 8 we also did not mention the backpropagation term. It only modifies the proof by introducing the Jacobian of backpropagation that transfers the error-vector from the action neurons to the single synapses

$$\frac{\partial \langle R \rangle}{\partial \mathbf{q}} = \underbrace{\frac{\partial \mathbf{q}}{\partial W}}_{\text{backprop Jacobian}} \frac{\partial \langle R \rangle}{\partial \mathbf{q}} \quad , \tag{5.81}$$

where W contains all the synaptic weights that parametrize the function $\mathbf{q}(\mathbf{x})$. The backpropagation of errors is the result of the principle of least action theory (section 5.1.1 and theorem 1) in the time-continuous framework. Because the deep neural network is a non-linear function, the Hill-climbing property cannot be guaranteed for the synaptic updates. Still simulations show that the model learns in practical applications. This is a common problem of deep-learning models, that rigorous proofs can only be given for linear approximations, e.g in Lillicrap et al. [2016], Bernacchia et al. [2018].

5.2.3 Deep reinforcement learning in an artificial neural network and winner-nudges-all error

Unfortunately, the full simulation of the time-continuous model (section 5.2.1) is tedious and requires large computational capacity. Iterative works and parameter studies are not feasible with the full model. Hence, we develop an ANN based analogous model to make these kinds of studies feasible, even though without the time-continuous dynamics.

We base our setup on the standard policy-gradient learning with an ANN (section 5.1.3). We consider a feed-forward ANN parameterized by the synaptic weights W with $W^{(i)}$ the weight matrix projecting from layer $i-1$ to layer i. We identify the neurons in the last layer of the network with the actions and hence call them action neurons. For a given sample from the dataset $\mathbf{x}$, the membrane potential of the neurons in the last layer is denoted by $\mathbf{u}_N$. The action neurons experience Gaussian noise on their membrane potential and show an activity according to the realized noisy membrane potential:

$$\mathbf{r} = \rho\left(\mathbf{u}_N + \mathcal{N}\left(0; \sigma^2_{\text{ANN}}\right)\right) \quad , \tag{5.82}$$

where ρ is the activation function of the neurons and σ_{ANN} is the standard deviation of the Gaussian noise. Note, that compared to the time-continuous model, we changed the exploration noise from an Orstein-Uhlenbeck process to a Gaussian noise, because the ANN model does not have time a time-continuous dynamics but only consecutive iterations. Taking Gaussian noise can be thought of as taking samples from the Ornstein-Uhlenbeck process with a large enough time-separation, that the samples can be considered independent of each other.

The action j is chosen as the most active action-neuron in the given realization of the Gaussian noise $\text{argmax}_j\,(\mathbf{r})$. The plasticity is composed of a reward-maximizing (ΔW_{RM}) and a homeostatic term (ΔW_{HOM}):

$$\Delta W = \Delta W_{\text{RM}} + \Delta W_{\text{HOM}} \quad . \tag{5.83}$$

The reward maximizing term is calculated in analogy to policy-gradient learning but instead of the error-vector on the output we use the winner-nudges-all error-vector,

$$\Delta W_{\text{RM}} = \eta_{\text{RM}} \left(R - \langle R \rangle\right) \frac{\partial \mathbf{u}_N}{\partial W} M \mathbf{r} \quad , \quad . \tag{5.84}$$

Here, M denotes the usual WNA matrix with $M \in \mathbb{R}^{N\times N}$ with $m_{ij} = -\frac{1}{N-1}\ \forall i \neq j; i,j \in \{1,\dots,N\}$, and $m_{ii} = 1 \quad \forall i \in \{1,\dots,N\}$; R the obtained reward, $\langle R\rangle$ the expected reward under the current policy and η_{RM} the learning rate. $(R - \langle R\rangle)$ is the familiar reward-prediction-error. Finally, $\frac{\partial \mathbf{u}_N}{\partial W}$ is the Jacobian connecting the error-vector on the membrane potential of the action neurons and the synaptic weights of the model. The homeostatic terms mimic the extreme value homeostases and the midpoint homeostases introduced in section 5.2.1. They are modulated by the absolute value of the reward-prediction-error:

$$\begin{aligned}\Delta W^{(i)}_{\text{HOM}} = |R - \langle R\rangle| \, (\eta_{\text{e-hom}} \left(\max\left(0; u_{\text{ulow}} - u_i\right) - \max\left(0; u_i - u_{\text{uhigh}}\right)\right) r^T_{i-1} + \\ + \eta_{\text{m-hom}} \left(u_{\text{mid}} - u_i\right) r^T_{i-1}) \quad ,\end{aligned} \tag{5.85}$$

where the learning rates $(\eta_{\text{e-hom}}; \eta_{\text{m-hom}})$ and the extreme and midpoint membrane potentials $(u_{\text{ulow}}; u_{\text{mid}}; u_{\text{uhigh}})$ play the same role as in the time-continuous model. u_i is the membrane potential in layer i and r_i is the activity of layer i. The expected reward $\langle R\rangle$ is in practice approximated with a moving average as in the time-continuous model:

$$\langle R\rangle_n = \langle R\rangle_{n-1} (1-\gamma) + \gamma R_{n-1} \quad , \tag{5.86}$$

where the index n denotes the values in the $n-$th iteration. The parameter γ regulates how sensitively the moving average reacts to new values. The algorithm is summarized in algorithm 5.2.

Algorithm 5.2: ANN based model with winner-nudges-all error. The algorithm is inspired by the standard policy-gradient learning with online learning, that is learning on every sample. However, the error-vector is replaced by the winner-nudges-all error and the plasticity is amended with homeostases.

```
Data: Feed-forward network parametrized by $\boldsymbol{W}$; dataset with entries $\mathbf{x}$
Initialize $\langle R\rangle_0 = 0$ ;
Initialize $\boldsymbol{W}_0$;
for $n = 1$ to max number of iterations do
    obtain random data sample $\mathbf{x}$ ;
    calculate corresponding $\mathbf{u}_N$ ;
    get sample from $\mathrm{r}_N = \rho(\mathbf{u}_N + \mathcal{N}(0; \sigma^2_{\text{ANN}}))$ ;
    take action argmax $(\mathbf{r}) \rightarrow$ observe reward $R_n$ ;
    update weights $\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \Delta\boldsymbol{W}$ from equation (5.83) ;
    update $\langle R\rangle_n = \langle R\rangle_{n-1}\gamma + (1-\gamma) R_n$ ;
end
```

5.3 Simulations and results

We tested the model in several learning setups. The tedious numerics and hence the slow simulation prohibit the tackling of large, more state-of-the-art datasets, and we restrict ourselves to a reduced version of the classic MNIST dataset [LeCun et al., 1998]. We test the framework by classifying the dataset in the reinforcement learning framework. We focus on the technical details of learning; we verify that backpropagation actually takes place in the model, we test the robustness to delayed rewards and to fixed-pattern noise in the winner-nudges-all circuit.

5.3.1 Setup of the simulations

In the following simulations, we apply the full time-continuous network, and where it is not practical its ANN-based analog, to a classification task. Classification is neither the ultimate goal of reinforcement learning algorithms nor it is efficiently solvable with reinforcement learning algorithms, but the simplicity of the task makes it easier to study and experiment with.

We use a feed-forward network structure, where the first layer serves as the input layer and in the last (action) layer we associate the neurons with the classes, as one would do in a one-hot coding setting. The input is realized as a current to the neurons of the first layer with the introduced ramp-up and ramp-down transitions (figure 5.5). The action is read-out as the action-neuron with the maximal low-pass filtered activity just before the beginning of the ramp-down phase. If the predicted class is correct, a reward $R = 1$ is given and $R = -1$ if the predicted class is false.

We applied the model to a reduced version of the MNIST hand-written digits dataset [LeCun et al., 1998]. A reduction was necessary due to the costly numerical simulation of the model, and such that we are able to carry out several different experiments in feasible time. We reduced the size of images from the original 28×28 size to 20×20 by cropping a 4 pixel wide frame (figure 5.8 A), which is an empty padding for most of the images. Finally, we only kept the classes 0,2 and 4 from the ten possible classes but preserved all the examples in the classes both in the training and the test data. The used dataset is available via the repository containing the source code (appendix A.3.3).

Until now the activation function of the neurons was arbitrary. The proof of reward maximization (lemma 8) only requires a strict monotonously increasing function. In the simulations, we used the softplus function (figure 5.8 B), which can be seen as a differentiable alternative of the widely-used rectified linear unit:

$$\rho\left(u\right) = s_{\text{sp}} d \ln\left(\exp\left(\frac{u}{d}\right) + 1\right) \quad , \tag{5.87}$$

with d the width and s_{sp} the slope of the activation function.

The weights of the single matrices are initialized following He et al. [2015]. The initial weights follow a random distribution with zero mean

$$\boldsymbol{W}_i^{\text{init}} \propto \mathcal{N}\left(0; \frac{2}{K_{i-1}}\right) \quad , \tag{5.88}$$

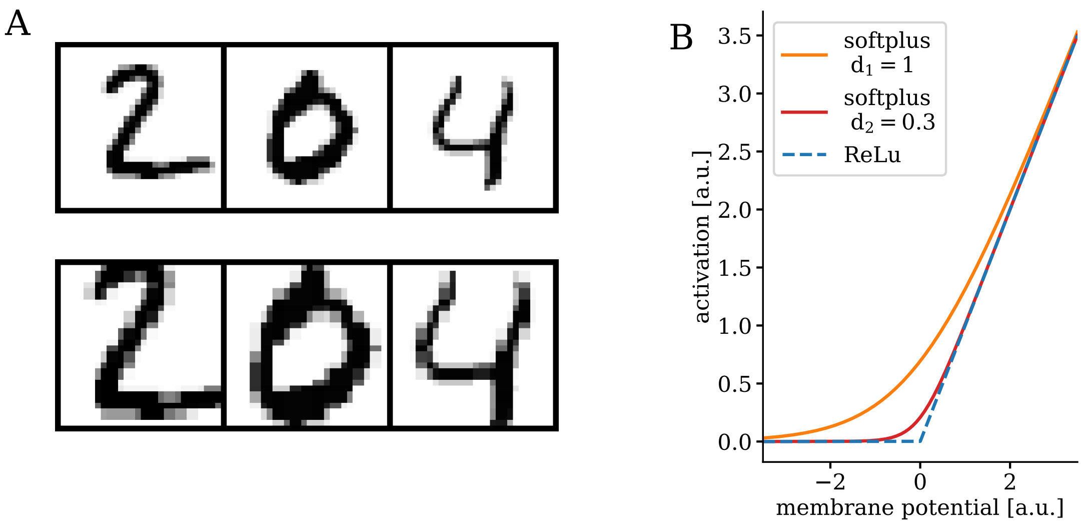


Figure 5.8: The used dataset and the activation function. (A) In simulations we used the reduced version of the MNIST hand-written digits dataset [LeCun et al., 1998]. Samples from the original data are shown in the upper panel and from the reduced images in the lower panel. **(B)** The used softplus activation function (equation (5.87)) can be seen as an everywhere-differentiable alternative of the rectified linear unit (ReLu) activation function, $ReLu(x) = \max(0, x)$. In the plot we show softplus functions with widths of $d_1 = 1$ and $d_2 = 0.3$. In the limit of $d \to 0$ the softplus converges to the ReLu function.

where K_{i-1} is the number of neurons in the $(i-1)$-th layer or in another view the fan-in (number of pre-synaptic partners) of the neurons in the i-th layer. The standard deviation of $\sqrt{\frac{2}{K_{i-1}}}$ makes sure that the expected amplitude of the input signal (image) is preserved throughout the network. We used this initialization to accelerate the learning. We emphasize that apart from the initialization technique, we explicitly refrain from further known tools from machine learning to enhance learning, such as weight regularization, adapting learning rate or batch learning. It is not clear how these techniques could be realized in the brain, hence they might dominate our results while not being biologically plausible.

The learning happens in online-learning fashion, corresponding to the dynamic equations and the plasticity rules. The input images are presented sequentially, drawn randomly form the training set. Importantly, we do not reset the state of the network (membrane potentials, eligibility traces) between the images, instead we introduce the continuous ramp-up and ramp-down of the inputs. The time of the ramps is smaller than the membrane-time constant, hence it is not the same as waiting long enough for the system to relax to a neutral state. Plasticity is applied upon arrival of the reward immediately. This leads to small discontinuities due to the instantaneous weight updates, but according to our experience it does not disrupt the dynamics. Due to the nature of the classification task, a slightly smeared-out reward input would most likely show the same model behavior.

For testing, we turn off the winner-nudges-all network and the exploration noise on the action neurons. The classification is implemented in the same sequential time-continuous way as in the case of learning, but the weights are kept constant. We measure the development of the classification rate for a reduced test set (200 images per class) and measure the classification rate on the full test set once at the end of the learning.

The software was implemented using the TensorFlow machine learning software tool [Abadi et al., 2015]. The source-code is available upon request from the author, samples from the implementation are shown in (appendix A.3.3). The used parameters are given in table 5.1 for the time-continuous model, in table 5.2 for the ANN-based model with winner-nudges-all error and in table 5.3 for the standard policy gradient algorithm. The membrane time-constant is set to $\tau = 10\,\text{ms}$ neurons to put the model into the time-scales of biological systems. $\tau = 10\,\text{ms}$ is a typical membrane time constant in biological neurons [Dayan and Abbott, 2001]. We used the same parameters in all the experiments unless mentioned differently in the corresponding text. Note that the nudging parameter is only one order of magnitude smaller then the inference direction dynamics. The reason for this is again the tedious numerics: the error nudging has to be larger than the errors introduced by the numerical precision. At the same time, we need a possibly large timestep Δt such that the simulation-time stays feasibly long. An optimum between these two requirements was found with $\Delta t = 0.1$ and $\beta = 0.1$. Second, the two homeostatic processes push the network towards positive membrane potentials. The reason lies in the error-generation mechanism: The winner-nudges-all circuit requires non-zero and diverse activity in the action neurons to create a sensible error on the actions (see lemma 8). With the used softplus activation functions we require positive membrane potentials to achieve such an activity.

Symbol	*Value*	*Description*
-	(400, 400, 3)	layers
τ	10 ms	membrane time-constant
d	0.3	softplus width
s_{sp}	1	softplus slope
β	0.1	nudging parameter
τ_{elig}	40 ms	eligibility time-constant
η	10^{-3}	learning rate
$\eta_{\text{e-hom}}$	3×10^{-3}	e-hom learning rate
u_{ulow}	-1.25	lower limit potential
u_{uhigh}	1.25	upper limit potential
$\eta_{\text{m-hom}}$	5×10^{-3}	m-hom learning rate
u_{mid}	1	homeostases middle potential
T_{ramp}	5 ms	input ramping time
T_{image}	30 ms	image presentation time
γ	0.9	moving average parameter
σ_{OU}	0.2	standard deviation; exploration noise
τ_{OU}	30 ms	exploration noise; autocorrelation time
Δt	1 ms	timestep in the simulation

Table 5.1: Parameters in the time-continuous model. In each experiment we used the same parameters unless stated otherwise.

5.3.2 Experiments with immediate reward

In this section, we present the result of learning with immediate reward. The reward was given immediately after the network decided for a class during the learning phase. Note that we kept the eligibility time-constat τ_{elig} at a finite value such that the results are compatible with those with delayed reward. Here the learning would work similarly without eligibility trace $\tau_{\mathrm{elig}} \to 0$.

Learning with the full model

In the first experiment, we learned the reduced MNIST dataset with policy gradient model, the abstract winner-nudges-all model and with the time-continuous model. In this setup the reward arrived immediately after the decision of the model and we applied the plasticity at the arrival of the reward. All three models were able to learn the task and reached a similar performance (figure 5.9). On the full test set the policy gradient model reached an error rate of $6.51^{+0.12}_{-0.24}\%$, the abstract model an error rate of $3.96^{+1.4}_{-0.68}\%$ and the time-continuous model an error rate of $6.0^{+1.7}_{-2.0}\%$ with median and interquartile values. Chance level is at an error rate of 66.66%.

The results are similar, however we can only qualitatively compare the values, because the meta-parameters were not fine-tuned — e.g. via meta-parameter learning as in chapter 4 — to the problem. For example, the abstract winner-nudges-all

Symbol	*Value*	*Description*
-	(400, 400, 3)	layers
d	0.3	softplus width
s_{sp}	1	softplus slope
σ_{ANN}	0.2	exploration noise, standard deviation
η	10^{-5}	learning rate
$\eta_{\text{e-hom}}$	10^{-5}	e-hom learning rate
u_{ulow}	-1.5	lower limit potential
u_{uhigh}	6.5	upper limit potential
$\eta_{\text{m-hom}}$	10^{-5}	m-hom learning rate
u_{mid}	4	homeostases middle potential
γ	0.9	moving average parameter

Table 5.2: Parameters in the artificial neural network based model with winner-nudges-all error. In each experiment we used the same parameters unless stated otherwise.

Symbol	*Value*	*Description*
-	(400, 400, 3)	layers
d	0.3	softplus width
s_{sp}	1	softplus slope
η	10^{-5}	learning rate
γ	0.9	moving average parameter

Table 5.3: Parameters in the policy gradient model with an artificial neural network. The parameters of the model described in section 5.1.3. In each experiment we used the same parameters unless stated otherwise. The model has less parameters than the other two due to its simplicity.

model shows sings of overfitting because the test error rate starts to increase after approximately 7×10^4 iterations while the mean reward during learning still increases. The time-intensive simulation of the time-continuous model prohibits meta-parameter learning. Still, the results show that the time-continuous model can learn the task and shows similar behavior as similar reinforcement learning models do.

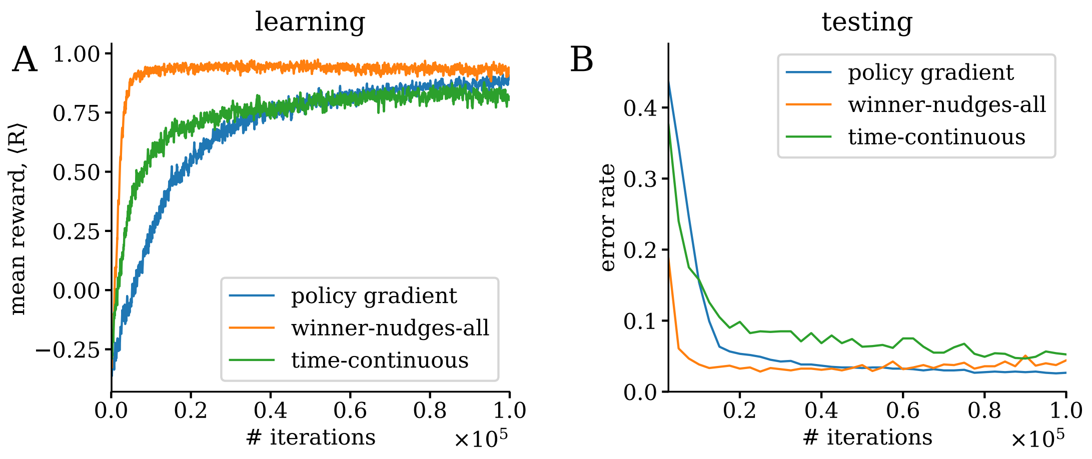


Figure 5.9: Learning results with immediate reward. The plot shows the learning results in terms of **(A)** median reward and **(B)** error ratio on the small test set for the policy gradient model, the winner-nudges-all abstract model and the time-continuous model. The mean reward is plotted with a moving average of 200 iterations to reduce the variance for the sake of visualization. The results are shown as the median over 10 repetitions for each model. All three models learn with similar performance. Note that the abstract winner-nudges-all model shows signs of overfitting because the test-error increases after approximately 7×10^4 iterations.

The rationale of the midpoint homeostases

We show the effect of the midpoint homeostases in two single-shot experiments with identical parameters (including the random seed) up to the midpoint homeostases in the abstract winner-nudges-all model (figure 5.10).

Learning is possible in both cases, although it is somewhat faster with midpoint homeostases. Without it, the network manages to learn early two out of the three classes, and the observed reward on these two classes is high. In the third class, the network only seldom chooses the correct class and hence the learning is slow. With homeostases, exploration is encouraged, because it pulls the action probabilities closer to each other. In the plotted example, this means that the third class lags behind only for a shorter time, and learning can continue.

The apparent effect of the homeostases depends on the learning task, on the initial network parameters and on pure chance during learning. We expect that the midpoint homeostases will become more important in case of larger action space (e.g. more classes), where the exploration has to cover larger areas to find the correct action.

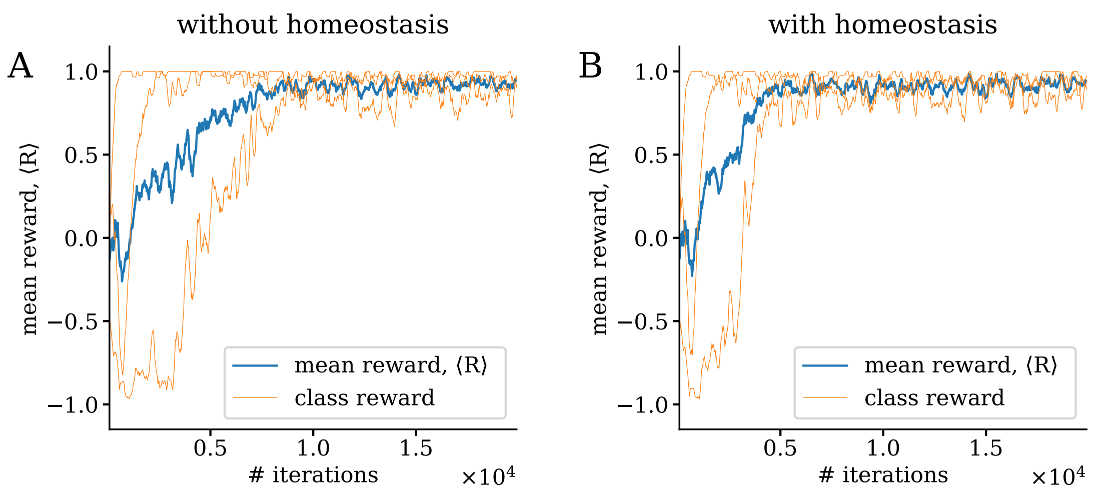


Figure 5.10: The effect of the midpoint homeostases. We show the effect of the midpoint homeostases on the early phase of learning in two single-shot experiments. The expected mean reward (blue) and the expected class-wise rewards (orange) are shown as a function of the iteration number. The three orange curves show the predicted reward from the three distinct classes. The two experiments are identical (including the random seed) up to the homeostatic learning rate: **(A)** $\eta_{\text{m-hom}} = 0$ without homeostases and **(B)** $\eta_{\text{m-hom}} = 10^{-6}$ with homeostases. Without homeostases, one of the classes is learned much slower than the other two. Midpoint homeostases encourages the exploration in the not-yet-learned classes. The experiments were made with the abstract winner-nudges-all model.

Verifying the backpropagation property

In order to verify the backpropagation property of the model, we set up an experiment, where the plasticity is turned off from the hidden layer to the action layer. Hence, learning only takes places in the synapses between the input layer and the hidden layer. We used the time-continuous model for the experiments and all other parameters were left the same as in table 5.1.

The model is still able to learn the dataset (figure 5.11), and it reaches a final error ratio of $3.27^{+1.4}_{-0.10}\%$ on the full test set with median and interquartile values. It is surprisingly lower than the final error ratio of $6.51^{+0.12}_{-0.24}\%$ with the full model. Besides, the learning curve is smoother. The reason is probably the missing meta-parameter tuning and the fact that the dataset is simple. Hence, the frozen read-out weights impose an implicit regularization on the model, which is by chance beneficial. In general for complex problems, we expect that the frozen read-out weights should deteriorate the performance. The results verify that error is meaningfully propagated back to the lower layer of the network.

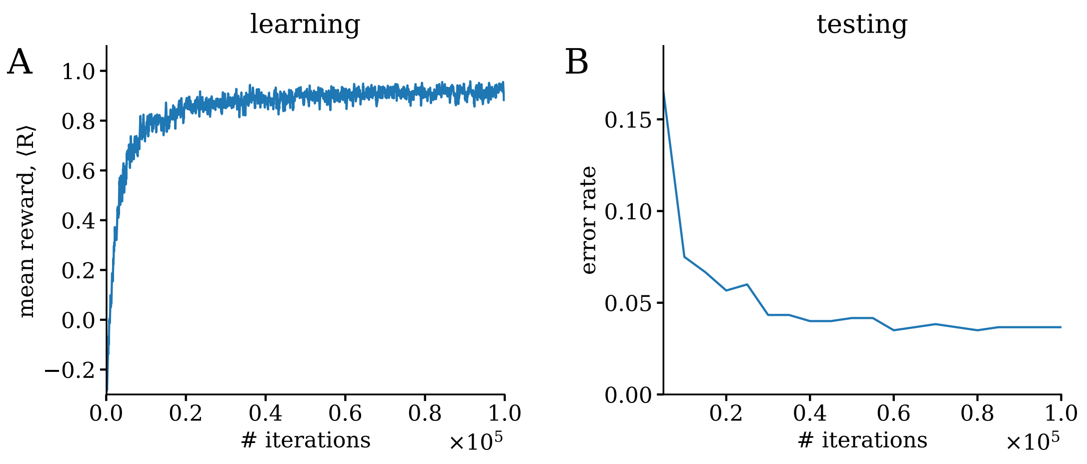


Figure 5.11: Learning with frozen read-out weights. We show **(A)** the median reward during learning and **(B)** the error rate on the small test set with frozen output weights. In this experiment, the plasticity is turned off for the synapses from the hidden layer to the output layer in order to verify that backpropagation takes place. The model is able to learn the dataset similarly as with plasticity at all synapses. The experiment was made in the time-continuous model, and we show the median values over 3 repetitions.

5.3.3 Robustness of the learning to disturbances

After having verified the learning capabilities of the setup, we now study its robustness. First, we study how the learning is affected if the reward is not immediate but it arrives either with a fixed or with a randomly distributed delay. In biological situations, it is rather the exception than the rule that the reward is immediate. Although the time-continuous model does not contain any elaborate mechanism to learn the rewards over long chains of actions, it still displays a certain amount of robustness due to the eligibility traces.

Second, inspired by the two other projects in neuromorphic computing (chapters 3 and 4), we study the effect of fixed-pattern noise on the model. Fixed-pattern noise, or in the biological terminology heterogeneous neuro-synaptic parameters, is clearly present in the nervous system: neural circuits with different parameters have observed to fulfill the same task [Taylor et al., 2006, Marder and Goaillard, 2006]. Therefore, it is an obvious requirement for a model of neural circuits to be robust against this kind of disturbance. We only consider the effect of fixed-pattern noise on the WNA circuit, because all the other synaptic parameters are learned during training.

Robustness to delayed reward with fixed time-delay

In this experiment, the reward is delayed with a fix time delay that matches the presentation time of a single image $t_{\text{delay}} = T_{\text{image}} = 30\,\text{ms}$. Otherwise, the used parameters are identical to the ones in the other simulations.

At the time of the arrival of the reward, the eligibility trace contains a combination of the error-vector from the current action and the error-vector one iteration before. But only the error-vector from one iteration before corresponds causally to the obtained reward. The model learns slower and reaches higher error rates than learning with immediate reward (figure 5.12). This is expected, because only a fraction of the eligibility trace contains correct information about the action causing the reward. A much larger part carries only noise caused by the following action. Still, learning is possible, and the network performs well above chance level with a final test error of $11.29^{+0.28}_{-0.38}\%$ on the full test set with median and interquartile values over 3 repetitions. Furthermore, learning is slower than with immediate reward; further learning would probably lower the achieved error ratio.

Robustness to delayed reward with scrambled delay

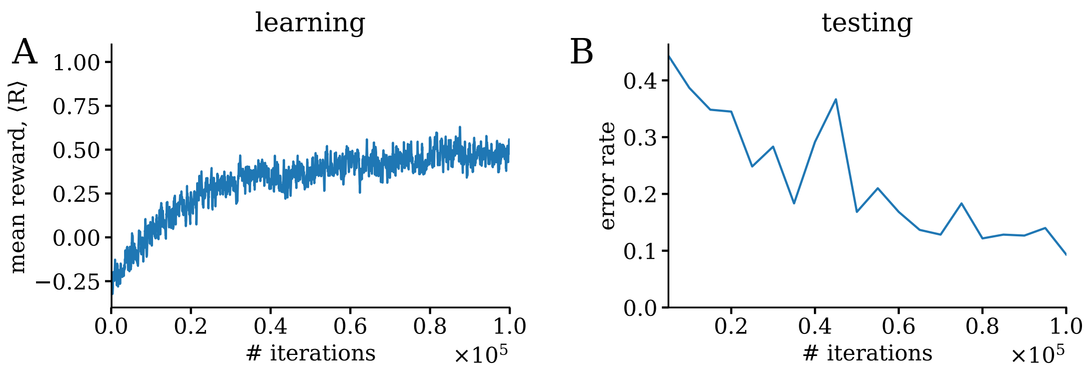
In the this experiment we study the effect of "scrambled" time-delay in the reward, following similar studies in Friedrich et al. [2011]. The reward is again delayed compared to the time of decision, but the delay is not fixed but distributed according to a Gamma probability distribution $p_\mathrm{d}(t_\mathrm{delay}) = \Gamma(2, 30\,\mathrm{ms}/2)$ with order 2 and an expected time delay $\langle t_\mathrm{delay} \rangle = 30\,\mathrm{ms}$. The expected delay is chosen to be the same as in the fixed time-delay experiment (figure 5.13) and such that it matches the presentation time of one image $\langle t_\mathrm{delay} \rangle = T_\mathrm{image}$. Apart from the delayed reward, other parameters are the same as in table 5.1.

The results are similar as with fixed delay (figure 5.13): the learning is possible but slower than with immediate reward, which is expected. The model reaches an error ratio of $10.25^{+0.83}_{-0.23}\%$ on the full test set, compatible with the results in case of the fixed reward delay. Both the expected reward and the test error ratio are still improving at the end of the simulation, suggesting that more iteration would be beneficial for the model.

The robustness to delayed reward is not to be confused with learning with sparse reward, when reward is only given after a long sequence of actions, for example after ending a game of chess. In this model, the robustness is purely based on direct correlation between the tail of the eligibility trace and the obtained reward. We can only expect a robustness on the time-scale of the eligibility trace. For learning with sparse rewards, we need more elaborate mechanisms.

Robustness to fixed-pattern noise in the winner-nudges-all circuit

Until now, we prescribed the connectivity among the action neurons to build the WNA connectivity. In nature this has to be developed at some point during the development of the specimen. It is highly unlikely that the connectivity will perfectly match the desired matrix, hence the robustness to heterogeneity in the WNA network is of importance.



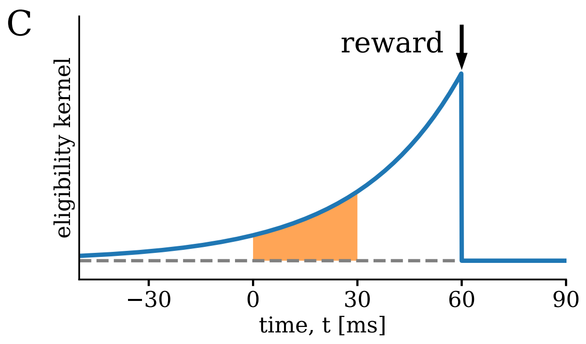


Figure 5.12: Learning with delayed reward with fixed time-delay. We show **(A)** the median reward during learning and **(B)** the error rate on the small test set with fixed time-delay. The reward arrives exactly one iteration after the decision of the network $t_{\mathrm{delay}} = T_{\mathrm{image}} = 30\,\mathrm{ms}$. The model learns much slower than the one with immediate reward, but learning is possible and the model performs above chance level. The experiment was made in the time-continuous model, and we show the median values over 3 repetitions. **(C)** If a reward signal arrives at, for example 60 ms (arrow), then a part of the eligibility trace still carries information about the action that lead to the reward (orange section from 0 ms to 30 ms). Learning is possible but slower than with immediate reward because only a smaller part of the eligibility trace corresponds to the rewarded action.

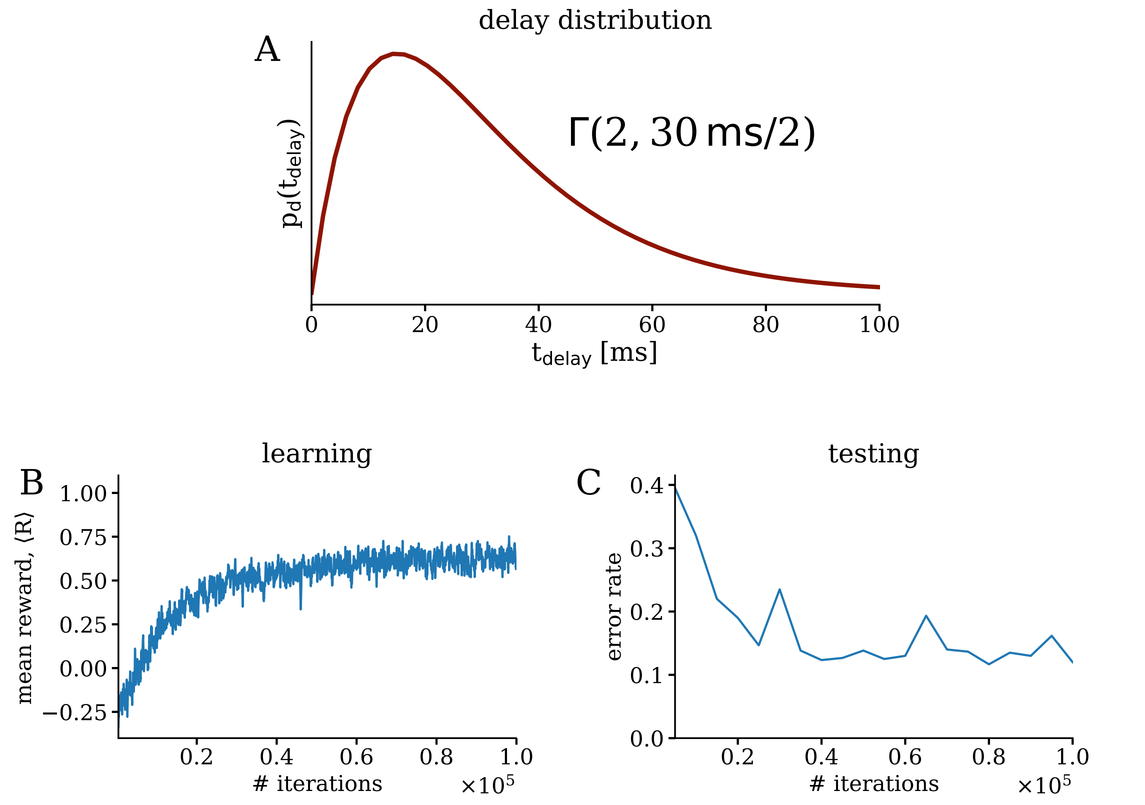


Figure 5.13: Learning with delayed reward with random time-delay. (A) The reward is delayed by a random time-delay which is drawn each time from the probability distribution $p_{\mathrm{d}}(t_{\mathrm{delay}}) = \Gamma(2, 30\,\mathrm{ms}/2)$. The expected value of the reward delay matches the presentation time of the single images $\langle t_{\mathrm{delay}} \rangle = T_{\mathrm{image}} = 30\,\mathrm{ms}$. We show **(B)** the median reward during learning and **(C)** the error rate on the small test set with random time-delay. The results are similar as with fixed time-delay (figure 5.12). The model learns much slower than the one with immediate reward, but learning is possible and the model performs above chance level. Interestingly, the learning is faster than with fixed reward delay. The experiment was made in the time-continuous model, and we show the median values over 3 repetitions.

To test the robustness, we modulated the synaptic connections with multiplicative fixed-pattern noise:

$$w_{ij}^{\mathrm{WNA}} \rightarrow w_{ij}^{\mathrm{WNA}} \times \max\left(0, \mathcal{N}\left(1, \sigma_{\mathrm{FP}}^2\right)\right) \quad , \tag{5.89}$$

where w_{WNA} represents a synaptic weight in the winner nudges-all circuit, $\mathcal{N}$ is the normal distribution and σ_{FP} is the standard deviation of the fixed-pattern noise. We use multiplicative noise, because the inhibitory connections are much weaker than the excitatory ones. We assume that fixed-pattern noise will not change this relation. Second, we prohibit the sign change of of the weights w_{ij} due to fixed-pattern noise. It is unlikely that synapses change their sign as their weight varies. In biological systems, neurons are typically either excitatory or inhibitory in their effect, according to Dale's principle (section 2.2.1). In neuromorphic hardware, it is possible that synapses erroneously change their sing, but this is rather due to digital bit-flip errors and not due to high fixed-pattern noise.

The learning results with the WNA-based model show that learning is only slightly affected up to a standard deviation of the fixed-pattern noise of $\sigma_{\mathrm{FP}} = 0.2$ (figure 5.14 A). For higher σ_{FP}, we find that there are several outliers; for some samples of the fixed-pattern noise learning becomes impossible, however if the learning is possible than the model reaches similar error ratios as with ideal WNA synapses. Simulations with the time-continuous model verify this finding (figure 5.14 B): There is only a minor difference in the final error ratio between the cases with and without fixed-pattern noise. We can explain the lack of the outliers by the smaller number of experiments with the time-continuous model. Hence, we conclude that the model is robust up to a fixed-pattern noise of approximately 20%.

To understand the mechanism of this robustness, we take a look at the phase-space representation of the learning dynamics in figure 5.15, where we show the phase space of learning in the $\mathbf{q}$ space in two dimensions. Note, the difference to figure 5.7; here we show the direction of learning with reward modulation assuming (without loss of generality) that action 1 is the correct action ($q_1 > q_2$ is desired). With a perfect WNA circuit, action 1 is strengthened and action 2 is weakened on the entire space (figure 5.15 A). Adding fixed-pattern noise the WNA matrix distorts the streamlines (figure 5.15 B). The direction of the desired learning is disrupted along the line $q_1 = q_2$, and this region stalls the learning. The introduction of exploration noise restores the learning. Now at each point in the $\mathbf{q}$-space, we can calculate an expected update direction due to the exploration noise. In our Gaussian exploration scheme (section 5.2.3 and equation (5.52)), this corresponds to filtering the streamlines with fixed-pattern noise via a Gaussian kernel (figure 5.15 C). The learning can proceed as long as the exploration noise is large enough to help the system overcome the stalling region.

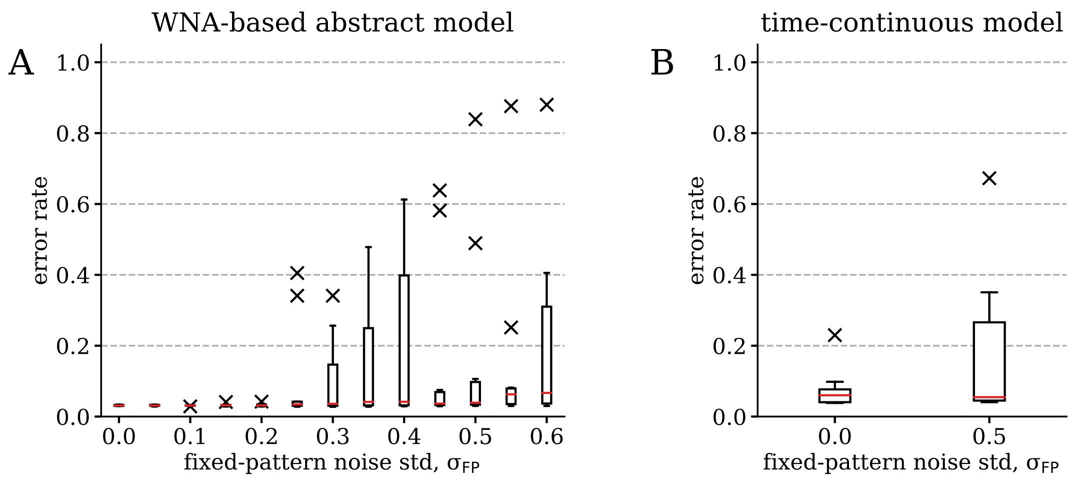


Figure 5.14: Learning with fixed pattern noise in the winner-nudges-all circuit. (A) The final error ratio is shown as a function of the standard deviation of the multiplicative noise on the WNA circuit in the abstract model with WNA. **(B)** The final error ratio stays robust in the time-continuous model as well. The data is plotted both in (A) and (B) following the traditional box-and-whiskers scheme: the orange line represents the median, the box represents the interquartile range, the whiskers represent the full data range and the $\times$ represent the far outliers. Each boxplot contains 10 repetitions of the same experiment.

5.4 Alternative formulations of deep reinforcement learning

Reinforcement learning can be achieved in the framework of the principle of least action with other mechanisms as well. Intuitively, we imagine that the learning is made of two components: 1) the backpropagation mechanism through the network and 2) the error-vector generation mechanism in the output layer. The latter corresponds to the definition of a cost function, where, in principle, any cost function can be given. Supervised learning and reinforcement learning with the WNA network are two possible implementations of this more general framework. Here, we briefly introduce two alternative formulations of reinforcement learning, discuss their relation to the WNA model and compare the advantages and disadvantages of the three models.

5.4.1 Reinforcement learning with direct node perturbation

In the first — more simple – alternative, we use the idea of direct node perturbation [Williams, 1992, Werfel et al., 2004, Fiete and Seung, 2006]. In this framework, we refrain from elaborate synaptic connections in the action layer. Instead, noise

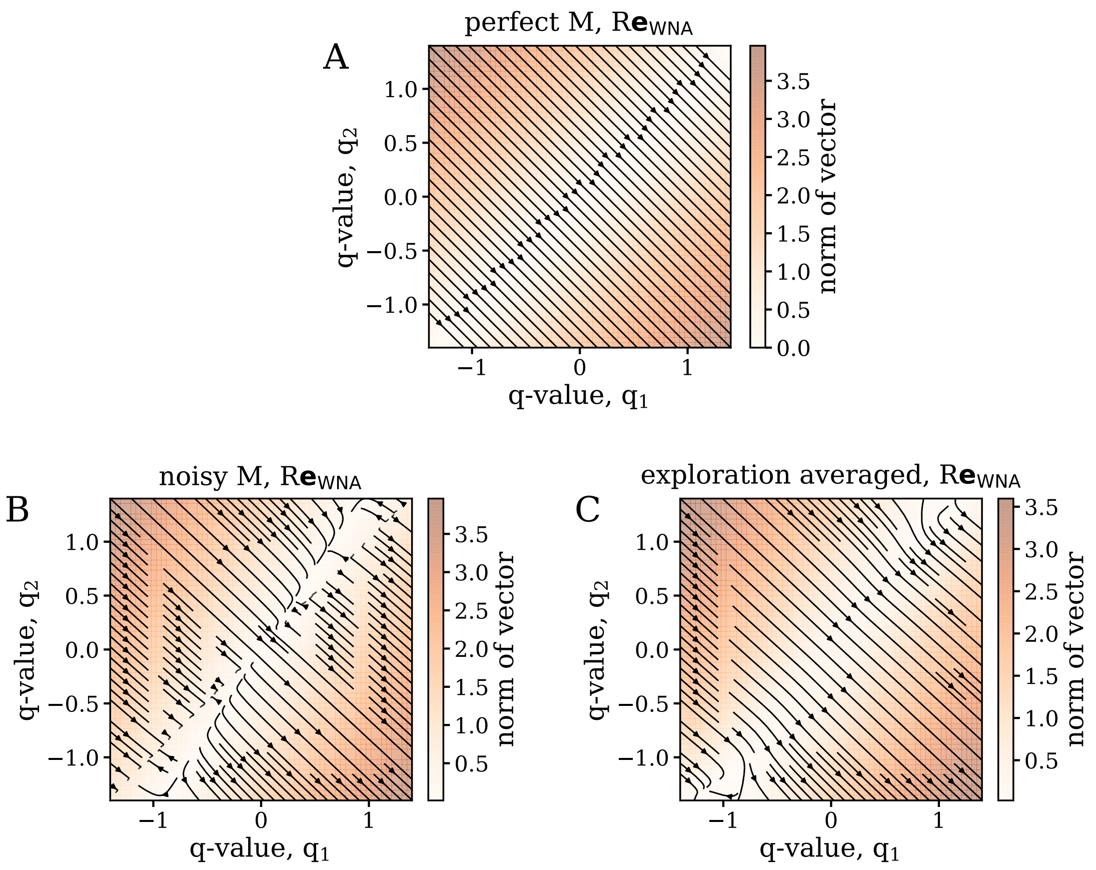


Figure 5.15: Mechanism of robustness to fixed-pattern noise. (A) Phase-plot of learning assuming that the action q_1 is the correct action. In any point of the phase-space, action 1 is strengthened. Mind the difference to the phase plots in figure 5.7, where the error-vector is shown without the reward modulation. **(B)** Phase-plot for a sample from fixed-pattern noise on the WNA matrix. Fixed-pattern noise disrupts the logic of learning: There is now a region around the $q_1 = q_2$ line, where learning progresses into the wrong direction. This region stalls the learning. **(C)** The learning is restored by introducing exploration noise. The expected direction of the update is given by filtering the phase-space plot with a Gaussian kernel, assuming that we use the Gaussian exploration introduced in section 5.2.3 and equation (5.52). Plots are shown in the **q** space for the sake of clarity, but they would look qualitatively similar in the space of firing rates.

on the action neurons is responsible for both the exploration and the gradient building procedure. We use the full energy function:

$$L = E + \beta C_{\mathrm{NP}} = \sum_{i=1}^{N} \frac{1}{2} \left\| u_i - W_i \bar{r}_{i-1} \right\|^2 + \frac{\beta}{2} \left\| \bar{\xi} \right\|^2 , \tag{5.90}$$

with $\bar{\xi}$ the low-pass-filtered noise on the action neurons. The noise follows the same logic as in the WNA-based model (equation (5.52)). The principle of least action framework leads to the dynamics:

$$\begin{aligned} \tau \dot{u}_i &= W_i r_{i-1} - u_i + e_i \quad , \\ r_i &= \bar{r}_i + \tau \dot{\bar{r}}_i \; ; \quad e_i = \bar{e}_i + \tau \dot{\bar{e}}_i \quad , \\ \bar{e}_i &= \bar{r}_i \odot W_{i+1}^T \left(u_{i+1} - W_{i+1} \bar{r}_i \right) \quad , \\ \bar{e}_N &= \beta \bar{\xi} \quad , \end{aligned} \tag{5.91}$$

with all the quantities matching the ones introduced in section 5.2.1 apart from the noise. In the WNA-based model, the noise arrived at the dendrite of the action neurons, because it only played a role in the exploration but not in the error-vector mechanism. Here, the noise arrives at the soma of the action neurons, because now it is responsible for both the exploration and the error-vector generation.

To test the model, we implemented the corresponding analogy using artificial neurons. The artificial model follows the same logic and algorithm as the WNA-based artificial model, but the synaptic plasticity rule in equation (5.84) is modified as:

$$\Delta W_{\mathrm{RM}}^{(\mathrm{DN})} = \eta_{\mathrm{RM}} \left(R - \langle R \rangle \right) \frac{\partial \mathbf{u}_N}{\partial \boldsymbol{W}} \xi^{(\mathrm{ANN})} \quad , \tag{5.92}$$

where $\xi^{(\mathrm{ANN})}$ is a sample from the noise distribution $\mathcal{N}(0; \sigma_{\mathrm{ANN}}^2)$. The model is summarized in algorithm 5.3, and the used parameters are shown in table 5.4. We did not perform any meta-parameter optimization on the model.

The model with direct node perturbation is capable of learning the task (figure 5.16). It is slower than the WNA-based model in terms of learning progress per iteration. Note that the direct node perturbation model was trained much longer than its WNA-based counterpart. This is the expected behavior, because direct node perturbation explores the action-space without any knowledge about the structure of the chosen action. The WNA-based model gets additional information based on the generated error-vector. However, it is not clear how much of the observed difference stems from this inherent characteristics and how much from the lack of meta-parameter learning. After learning, this model achieves an error ratio of $3.01^{+0.38}_{-0.03}\%$ on the test set.

5.4.2 Reinforcement learning using preserved synaptic connections and node perturbation

The second alternative is to assume the same synaptic connections throughout the whole network, including the cost function; hence the name preserved synaptic

Algorithm 5.3: ANN based model with direct node perturbation. The algorithm is inspired by the standard policy gradient learning with online learning, that is learning on every sample. However, the error-vector is replaced by the direct node perturbation error and the plasticity is amended with homeostases.

```
Data: Feed-forward network parametrized by $\boldsymbol{W}$; dataset with entries $\mathbf{x}$
Initialize $\langle R \rangle_0 = 0$ ;
Initialize $\boldsymbol{W}_0$;
for $n = 1$ to max number of iterations do
    obtain random data sample $\mathbf{x}$ ;
    calculate corresponding $\mathbf{u}_N$ ;
    get sample from $\xi^{(\mathrm{ANN})} \sim \mathcal{N}(0; \sigma^2_{\mathrm{ANN}})$ ;
    calculate $\mathrm{r}_N = \rho(\mathbf{u}_N + \xi^{(\mathrm{ANN})})$ ;
    take action argmax $(\mathbf{r}) \to$ observe reward $R_n$ ;
    update weights $\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \Delta \boldsymbol{W}$ using equation (5.83) and
      equation (5.92) ;
    update $\langle R \rangle_n = \langle R \rangle_{n-1} \gamma + (1 - \gamma) R_n$ ;
end
```

Symbol	*Value*	*Description*
-	(400, 400, 3)	layers
d	0.3	softplus width
s_{sp}	1	softplus slope
σ_{ANN}	0.1	exploration noise, standard deviation
η	3×10^{-5}	learning rate
$\eta_{\text{e-hom}}$	10^{-3}	e-hom learning rate
u_{ulow}	0.0	lower limit potential
u_{uhigh}	6.0	upper limit potential
$\eta_{\text{m-hom}}$	3×10^{-7}	m-hom learning rate
u_{mid}	3	homeostases middle potential
γ	0.97	moving average parameter

Table 5.4: Parameters in the ANN-based model with direct node perturbation. Table of parameters used in simulations shown in figure 5.16.

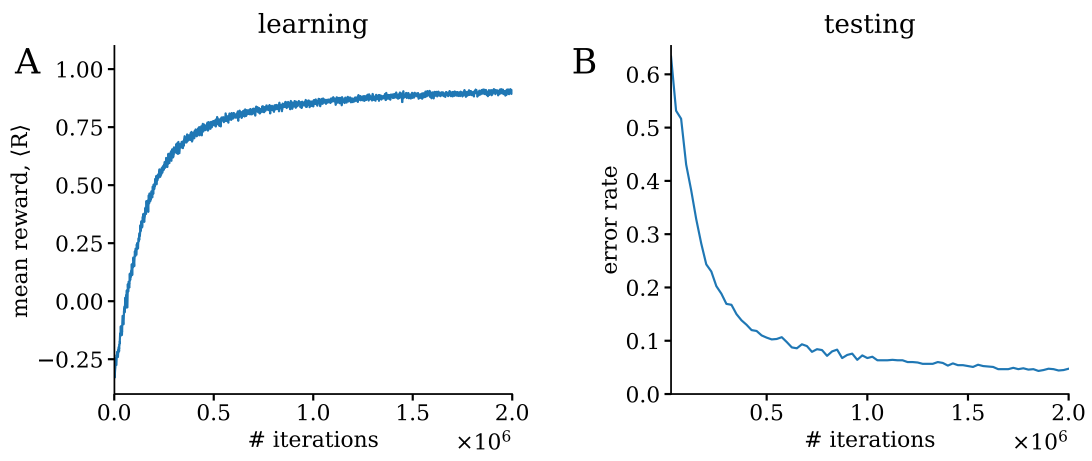


Figure 5.16: Learning results with the direct node perturbation model. The plot shows the learning results in terms of **(A)** median reward and **(B)** error ratio on the reduced test set. The median reward is plotted with a moving average of 200 iterations to reduce the variance for the sake of visualization. Note that in these experiments, we used 10 times longer training than in the other experiments. The results are shown as the median over 10 repetitions.

connection model (PrSC-Model). We assume that there is the same microcircuit structure between the action neurons as between the layers of the feed-forward part of the network. This implies the total energy function:

$$L = E + \beta C_{\text{PrSC}} = \sum_{i=1}^{N} \frac{1}{2} \left\|u_i - W_i \bar{r}_{i-1}\right\|^2 + \frac{\beta}{2}\left(\left\|u_N - M\bar{r}_N\right\|^2 - \left\|u_N\right\|^2\right) \quad , \tag{5.93}$$

where M is the connection matrix of the WNA connections. The noise is introduced as in the WNA-based model. It arrives at the basal dendrite of the action neurons and follows the time-continuous Ornstein-Uhlenbeck process defined in equation (5.52). The total energy function then leads to the dynamics

$$\begin{aligned} \tau \dot{u}_i &= W_i r_{i-1} - u_i + e_i \quad , \\ r_i &= \bar{r}_i + \tau \dot{\bar{r}}_i \; ; \quad e_i = \bar{e}_i + \tau \dot{\bar{e}}_i \quad , \\ \bar{e}_i &= \bar{r}_i \odot W_{i+1}^T \left(u_{i+1} - W_{i+1}\bar{r}_i\right) \quad , \\ \bar{e}_N &= \beta \left(M\bar{r}_N + \bar{r}_N' \odot M\left(u_N - M\bar{r}_N\right)\right) \quad , \end{aligned} \tag{5.94}$$

where we used that the WNA is symmetric, $M^T = M$. Here, the error-vector is more complicated, and it is not immediately clear why the learning should maximize the expected reward, but we can again look at the stream-plots of the generated error-vector.

In general, the cost function measures the self-prediction error of the action layer with a modification from the $\frac{1}{2}\left\|u_N\right\|^2$ term. The first $\bar{r}_N' \odot M\left(u_N - M\bar{r}_N\right)$ term pushes the network towards this self-predictive state (figure 5.17 A). With the WNA matrix, a point where self-prediction of the last layer is satisfied would be a

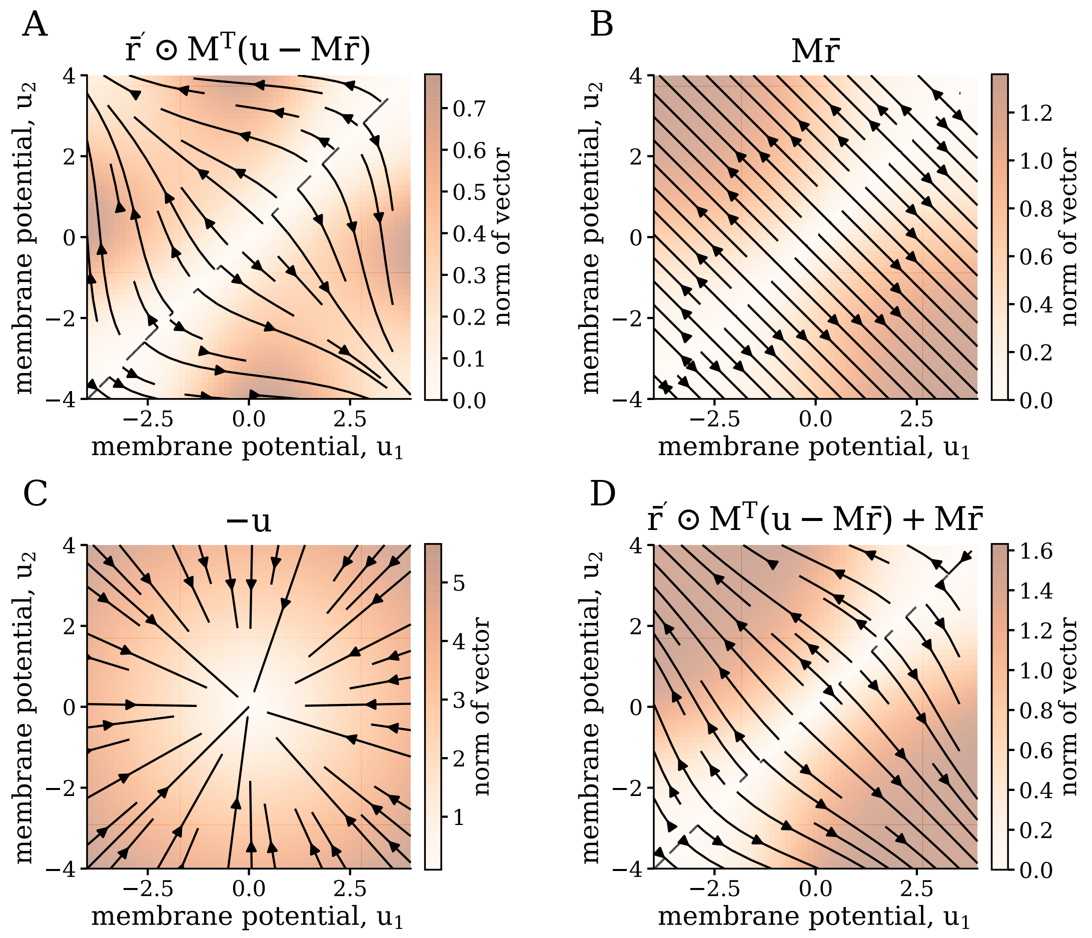


Figure 5.17: error-vector and components in the preserved synaptic connection model. **(A)** The first contribution stems from the self-prediction requirement of the cost function. It pushes the network slightly towards the closest state similar to one-hot coding. **(B)** The second term is the same WNA-based contribution as discussed in the WNA-based model. It helps maximizing the expected reward. **(C)** The third term is similar to a homeostatic term that pulls the network back towards the $u_1 = u_2 = 0$ state. This term is canceled by construction. **(D)** The combination of the two terms results in an error-vector in the action layer that enables learning. The plots were made with a sigmoid activation function $\bar{r}(u) = \frac{1}{1+\exp(-u)}$.

one-hot coding-like scenario: The strong self-connection and mutual inhibition imply that one neuron should be active predicting its high firing rate and inhibiting the firing of the other neurons. This direction also roughly matches the desired direction of learning, that is the winner is strengthened and the loosing neuron is weakened in the firing activity. The second $M\bar{r}_N$ term is the same as in the WNA-based model (figure 5.17 B). It drives the network towards maximizing the expected reward.

The term $-\|u_N\|^2$ in the cost function is required to compensate for the error-term $-u_N$ stemming from the self-prediction criteria $\|u_N - M\bar{r}_N\|^2$. At first glance, such a homeostatic term would be beneficial for the model because it would induce a homeostases from first principles (figure 5.17 B). It turns out that it harms the learning. First, this homeostatic term is now mixed together with the reward-maximizing terms and it would be modulated by the reward-prediction error. Therefore, a homeostatic mechanism would be only true for positive-valued modulation. Second, the term $-u_N$ makes the $u_N = 0$ point a stable fixed-point in the total error-vector depending on the shape of the activation function. This would introduce a region where the streamlines do not flow towards the reward maximizing direction.

Algorithm 5.4: ANN based model with preserved synaptic connection model. The algorithm is inspired by the standard policy gradient learning with online learning, that is learning on every sample. However, the error-vector is replaced by the preserved synaptic connection error and the plasticity is amended with homeostases.

```
Data: Feed-forward network parametrized by $\boldsymbol{W}$; dataset with entries $\mathbf{x}$
Initialize $\langle R\rangle_0 = 0$ ;
Initialize $\boldsymbol{W}_0$;
for $n = 1$ to max number of iterations do
    obtain random data sample $\mathbf{x}$ ;
    calculate corresponding $\mathbf{u}_N$ ;
    get sample from $\xi^{(\mathrm{ANN})} \sim \mathcal{N}(0; \sigma^2_{\mathrm{ANN}})$ ;
    calculate $\mathrm{r}_N = \rho(\mathbf{u}_N + \xi^{(\mathrm{ANN})})$ ;
    take action argmax $(\mathbf{r}) \rightarrow$ observe reward $R_n$ ;
    update weights $\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \Delta \boldsymbol{W}$ using equation (5.83) and
      equation (5.95) ;
    update $\langle R\rangle_n = \langle R\rangle_{n-1}\gamma + (1-\gamma) R_n$ ;
end
```

The full error-vector $\bar{e}_N$ (figure 5.17 D) is qualitatively suitable for reinforcement learning: It strengthens the winner neurons and penalizes the loosing neuron. Additionally, it features a push towards solutions similar to one-hot coding.

To prototype the learning capabilities of the model, we implemented an ANN-based implementation, similarly as we did for the direct node perturbation model.

The artificial model follows the same logic and algorithm as the WNA-based artificial model, but the synaptic plasticity rule in equation (5.84) is modified as:

$$\Delta W_{\text{RM}}^{(\text{PrSC})} = \eta_{\text{RM}} \left(R - \langle R \rangle \right) \frac{\partial \mathbf{u}_N}{\partial \boldsymbol{W}} \left(M r_N + r'_N \odot M \left(u_N - M r_N \right) \right) \quad . \tag{5.95}$$

The model is summarized in algorithm 5.4 and the used parameters are shown in table 5.5. In the simulations, we used the sigmoid activation function:

$$\bar{r} = \frac{1}{1 + \exp\left(-u\right)} \quad . \tag{5.96}$$

Symbol	*Value*	*Description*
-	(400, 400, 3)	layers
σ_{ANN}	0.1	exploration noise, standard deviation
η	10^{-1}	learning rate
$\eta_{\text{e-hom}}$	10^{-3}	e-hom learning rate
u_{ulow}	-1.5	lower limit potential
u_{uhigh}	1.5	upper limit potential
$\eta_{\text{m-hom}}$	10^{-5}	m-hom learning rate
u_{mid}	0	homeostases middle potential
γ	0.97	moving average parameter

Table 5.5: Parameters in the ANN-based model with preserved synaptic connections. Tables of the parameters of the simulations shown in figure 5.18.

Learning is possible similar as in previous models (figure 5.18). The expected reward increases and the test error decreases with increasing number of iterations. The model achieves an error ratio of $3.01^{+0.48}_{-0.03}\%$ on the full test set.

5.4.3 Comparison of the three models

The three presented models come with their own advantages and disadvantages (table 5.6). All three models offer realizations of reinforcement learning that harmonize with the mechanistic model of backpropagation: they can be used as additional modules attached to the backpropagation model. In principle, they would similarly harmonize with other models of biological backpropagation if they use the notion of update generation via the nudging mechanism. However, we explicitly used the fact that the principle of least action framework model does not require separate phases with and without nudging: Turning off the lateral connections in the action layer for one phase and off for another is *a priori* implausible for a biological system.

In the WNA-based model, we have shown that the generated error in the action layer indeed points in the direction of larger rewards. The proof requires a particular lateral connectivity among the action neurons, but as we have shown

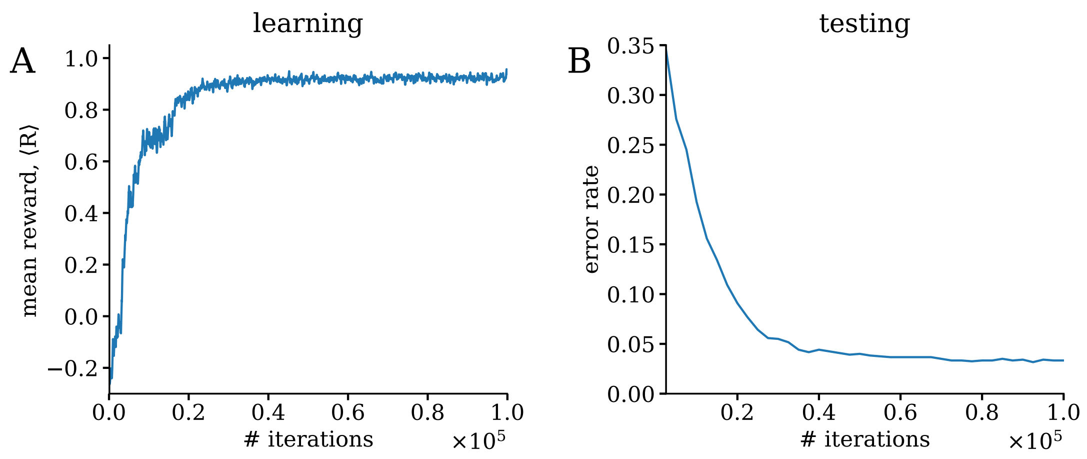


Figure 5.18: Learning results with the preserved synaptic connections model. The plot shows the learning results in terms of **(A)** median reward and **(B)** error ratio on the small test set. The median reward is plotted with a moving average of 200 iterations to reduce the variance for the sake of visualization. The results are shown as the median over 10 repetitions.

Name	WNA-based	preserved synaptic	direct node pert.
cost function	no	yes, simple	yes, structured
relation to policy gradient	proven	qualitatively	random
connections in the action layer	structured, unique	structured, invariant	none
separation of error and exploration	yes	yes	no

Table 5.6: Comparison of the three deep-reinforcement-learning models. All three proposed models have their own advantages and disadvantages.

this connectivity can tolerate fixed-pattern noise of up to 20%. Finally, the WNA-based model displays a separation of error generation and exploration. While the exploration noise arrives at the basal dendrite, the error-vector is generated via nudging connections at the soma. In the abstract analogies, this does not mean a difference, but in the time-continuous model it becomes essential. The back-propagation theorem (theorem 1) assumes weak nudging. Therefore, a coupling between exploration and error generation would restrict the magnitude of the exploration. This would impair the ability of the network to abandon strongly imprinted but suboptimal choices.

In the PrSC-Model model, we have an explicit cost function with the appealing property that it has the same form as the prediction-error cost functions throughout the rest of the network. Although, an additional term was required to make the learning more robust or depending on the activation function even possible. We lack a rigorous proof for the error maximization property, but the phase-space analysis for two dimensions and the simulation results suggests that the error-vector is suitable for reinforcement learning. The synaptic connections between the action neurons are even more elaborate than in the WNA-based network, and its robustness to fixed-pattern noise is to be shown. However, they follow the same stereotypical wiring structure as other parts of the brain. The PrSC-Model also provides the separation of exploration and error-vector generation.

Finally, the direct node perturbation model comes with the simplest cost function. The model does not require any lateral connection between the action neurons; the learning is purely based on the correlation of the received error and the random nudging on the action neurons. Both the advantages and disadvantages of this method lie in this simplicity. Robustness to fixed-pattern noise in the error-generation is not an issue due to lack of lateral connections. However, the learning is slow, because the random nudging does not provide any information about the activity structure of the action neurons.

In early learning, the random nudging could make the difference between two actions corresponding well to the reward. When two actions have approximately the same action value, the random perturbation could decide the winner. Hence, the random perturbation is causally connected to the obtained reward. In late learning, when some of the actions are already strongly imprinted, the random nudging loses its causal connection to the obtained reward. For example, if an action has a much larger action value then the others, that is the difference is several times the magnitude of the random perturbations. In this case, the perturbation cannot influence the winner-selection and hence it is uncorrelated with the obtained reward.

Note that our proposed model is more "intelligent" than other models where random node perturbation is imposed in the entire network [Fiete and Seung, 2006]. In our model, the random perturbation acts only in the space of the actions which has much smaller dimension than the space of the entire network. The random perturbation on the action neurons is backpropagated to the hidden layers by the neuronal dynamics and the microcircuit structure. Finally, the direct node perturbation model couples the exploration and error-vector generation; and hence it restricts the magnitude of exploration in the time-continuous framework.

In the main part of this study, we opted for the WNA-based model because of its rigorous reward maximizing property and its close relation to the winner-take-all like structures, which are well established in the computational neuroscience literature (see section 5.5.2).

5.5 Discussion

In this chapter, we presented an extension of biologically plausible deep learning in the principle of least action framework [Senn et al., in preparation] to reinforcement learning. By that, we achieved a model that combines the reinforcement learning paradigm, time-continuous dynamics and deep learning without phases in a biologically plausible mechanistic model. We introduced a winner-nudges-all (WNA) lateral synaptic connection structure among the action neurons, that resembles a soft winner-take-all circuit (figure 5.4). These lateral connections generate an appropriate error-vector for deep learning. We showed that the resulting error-vector combined with the reward-prediction error, a single global modulator signal (equation (5.55)), drives the learning towards maximizing the mean expected reward.

We tested the model on a reduced version of the MNIST dataset. The time-continuous model reached similar performance as the standard deep-policy-gradient method (section 5.3.2). Further, we analyzed the robustness of the model, and found that it is robust against delays in the reward feedback and to fixed-pattern noise (inhomogeneities) on the synaptic weights of the WNA circuit (section 5.3.3). Finally, we also briefly sketched two alternative models for deep reinforcement learning and discussed their advantages and disadvantages (section 5.4).

We also gave postdictions to the potential mechanistic meaning of cortical microcircuits, soft winner-take-all structures and the close input-following property discussed by Köndgen et al. [2008]. Although these postdictions are not particularly novel on their own: 1) Sacramento et al. [2018] introduced the stereotypical microcircuits for backpropagation of errors, 2) Senn et al. [in preparation] used the input-following property to realize learning without phases and 3) WTA-like structures have been already used for reinforcement learning, for example in Frémaux et al. [2013]. The novelty in our work is their combination in the context of time-continuous deep reinforcement learning. With this work, we contribute to the search for deep reinforcement learning in the brain by providing plausible mechanistic models with interpretable neurodynamics, phase-less learning, self-generated plasticity without an external supervisor as well as time-continuous dynamics.

5.5.1 Limitations of the study

One of the main limitations of the study is the tedious numerics required to integrate the model, which restricted the maximum number of iterations during learning and the size of the used dataset. Simulating the dynamics requires solving a system of linear equations for each time-step because the time-derivative of the membrane potential is given as a self-consistency equation (appendix A.3.1). This requirement originates from the look-ahead firing property of the neurons, and it is closely tied to the desired phase-less and real-time learning capability. To circumvent the problem, we designed ANN-based corresponding models that enabled us extensive parameter-sweeps. From a computational neuroscience perspective, this is merely a practical problem, and it could be tackled by using more advanced numerical methods. If the framework is regarded as a contribution to machine learning research, then the tedious simulation is a drawback. Hence, the proposed learning framework contributes rather to the modeling of biologically plausible deep-learning.

Our model exhibits several aspects found in biological systems such as the time-continuous neural and synaptic dynamics and the cortical microcircuits, but it has gaps in the mechanistic modeling. The central gaps are related to the look-ahead principle.

First, the look-ahead mechanism is problematic for spikes: the look-ahead mechanism is formulated for time-continuous input for the neurons. Formulating the same look-ahead mechanism is challenging for spiking input because of the non-differentiable nature of the spikes. If we assume that spike-rates are proxies of firing rates — what we indeed do by using a rate-based model — we have to average over several spikes in a neuron's activity. Depending on the firing-rate of a neuron, this can mean averaging over a time on the order of time-constant τ of the look-ahead mechanism. Alternatively, we can assume that the input to the neurons stems from a pool of pre-synaptic neurons, and hence the rate-based input idea holds as an average over this pre-synaptic pool.

Second, in experiments [Köndgen et al., 2008] the neurons can follow closely the input if the input is sufficiently slow. The experiments were carried out with sinusoidal input on the neurons; at high input-frequencies the following property breaks down and the neuron follows the input with a phase lag. In contrast, our proposed look-ahead mechanism can follow any differentiable input, if we neglect the practical problems regarding numerics. Designing mechanistic models of the look-ahead mechanism should be the topic of further research. These envisioned models could fill in the gap between the experimentally observed look-ahead and the look-ahead property in our model.

In our model we only implicitly included aspects of reinforcement learning such as the computation of the reward-prediction error and the strong action-selection mechanism. We focused on the interaction of phase-less backpropagation and the generation of output errors. Of immediate relevance is the strong action-selection: feedback from the strong action-selection layer could contribute to the error-vector generation, and might thereby improve the learning.

Our model realizes a model-free reinforcement learning scheme (neuroscience terminology), also called end-to-end reinforcement learning (machine learning terminology). This means that the network does not use previous knowledge about the environment and learning is purely based on the experience it gathered by trial and error. Model-free reinforcement learning has been criticized both from a biological and machine learning point of view. On the one hand, animals and humans can clearly do more than trial and error [Niv, 2009]. On the other hand, model-free learning requires a huge number of iterations, which would be clearly impractical and expensive if, for example, one would try to learn to drive a car with a model-free algorithm [LeCun, 2019]. This observation applies to biological systems as well. Still, we think that it is worth studying these types of models. On the one hand, there are indications that model-free learning is present in the brain, for example in situations when habitual responding is favored Niv [2009]. On the other hand, the proposed model could be combined with other learning mechanism, for example unsupervised or self-supervised learning [LeCun, 2019], and hence perform efficient reinforcement learning based on preprocessed input.

The used MNIST [LeCun et al., 1998] dataset is small and — from the perspective of machine learning research — too simple to show competitive results. But MNIST only serves as an example to demonstrate the model's learning capability. The main limitation of MNIST lies not in its small size but in its nature: MNIST consists of static images. For the nervous system, even looking at static images produces a dynamic stimulus due to the small involuntary eye-movements (microsaccades) [Martinez-Conde et al., 2009, Rolfs, 2009]. Hence, including more dynamic behavior into the datasets would be more interesting for biology-focused research than increasing the complexity of otherwise static images. Alternatively, creating dynamic input based on static images could be seen as part of the model, similarly to a pre-processing step. The underlying model in Senn et al. [in preparation] is based on learning immediate input-output relations through a neural network, therefore further work is required to include learning from temporal sequences.

5.5.2 Relation to other works

Models of deep learning in the brain

Several publications aim to give biologically plausible models of backpropagation. They tackle the problem from different perspectives such as synaptic tagging [Roelfsema and Ooyen, 2005, Rombouts et al., 2015], relation to Hebbian learning rules [O'Reilly, 1996, Xie and Seung, 2003, Amit, 2019], predictive coding [Whittington and Bogacz, 2017], introduction of feedback gating ghost units [Mesnard et al., 2019] and deriving the dynamics from an energy function [Scellier and Bengio, 2017]. All of these models come with a different level of biological plausibility. Some of them lack time-continuous dynamics, and all of them assume the presence of separate phases for the forward propagation of information and the backpropagation of errors, or separate the plasticity and the network dynamics by using sufficiently small learning rates. Marblestone et al. [2016] and Whittington and Bogacz [2019] give recent reviews on many of the candidate algorithms. Richards et al. [2019] aim to give a "principled perspective" to define the search for deep learning in the brain. They emphasize that more focus should be given to objective functions, learning rules and architectures in neuroscience to achieve a more rapid progress. In a project complementary to the pursuit of biological deep learning, Illing et al. [2019] study the learning capabilities of biological models of shallow-learning combined with different preprocessing steps. They propose that their results could be a comparison baseline for models of deep learning.

In this project we built largely on three previous studies. Urbanczik and Senn [2014] established the idea of plasticity based on dendritic predictions and the introduced the conductance-based nudging as a supervisory signal guiding learning. Conductive nudging has the desirable property that the amount of nudging depends on the distance of neuron's membrane potential to the target value. A naïve neuron before learning, with a membrane potential far away from the target, receives a large supervisor current. In contrast, a learned neuron already with the desired target receives none. Hence, the learning is continuous and can be left active after the learning phase because in lack of conductive nudging the learning stops. Sacramento et al. [2018] used the microcircuit structure for error-backpropagation and they combined them with learning from dendritic predictions. By that, they formed a bottom-up model of time-continuous backpropagation in layered neural networks. The authors also relaxed the requirements of shared weights in the microcircuits by mechanism similar to feedback alignment [Lillicrap et al., 2016] and by introducing plasticity rules for the lateral connections. Senn et al. [in preparation] introduced the idea of using the principle of least action to derive the neuronal and synaptic dynamics. This results in the look-ahead firing of the neurons keeping the feed-forward information and feedback error-propagation in phase. We discussed the content of Senn et al. [in preparation] in detail in section 5.1.

Winner-takes-all structures in models of neural networks

Intralayer WTA structures with short-range excitation and long-range inhibition are widely used in simulation-based models of the nervous system and in neural implementations of learning. They appear — without claim to completeness — in models of decision making [Wang, 2008], in models of object recognition [Riesenhuber and Poggio, 1999], in models of self-organized maps in the visual cortex [Miikkulainen et al., 2006], in biological implementations of sparse coding [Rozell et al., 2008, Ecke et al., 2019], in models of reinforcement [Frémaux et al., 2013, Leimer et al., 2019] and unsupervised learning [Habenschuss et al., 2013, Urbanczik and Senn, 2014, Diehl and Cook, 2015, Breitwieser, 2015], and even in experiments with learning on neuromorphic hardware [Kreiser et al., 2017, Spilger, 2018].

There are two main types of WTA circuits. In the hard WTA circuits, we assume that the spiking activity of a single neuron completely inhibits the activity of the other neurons, e.g. in Diehl and Cook [2015], Breitwieser [2015], Spilger [2018], Leimer et al. [2019]. This can be understood as a hard decision-making process. In soft WTA circuits, the spiking activity is not reduced to solely one neuron. The winner only increases its own activity and decreases the activity of others, but does not inhibit them completely, e.g. in Habenschuss et al. [2013], Frémaux et al. [2013], Urbanczik and Senn [2014].

Our approach is inspired by the soft WTA mechanism in Urbanczik and Senn [2014]. There, the WTA circuit provides a small nudging of the somatic membrane potential that drives learning. However, the authors do not call the resulting nudging an error-vector. The novelty of our work lies in interpreting the lateral nudging in the action layer as an error-vector, making it compatible with the error-backpropagation mechanism. Furthermore, we proved the reward-maximizing property of the plasticity rule (section 5.2.2).

Q-AGREL: a model for biological deep learning with attention network and synaptic tagging

Recently, the Q-AGREL model has been proposed as a biologically plausible model for deep reinforcement learning [Pozzi et al., 2018]. AGREL stands for attention-gated reinforcement learning, and the Q refers to the q-values of the actions. Q-AGREL assumes different mechanisms to implement a similar functionality as our network (table 5.7).

Model	Q-AGREL	this model
dynamics	no notion of time	time-continuous
phases in backpropagation	yes	no
action selection	max-Boltzmann controller	time-continuous noise
error accumulation	synaptic tag	eligibility trace
modeling approach	bottom-up	top-down dynamics, bottom-up homeostasis
dataset	MNIST, CIFAR10/100	reduced MNIST

Table 5.7: Comparison of the time-continuous reinforcement learning model and Q-AGREL. Comparison of our model and the Q-AGREL model described in Pozzi et al. [2018].

The main differences between their and our work are the presence of separate forward and backward phases and the lack of neural dynamics in Q-AGREL. Q-AGREL is based on neuron models that only consider input-output relations as in artificial neural networks (section 2.1.1). In the forward pass a hard WTA-based action selection is assumed in the action layer. The action selection is implicitly modeled by a max-Boltzmann controller [Wiering and Schmidhuber, 1997], a modification of the ϵ-greedy policy [Sutton and Barto, 2018]. In the backward pass, only the activated action backpropagates error signals via an attention network. The error-term in the single synapses is held in a so-called synaptic tag, which is similar to our eligibility trace assumption. Finally, a reward prediction error modulates the plasticity of the synapses. Q-AGREL is fully constructed as a bottom-up model.

We argue that our model includes more mechanistic details than Q-AGREL such as time-continuous dynamics, cortical microcircuits, error-vector-generating WNA circuits and the explicit exploration mechanism; at the expense of simulation complexity. Pozzi et al. [2018] tested Q-AGREL not only on the MNIST [LeCun et al., 1998] but also on CIFAR10 and CIFAR100 [Krizhevsky et al., 2009] datasets with good results, while our model was restricted to a reduced version of MNIST due to the numerical complexity.

5.5.3 Conclusion and outlook

We presented a theory of time-continuous deep reinforcement learning. We see three main paths for future work.

We used a simple classification task to prototype and demonstrate the learning capabilities of the model, but classification is an unusual task for reinforcement learning. In order to apply the model to typical reinforcement learning tasks (cart-pole, water maze), we have to extend the model to enable learning over different states of the environment. Inspired by the works of Doya [2000] and Frémaux et al. [2013], we suggest an actor-critic framework where the two networks get input from the current state of the environment (figure 5.19). In the action layer of the actor network, a WNA structure gives rise to an error-vector, while in the critic network the neuron representing the state value gets constant positive nudging. Plasticity in the whole network is modulated by the time-continuous version of the reward-prediction error [Doya, 2000, Frémaux et al., 2013]. Frémaux et al. [2013] used an almost identical framework, but their approach did not scale well for high-dimensional problems since their model was restricted to shallow learning only. Our framework could mitigate this problem because both the critic and the actor network can learn hierarchies of non-linear features.

Another future path of research could focus more on the mechanistic modeling of the look-ahead mechanism. The look-ahead firing of the neurons is a central property, which gives rise to the possibility of backpropagation without phases. Currently, it appears as a prescribed dynamic of the neurons. It would be interesting to study the possible mechanisms of look-ahead firing in biological neurons and their compatibility with deep learning in a network. This effort would greatly increase the biological plausibility of the model. In addition, understanding the mechanism and requirements of look-ahead firing would be the first step towards a potential neuromorphic implementation.

Finally, in the current form, this project only gives postdictions to biological observations and lacks any directly falsifiable predictions. The quest for biological deep learning has only recently gained momentum and several models have been

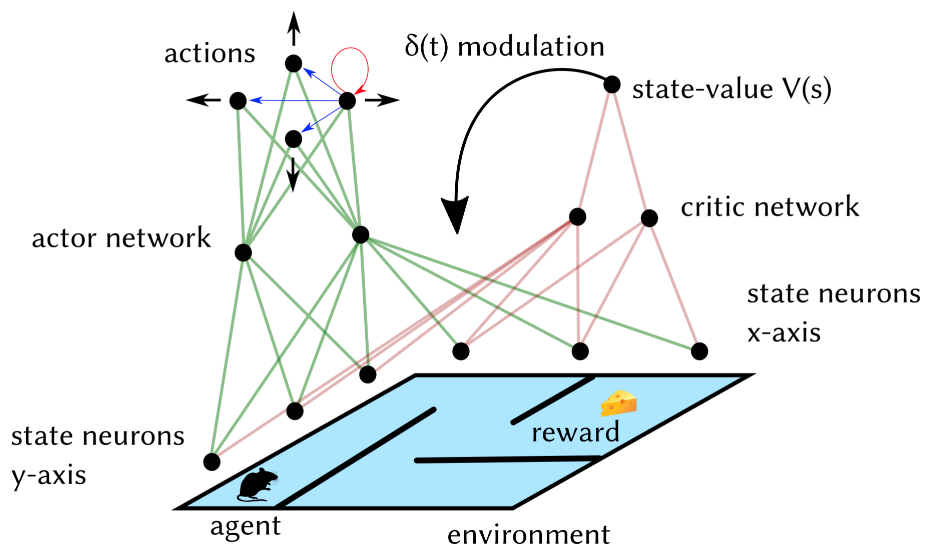


Figure 5.19: Sketch of the proposed actor-critic network architecture. We propose to extend our model to an actor-critic architecture. The actor network implements the current policy: at the action layer it forms a WNA circuit to give rise to an error-vector. The critic network learns the state value of the current state of the environment. The complete network underlies the neuro-synaptic dynamics of the principle of least action framework. The plasticity is modulated by the time-continuous version of the reward prediction error $\delta(t)$ [Doya, 2000, Frémaux et al., 2013], which is calculated as a combination of the learned critic value and the obtained reward. Due to the backpropagation property, the network could learn non-linear features from the state of the environment. The cheese and the mouse motives are taken from Clipart [2020].

proposed how the brain might implement deep learning. We should focus more on suggesting testable predictions to allow for a closer feedback-loop between theory and experiment.

6 Concluding remarks

In this thesis, we present three projects contributing to the pursuit of robust and powerful learning algorithms for biological and neuromorphic substrates. Throughout the projects, we emphasize the aspect of robustness to disrupting effects. We find that appropriate learning rules can help mitigating distorting effects on analog neuromorphic hardware and at the same time the implementation can benefit from the advantages of the neuromorphic substrate. This result helps to extend the range of applications of neuromorphic hardware. Further, we implicitly show the consecutive steps of model-development for imprecise substrates: from the inspiration in machine learning, via the development of a computational neuroscience model to the implementation and benchmarking on the neuromorphic hardware. Finding powerful mechanistic computational models is central for neuromorphic engineering to realize its advantages in applications and for computational neuroscience to understand the information processing in the brain. Here, we discuss the contribution of these projects to the synergistic developments of the brain-inspired research fields (figure 6.1).

In chapter 3, we present an implementation of accelerated Bayesian inference on the BrainScaleS-1 system (BSS-1) neuromorphic system. The project exemplifies how a model rooted in machine learning is adapted and developed for mixed-signal neuromorphic substrate. The introduced iterative learning scheme is able to mitigate distorting effects resulting from fixed-pattern noise on the neuromorphic hardware. The implementation benefits from the acceleration of the hardware reaching two orders of magnitude speed-up compared to equivalent biological real-time. Due to the underlying probabilistic paradigm, the model is not only able to perform classification but pattern completion and data generation as well. Appealing characteristics of the model are its capability to perform both inference and generative tasks with the same network, its local learning rule and the wide applicability of the underlying machine learning model. The model could be extended to solve more complex tasks, or to include on-chip learning. The latter combined with the immense acceleration of analog neuromorphic hardware would be interesting for machine learning applications.

In chapter 4, we present a demonstration of advantages of neuromorphic computation. In the project, we study and — more importantly — quantify the aspects of robustness, speed, energy consumption and self-learning capability of neuromorphic hardware. The results demonstrate that a learning rule implemented and executed completely on-chip can compensate for imperfections on the analog hardware. The presented numbers of power-consumption and speed give important estimates about what is to be expected from the accelerated mixed-signal neuromorphic approach. The neuromorphic setup outperformed the equivalent

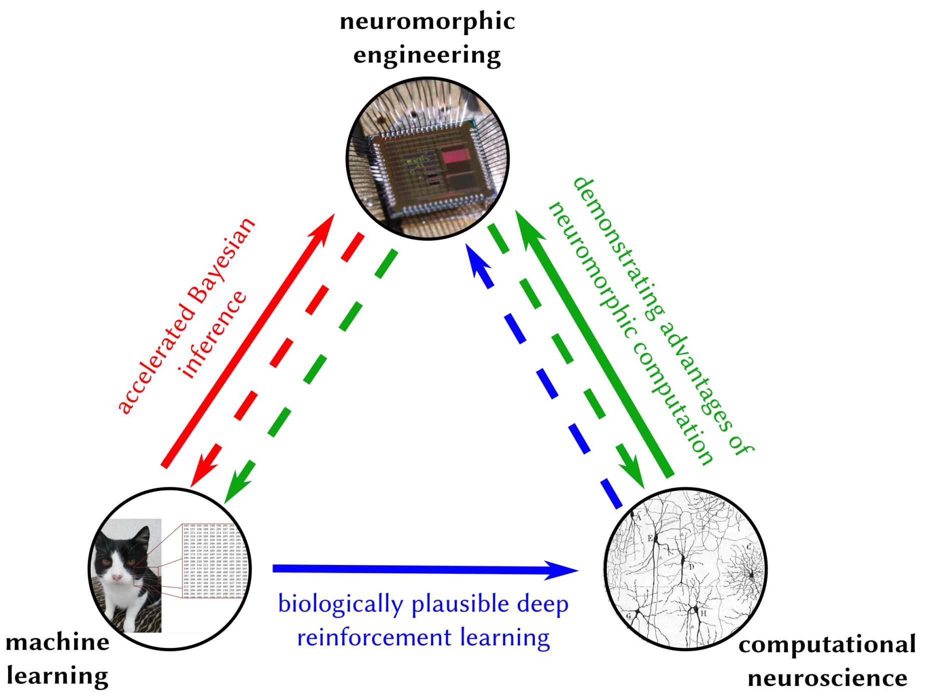

neuromorphic engineering
accelerated Bayesian inference
demonstrating advantages of neuromorphic computation
machine learning
biologically plausible deep reinforcement learning
computational neuroscience

Figure 6.1: Contribution of the presented projects to synergistic advancement of the three fields. Summary of how the presented projects contribute to the advancement of neuromorphic engineering, computational neuroscience and machine learning. The unbroken lines symbolize the contribution of the projects, and the dashed lines indicate their potential further effect or extension. The three projects are: i) accelerated Bayesian inference (chapter 3, red), ii) demonstrating advantages of neuromorphic computation (chapter 4, green) and iii) biologically plausible deep reinforcement learning (chapter 5, blue). Compare to figure 1.1 that summarizes the general interaction between the three fields.

software implementation running on a CPU by an order of magnitude in terms of speed of computation and by three orders of magnitude in terms of energy consumption. Especially, the speed-comparison with the CPU-based simulator suggests that accelerated neuromorphic hardware has the potential as a low-precision but high-speed simulator for computational neuroscience. The analyzes of robustness and the transferability of the results between chips are key observations for potential applications.

In chapter 5, we show a biologically plausible model of deep reinforcement learning. First, the project contributes to the recently reopened pursuit for deep learning mechanisms in the brain by providing a mechanistic model with time-continuous dynamics and self-learning capabilities. The robustness of the model to small time-delays in the reward and its robustness to fixed-pattern noise can be seen as first steps towards a potential neuromorphic implementation. However, to reach this potential extension several additional steps are needed, such as the inclusion of spikes in the model.

There are also interactions and connections among the three projects. First, we can see accelerated Bayesian inference and biologically plausible reinforcement learning as two similar projects at different milestones of their development. They both fit into the logic that a model originally from machine learning (Boltzmann machines and backpropagation) is extended into a model in computational neuroscience and potentially further towards application on neuromorphic hardware. In this analogy, the project of deep reinforcement learning is at the same stage as accelerated sampling was when Buesing et al. [2011] mapped sampling to the dynamics of spiking neural networks. Future research might lead to a neuromorphic implementation, similarly as in chapter 3.

Second, mainly two factors limited the demonstration of advantages of neuromorphic computation (chapter 4): 1) the size of the prototype chip and 2) the availability of spike-based algorithms with local learning rule accessible for on-chip learning. Chapter 3 and chapter 5 connect to the second limitation. Both the spike-based inference framework and the deep reinforcement learning framework are promising candidates as powerful spike-based algorithms with local learning rules for implementation on a neuromorphic hardware with on-chip learning. In case of deep reinforcement learning, several further intermediate steps are required for a neuromorphic implementation. The spike-based Bayesian inference framework has been successfully implemented on the BSS-2 prototype chip [Billaudelle et al., 2019a]. The framework is a good starting point for developing an on-chip version of the used (local) Hebbian learning rule.

The continuous interaction among these fields of research has led to remarkable progress in the past and we expect that we will benefit from it in the future as well. This thesis contributes to this synergistic advancement of science towards understanding the human brain and towards building novel hardware inspired by our new insights.

Appendix

A Calculations and code examples

A.1 Source code examples for accelerated Bayesian inference on BSS-1

The source code is available upon request from the Electronic Vision(s) Group Heidelberg or from the author in the repository `model-hw-sampling-hicann`. For the setup the source code uses the PyNN API [Davison et al., 2009].

```python
def setUpNetwork(self):
    """ Set upt the network based on the provided connection matrix
                    and the bias vector """

    # Make an initial guess for the correct weights and biases
    # The guess is based on the typical result
    # for the activation function
    # using bias neurons
    self.N = len(self.b_target)  # number of neurons
    self.b_HW_init = 1. * copy.deepcopy(self.b_target)  # initial b guess
    self.W_HW_init = 1. * copy.deepcopy(self.W_target)  # initial W guess

    # create mask for the RBM
    # not seen weights are not realized
    self.mask = SF.createRbmMask(
        self.N_visible + self.N_label, self.N_hidden)

    # Set up the network using pynn
    BmParams = {'N': self.N,
                # from central database
                'samplerParams': database.NEURON_PARAMS,
                 # from central database
                'biasParams': database.BIAS_PARAMS,
                'mask': self.mask,
                'W': self.W_HW_init}

    # options to specify the number of bias neurons
    # and to leave gaps between neurons while mapping
    if self.N_bias:
        BmParams['N_bias'] = self.N_bias
    if self.gapPlacement:
        BmParams['gap'] = self.gapPlacement
    [self.bmPops, self.samplers, self.biases, self.biasConn,
        self.samplingConn] = SF.createSkeletonBmMasked(BmParams)

    # activate the spike readout for the samplers
    for i in range(self.N):
        self.samplers[i].record()
```

```
# if specified then the list of the neurons is shuffled
if self.randomPlacement:
    np.random.shuffle(self.bmPops)

# Create the decorrelation network
[self.sonNeur, self.sonSyns] = SF.createSimpleSon(self.Son_params)

# Connect noise from the sea of noise network to the samplers
self.con = pynn.FixedNumberPreConnector(self.noisePartners,
                                        weights=1,
                                        allow_self_connections=False)
self.excNoise = pynn.Projection(self.sonNeur,
                                self.samplers,
                                self.con,
                                target='excitatory')
self.inhNoise = pynn.Projection(self.sonNeur,
                                self.samplers,
                                self.con,
                                target='inhibitory')
```

The training loop iterates between execution on the hardware and gradient evaluation on the host computer. The following function shows that the gradient is obtained in two experiments on the neuromorphic hardware.

```
def getOneGradient(self, miniBatch, W, b):
    """
        Make one CD step on a given example
        specified by its number in the dataset.
            -- Keywords: miniBatch
                -- miniBatch: minibatch in matrix form
                -- W: weight matrix
                -- b: bias vector
            -- Returns: [db, dW]
                -- db: Calculated gradient for the biases
                -- dW: Calculated gradient for the weights
    """

    ##########################
    # Emulate the data term  #
    ##########################

    # Get and apply the spiketrains
    clamping = SF.spikeTrainsFromPic(
        miniBatch,
        self.durationCD,
        10.,
        self.labels,
        withLabels=True)
    self.applyClamping(clamping, withLabels=True)

    # Turn-off the connections from hidden to visible and label
    W_mod = copy.deepcopy(W)
    W_mod[self.N_label + self.N_visible:,
          :self.N_label + self.N_visible] = 0
    self.applyNetworkChanges(W_mod, b)

    # Run the experiment
    numExamples = len(miniBatch[:, 0])
    endOfClamping = numExamples * self.durationCD
    duration = endOfClamping
    self.runExperiment(duration)

    # Gather the spikes
    spTrains = {}
    for inner in range(self.N):
        spTrains[inner] = self.samplers[inner].getSpikes()[:, 1].tolist()
```

```
# Evaluate the spike trains
# Calculate the data term based on the results
C_data = SF.calculateMiniBatchDataTerm(self.durationCD,
                                       spTrains,
                                       self.tau_mean,
                                       numExamples)

###########################
# Emulate the model term  #
###########################
# Obtain and apply clamping to gather the recontruction terms
clamping = SF.spikeTrainsFromPicModel(
    miniBatch,
    self.durationCD,
    10.,
    self.labels,
    withLabels=True)
self.applyClamping(clamping, withLabels=True)

# Turn-on the connections from hidden to visible and label
self.applyNetworkChanges(W, b)

# Run the experiment
timer.start()
endOfClamping = numExamples * self.durationCD * 2.
self.runExperiment(endOfClamping)

# Gather the spikes
spTrains = {}
for inner in range(self.N):
    spTrains[inner] = self.samplers[inner].getSpikes()[:, 1].tolist()

# Calculate the data term based on the results
timer.start()
C_model = SF.calculateMiniBatchModelTerm(self.durationCD,
                                         spTrains,
                                         self.tau_mean,
                                         numExamples)

dC = C_data - C_model
return [dC.diagonal(), C_data - C_model]
```

A.2 Source code examples for demonstrating advantages of neuromorphic computation

The source code is available upon request form the author or from the Electronic Vision(s) Group Heidelberg. The source code is found in `model-hw-pong` for the experiment and in the `model-sw-pong` for the software simulations. The hardware implementation was made with the `frickel-dls` repository, which is a gathering of low-level (close to the hardware) convenience function for using the BSS-2 neuromorphic chip. This can be seen on the implementation as well, because it uses more low-level functions to use the chip.

```python
class PongDLS:
    def __init__(self, runs_at_a_time=1, folder=None, cfgfile=None, \
            noisy_weights=None, chip=None, board=None, delay=0.001):
        logging.debug("Initializing PongDLS")
        self.path = os.path.dirname(os.path.abspath(__file__))
        # Loading the PPU program
        self.program = os.path.join(self.path,
                                    "ppu/firmware/pongdls.binary")
        if board is None:
            self.board = pydls.get_allocated_board_ids()[0]
        else:
            self.board = board
        logging.debug("Using board %s" % self.board)
        self.folder = folder
        # choose chip if specified
        if chip is None:
            if self.board == "B201330":
                self.chip = "22"
            elif self.board == "07":
                self.chip = "22"
            elif self.board == "B201319":
                self.chip = "22"
            else:
                logging.warning(
                    "Could not find chip for board.\
                     Using default chip 20.")
                self.chip = "20"  #return
        else:
            self.chip = chip
        self.mailbox_marker = 1
        self.mailbox_offset = 0x3000 / 4
        self.mailbox_size = (32 + 32 * 32 / 4) * 2
        self.message_location = 0
        self.runs_at_a_time = runs_at_a_time
self.run_delay = delay
self.normalization = float(0x7fffffff)
self.running = False
self.kys = False
self.setup_done = False
self.c = pydls.connect(self.board)
logging.debug("Connected to board %s" % self.board)
# load configuration file
self.dlscfg = PongDLSConfig(self.c, self.chip, self.board,
                            self.program,
                            cfgfile=cfgfile)
if noisy_weights:
    logging.debug(
        "Setting noisy Gaussian weights\
         with mean %d, std %d." %
        (noisy_weights[0], noisy_weights[1]))
    self.dlscfg.set_noisy_weights(noisy_weights[0],
                                  noisy_weights[1])
```

At the time of the project, the PPU had to be programmed with low-level calls and functions. Below, we show the plasticity rule as it is programmed on the PPU.

```
static void reward_network()
{
    uint8_t row;
    switch (abs(paddle.target_cell - ball.y_cell)) {
        case 0:
            reward = 0x7f;
            break;
        case 1:
            reward = F8(0.7);
            break;
        case 2:
            reward = F8(0.4);
            break;
        case 3:
            reward = F8(0.1);
            break;
        case 4:
            reward = F8(0.0);
            break;
        default:
            reward = 0;
            break;
    }
    // if this is the first run, set mean reward to current reward
    if (mean_rewards[ball.y_cell] == 128) {
        mean_rewards[ball.y_cell] = reward;
    }
    success = reward - mean_rewards[ball.y_cell];

    VR_SUCCESS = (vector uint8_t)fxv_splatb(
        (success >> learning_rate) + success_offset);
    mean_rewards[ball.y_cell] = mean_rewards[ball.y_cell] +
        (success >>  avg_runs);
        rewards[ball.y_cell] = reward;

    // update all synapse rows
    for (row = 0; row < 32; row++) {
        get_causal_correlation(&VR_CAUSAL_0, &VR_CAUSAL_1, row);
        get_weights(&VR_WIN_0, &VR_WIN_1, row);
        VR_CAUSAL_OFF_0 = causal_offsets_0[row];
        VR_CAUSAL_OFF_1 = causal_offsets_1[row];
                asm volatile (
                    "fxvshb␣%[W0],␣%[W0],␣1\n"
                    "fxvshb␣%[W1],␣%[W1],␣1\n"
                    "fxvshb␣%[COFF0],␣%[COFF0],␣-1\n"
                "fxvshb␣%[COFF1],␣%[COFF1],␣-1\n"
                "fxvshb␣%[C0],␣%[C0],␣-1\n"
                "fxvshb␣%[C1],␣%[C1],␣-1\n"
                "fxvsubbfs␣%[C0],␣%[C0],␣%[COFF0]\n"
                "fxvsubbfs␣%[C1],␣%[C1],␣%[COFF1]\n"
                "fxvcmpb␣%[C0]\n"
                "fxvsel␣%[C0],␣%[C0],␣%[CONST0],␣2\n"
                "fxvcmpb␣%[C1]\n"
                "fxvsel␣%[C1],␣%[C1],␣%[CONST0],␣2\n"
                "fxvmulbfs␣%[WADD0],␣%[C0],␣%[S]\n"
                "fxvmulbfs␣%[WADD1],␣%[C1],␣%[S]\n"
                "fxvaddbm␣%[TMP0],␣%[WADD0],␣%[CONST1]\n"
                "fxvcmpb␣%[WADD0]\n"
                "fxvsel␣%[WADD0],␣%[WADD0],␣%[TMP0],␣2\n"
                "fxvaddbm␣%[TMP0],␣%[WADD1],␣%[CONST1]\n"
                "fxvcmpb␣%[WADD1]\n"
                "fxvsel␣%[WADD1],␣%[WADD1],␣%[TMP0],␣2\n"
                "fxvaddbfs␣%[W0],␣%[W0],␣%[WADD0]\n"
                "fxvaddbfs␣%[W1],␣%[W1],␣%[WADD1]\n"
                "fxvcmpb␣%[W0]\n"
                "fxvsel␣%[W0],␣%[W0],␣%[CONST0],␣2\n"
                "fxvcmpb␣%[W1]\n"
                "fxvsel␣%[W1],␣%[W1],␣%[CONST0],␣2\n"
                "fxvsubbfs␣%[TMP0],␣%[W0],␣%[CONST127]\n"
                "fxvcmpb␣%[TMP0]\n"
                "fxvsel␣%[W0],␣%[CONST127],␣%[W0],␣2\n"
                "fxvsubbfs␣%[TMP0],␣%[W1],␣%[CONST127]\n"
                "fxvcmpb␣%[TMP0]\n"
                "fxvsel␣%[W1],␣%[CONST127],␣%[W1],␣2\n"
                "fxvshb␣%[W0],␣%[W0],␣-1\n"
                "fxvshb␣%[W1],␣%[W1],␣-1\n"
                : [W0] "+kv" (VR_WIN_0),
                  [W1] "+kv" (VR_WIN_1),
                  [C0] "+kv" (VR_CAUSAL_0),
                  [C1] "+kv" (VR_CAUSAL_1),
                  [TMP0] "+kv" (VR_TMP_0),
                  [WADD0] "+kv" (VR_WEIGHT_ADD_0),
                  [WADD1] "+kv" (VR_WEIGHT_ADD_1)
                : [COFF0] "kv" (VR_CAUSAL_OFF_0),
                  [COFF1] "kv" (VR_CAUSAL_OFF_1),
                  [CONST0] "kv" (VR_CONST_0),
                  [CONST1] "kv" (VR_CONST_1),
                  [CONST127] "kv" (VR_CONST_127),
                  [S] "kv" (VR_SUCCESS)
                      : /* no clobber */
                    );
    set_weights(&VR_WIN_0, &VR_WIN_1, row);
}
```

```
}
```

A.3 Source code examples and implementation details for time-continuous deep reinforcement learning

A.3.1 Implementation details

In the following, we describe the details of the numerics of the implementation, and show why the simulation of the time-continuous dynamic is tedious even with the help of GPUs. The noise is simulated according to equation (5.52):

$$\mathrm{d}\xi = \frac{1}{\tau_{\mathrm{OU}}}(\mu_{\mathrm{OU}} - \xi)\mathrm{d}t + \sqrt{\frac{2}{\tau_{\mathrm{OU}}}}\sigma_{\mathrm{OU}}\mathrm{d}\mathcal{W} \quad , \tag{A.1}$$

where τ_{OU} is the autocorrelation time, μ_{OU} is the equilibrium point and σ_{OU} is the standard deviation of the steady-state solution. In the dynamics we also need the low-pass filtering of the noise

$$\bar{\xi}(t) = \frac{1}{\tau}\int_{-\infty}^{t} \xi(\hat{t}) \exp\left(-\frac{t-\hat{t}}{\tau}\right) \mathrm{d}\hat{t} \quad . \tag{A.2}$$

In simulation we access the low-pass filtered $\bar{\xi}$ by simulation the ordinary differential equation

$$\dot{\bar{\xi}} = \frac{1}{\tau}\left(\xi - \bar{\xi}\right) \quad , \tag{A.3}$$

which is equivalent to the integral equation in equation (A.2).

For the explicit description of the numeric implementation, we first change to the notation where $\boldsymbol{u}$ represents the vector of all the neurons in the network and W represents all the synaptic connections. The synapse matrix W has a block-wise structure then the describe the feed-forward neural network. Similarly, let $\boldsymbol{r}$ be the instantaneous firing rate of all the neurons, $\boldsymbol{\xi}$ the noise on the action neurons (everywhere zero except for the action neurons), $\boldsymbol{I}$ the vector of input (everywhere zero except for the neurons receiving input) and M the matrix containing the winner-nudges-all structure as a block. Further let for a vector $\boldsymbol{x} \in \mathbb{R}^N$ define $\operatorname{diag}(\boldsymbol{x}) \in \mathbb{R}^{N\times N}$ the matrix that contains $\boldsymbol{x}$ on the main diagonal and zero otherwise.

In this notation the energy term of the Lagrange function (equation (5.49)) becomes

$$E = \frac{1}{2}\left\|W\bar{\boldsymbol{r}} + \bar{\boldsymbol{\xi}} + \boldsymbol{I} - \boldsymbol{u}\right\|^2 \quad . \tag{A.4}$$

If we calculate dynamic equations and write it out explicitly, we obtain

$$\begin{aligned}\tau\dot{\boldsymbol{u}} = W\left(\bar{\boldsymbol{r}} + \tau\bar{\boldsymbol{r}}' \odot \dot{\boldsymbol{u}}\right) + \bar{\boldsymbol{\xi}} + \tau\dot{\bar{\boldsymbol{\xi}}} + \boldsymbol{I} + \tau\dot{\boldsymbol{I}} - \boldsymbol{u} + \\ + \bar{\boldsymbol{r}}' \odot W^T\left(\boldsymbol{u} - W\bar{\boldsymbol{r}} - \boldsymbol{I} - \bar{\boldsymbol{\xi}}\right) + \tau\left(\bar{\boldsymbol{r}}'' \odot \dot{\boldsymbol{u}}\right) \odot W^T\left(\boldsymbol{u} - W\bar{\boldsymbol{r}} - \boldsymbol{I} - \bar{\boldsymbol{\xi}}\right) + \\ + \tau\bar{\boldsymbol{r}}' \odot W^T\left(\dot{\boldsymbol{u}} - W\left(\bar{\boldsymbol{r}}' \odot \dot{\boldsymbol{u}}\right) - \dot{\boldsymbol{I}} - \dot{\bar{\boldsymbol{\xi}}}\right) + \beta M\left(\bar{\boldsymbol{r}} + \tau\bar{\boldsymbol{r}}' \odot \dot{\boldsymbol{u}}\right) \quad .\end{aligned} \tag{A.5}$$

equation (A.5) can be seen as a self-consistency equation for the time-derivative of the membrane potential $\dot{\boldsymbol{u}}$. A direct analytic solution is not feasible, but we can formulate it in form of a system of linear equations:

$$\begin{aligned}
A\boldsymbol{x} &= \boldsymbol{y} \quad \text{, and} \\
\dot{\boldsymbol{u}} &= \frac{\boldsymbol{x}}{\tau} \quad \text{, with} \\
A &= \mathbb{1} - W\mathrm{diag}\left(\bar{\boldsymbol{r}}'\right) - \mathrm{diag}\left(\bar{\boldsymbol{r}}''\right)\mathrm{diag}\left(W^T\left[\boldsymbol{u} - W\bar{\boldsymbol{r}} - \bar{\boldsymbol{\xi}} - \boldsymbol{I}\right]\right) - \\
&\quad -\mathrm{diag}\left(\bar{\boldsymbol{r}}'\right)W^T + \mathrm{diag}\left(\bar{\boldsymbol{r}}'\right)W^TW\mathrm{diag}\left(\bar{\boldsymbol{r}}'\right) - \beta M\mathrm{diag}\left(\bar{\boldsymbol{r}}'\right) \quad \text{, and} \\
\boldsymbol{y} &= W\bar{\boldsymbol{r}} + \boldsymbol{I} + \bar{\boldsymbol{\xi}} - \boldsymbol{u} + \beta M\bar{\boldsymbol{r}} + \bar{\boldsymbol{r}}' \odot W^T\left[\boldsymbol{u} - W\bar{\boldsymbol{r}} - \bar{\boldsymbol{\xi}} - \boldsymbol{I}\right] \\
&\quad + \tau\bar{\boldsymbol{r}}' \odot W^T\left(\dot{\bar{\boldsymbol{\xi}}} + \dot{\boldsymbol{I}}\right) + \tau\dot{\boldsymbol{I}} \quad .
\end{aligned} \tag{A.6}$$

Here is $\mathbb{1}$ the identity matrix. For the simulation of neural dynamics, we solve in each time-step a system of linear equations and perform an explicit Euler step

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \mathrm{d}t\frac{A^{-1}\boldsymbol{y}_n}{\tau} \quad . \tag{A.7}$$

Because solving a system of linear equations in each time-step is expensive, the numerical simulations are tedious and time-consuming. In the implementation, naturally, we do not invert the matrix A but use the built-in linear solver in the TensorFlow API (appendix B.3).

A.3.2 Construction of the winner-nudges-all matrix

We postulated the winner-nudges-all (WNA) in equation (5.50) and we have shown in lemma 8 that the corresponding generated nudging error leads to Hill-climbing on the expected reward . However, we did not give the explicit train of thought that leads to the form of the matrix, which we present here.

- We start from the fact that the theorem based on the principle of least-action generates error by nudging the membrane potential of the soma. In contrast to the supervised case, we would like the network to generate its own error. The simplest ansatz is to assume a linear error-generating process from the action neurons

$$\boldsymbol{e}_{\mathrm{WNA}} = M\boldsymbol{r} \quad . \tag{A.8}$$

- We observe that there is no notion of distance between the action neurons *a priori*. Because we use the network for classification, we do not know what the distance between classes is before learning. This should be reflected in the WNA matrix as well. Hence, we only differentiate between the self-connections and the lateral connections:

$$M = \begin{pmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{pmatrix} \quad , \tag{A.9}$$

with $M \in \mathbb{R}^{N \times N}$ where N is the number of the action neurons.

- During learning, the norm of the vector can be absorbed into the learning rate, and we merely consider the direction of the error-vector. Further, we know that the true winner neuron should be strengthened while the other neurons should be weakened. Therefore we set $a := 1$ and $b > 0$:

$$M = \begin{pmatrix} 1 & -b & \cdots & -b \\ -b & 1 & \cdots & -b \\ \vdots & \vdots & \ddots & \vdots \\ -b & -b & \cdots & 1 \end{pmatrix} \quad . \tag{A.10}$$

- Finally, the classification is only based on the relative activity of the neurons but not on their absolute activity. To keep the activity of the network stable, and to prohibit it from falling into a hyperactive or quiescent sate we require that the sum of the generated error should be zero:

$$\begin{aligned} \sum_i [M\boldsymbol{r}]_i &\overset{!}{=} 0 \\ \sum_i r_i - b(N-1)\sum_i r_i &= 0 \\ b &= \frac{1}{N-1} \quad . \end{aligned} \tag{A.11}$$

In summary, we constructed the WNA matrix:

$$M = \begin{pmatrix} 1 & -\frac{1}{N-1} & \cdots & -\frac{1}{N-1} \\ -\frac{1}{N-1} & 1 & \cdots & -\frac{1}{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{N-1} & -\frac{1}{N-1} & \cdots & 1 \end{pmatrix} \quad . \tag{A.12}$$

A.3.3 Source code examples

The source code is available upon request from the author or from the computational neuroscience group in the Department of Physiology at the University of Bern. At the time of writing, the repository `https://github.com/unibe-cns/lagrange_reinforcement` was not yet published. It was written using the TensorFlow API (appendix B.3), which is an easy way to execute the implemented simulation on GPUs. Below, we first show the setup of the simulation. The code is written in a modular way to enable the exchange of the aspects, such as the activation function.

```python
def setUpNetwork(self):
    """
        Set up the network with the lagrange dynamics
    """
    # Set up the network structure
    self.W = lagrangeRL.tools.networks.feedForwardWtaReadout(
        self.layers,
        self.wtaStrength,
        offset=self.initWeightMean,
        noiseMagnitude=self.initWeightWidth,
        noWtaMask=True,
        fixedPatternNoiseSigma=self.fixedPatternNoiseSigma)
    self.logger.debug('The w matrix as it \
        comes from the tool function: {}'.format(self.W.data))

    # Create the underlying network
    self.simClass = lagrangeRL.network.lagrangeTfDirect()
    self.simClass.setPlasticSynapses(np.logical_not(self.W.mask))
    self.simClass.setLearningRate(self.learningRate)
    self.simClass.setTimeStep(self.timeStep)
    self.simClass.setTau(self.tau)
    self.simClass.addMatrix(self.W)
    self.simClass.setTauEligibility(self.tauElig)
    if self.saveOnlyReward:
        self.simClass.saveTraces(False)
    else:
        self.simClass.saveTraces(True)

    # Set the weights resulting from te cost term
    self.simClass.setCostWeightings(self.alphaWna,
                                    self.alphaNoise,
                                    self.beta)

    # Define the fixed weights
    # here the wna network has to stay fixed
    wMaxFixed = np.zeros((self.N, self.N))
    wMaxFixed[-self.layers[-1]:, -self.layers[-1]:] = 1
    self.simClass.setFixedSynapseMask(wMaxFixed.astype(bool))

    # set the regularization parameters
    self.simClass.setRegParameters(self.uTarget,
                                   self.learningRateH,
                                   self.uLow,
                                   self.uHigh,
                                   self.learningRateB)

    # set up the noise
    self.simClass.setNoiseParameter(0.,
                                          self.noiseStd,
                                          self.noiseAutoCorrTime)
    self.simClass.calcWnoWta(self.layers[-1])
    self.simClass.calcOnlyWta(self.layers[-1])

    # set the biases in the network
    biasVector = np.zeros(sum(self.layers))
    biasVector[-self.layers[-1]:] = 0.5
    self.simClass.setBias(biasVector)
```

The dynamics of the network are simulated in TensorFlow. In TensorFlow, the user builds computational graphs. Propagating the differential equation by one time-step is implemented as one evaluation on the computational graph.

```
def createComputationalGraph(self):
    """
        Create the computational graph in tensorflow
    """

    #######################################
    # Define tensorflow variables for the simualtion
    self.u = tf.Variable(np.zeros(self.N), dtype=self.dtype)
    self.rLowPass = tf.Variable(np.zeros(self.N), dtype=self.dtype)
    uNoise = tf.Variable(np.zeros(self.N), dtype=self.dtype)
    self.uNoiseLowPass = tf.Variable(np.zeros(self.N), dtype=self.dtype)
    self.uDotOld = tf.Variable(np.zeros(self.N), dtype=self.dtype)
    self.eligNow = tf.Variable(
        np.zeros((self.N, self.N)), dtype=self.dtype)
    self.eligibility = tf.Variable(
        np.zeros((self.N, self.N)), dtype=self.dtype)
    self.regEligibility = tf.Variable(
        np.zeros((self.N, self.N)), dtype=self.dtype)
    self.regEligibilityBorder = tf.Variable(
        np.zeros((self.N, self.N)), dtype=self.dtype)
    self.wTfNoWta = tf.Variable(self.WnoWta, dtype=self.dtype)
    self.wTfOnlyWta = tf.Variable(self.onlyWta, dtype=self.dtype)
    inputMask = self.input.getInput(0.)[2]
    self.inputMaskTf = tf.Variable(inputMask,
                                   dtype=self.dtype)
    outputMask = self.target.getTarget(self.T)[2]
    self.outputMaskTf = tf.Variable(outputMask,
                                    dtype=self.dtype)
    self.biasTf = tf.Variable(self.biasVector, dtype=self.dtype)
    # set up a mask for the learned weights in self.wTfNoWta
    # note that W.mask must omit the WTA network
    self.wNoWtaMask = tf.Variable(self.Wplastic.astype(float),
                                  dtype=self.dtype)

    #######################################
    # Variables for debugging
    self.error = tf.Variable(np.zeros(self.N), dtype=self.dtype)


    #######################################
    # Placeholders
    # The only datatransfer between the GPU and the CPU should be the
# input to the input layer and the modulatory signal
self.inputTf = tf.placeholder(dtype=self.dtype,
                              shape=(self.N))
self.inputPrimeTf = tf.placeholder(dtype=self.dtype,
                                   shape=(self.N))
self.modulator = tf.placeholder(dtype=self.dtype,
                                shape=())

####################################
# Aux variables for the calculations
nInput = len(np.where(inputMask == 1)[0])
nOutput = len(np.where(outputMask == 1)[0])
nFull = len(inputMask)

#####################################################
# Start the actual calculations for the comp graph  #
#####################################################

####################################
# Calculate the activations functions using the updated values
self.rho = self.actFunc(self.u)
rhoPrime = self.actFuncPrime(self.u)
rhoPrimePrime = self.actFuncPrimePrime(self.u)
self.rhoOutput = self.actFunc(self.u)

###################################
# Update the exploration noise on the output neurons
uNoiseOut = tf.slice(uNoise, [nFull − nOutput], [−1])
duOutNoise = self.noiseTheta ∗ (self.noiseMean − uNoiseOut)\
          ∗ self.timeStep + self.noiseSigma ∗ \
    np.sqrt(self.timeStep) ∗ \
    tf.random_normal([nOutput], mean=0., stddev=1.0, dtype=self.dtype)
updateNoise = tf.scatter_update(uNoise,
                                np.arange(nFull − nOutput, nFull),
                                uNoiseOut + duOutNoise)
# Update the low−pass noise
with tf.control_dependencies([updateNoise]):
    self.uNoiseLowPass = self.uNoiseLowPass + (self.timeStep/self.tau)\
                         ∗ (uNoise − self.uNoiseLowPass)

####################################
# Calculate the updates for the membrane potential and for the
# eligibility trace
with tf.control_dependencies([self.uNoiseLowPass,
                              updateNoise,
                                    rhoPrime,
                                    rhoPrimePrime]):

        # frequently used tensors are claculated early on
        wNoWtaT = tf.transpose(self.wTfNoWta)
        wNoWtaRho = tfTools.tf_mat_vec_dot(self.wTfNoWta, self.rho)
        c = tfTools.tf_mat_vec_dot(wNoWtaT, self.u - \
            wNoWtaRho - self.biasTf - self.inputTf - self.uNoiseLowPass)

        # get the matrix side of the equation
        A1 = tf.matmul(self.wTfNoWta, tf.diag(rhoPrime))
        A2 = tf.matmul(tf.diag(c), tf.diag(rhoPrimePrime))
        A3 = tf.matmul(tf.diag(rhoPrime), wNoWtaT)
        A4 = tf.matmul(tf.matmul(tf.diag(rhoPrime),wNoWtaT),
                       tf.matmul(self.wTfNoWta, tf.diag(rhoPrime)))
        A5 = self.beta * self.alphaWna *\
             tf.matmul(self.wTfOnlyWta, tf.diag(rhoPrime))
        A = self.tau*(tf.eye(self.N) - A1 - A2 - A3 + A4 - A5)

        # get the vector side of the equation
        y1 = wNoWtaRho + self.biasTf + self.inputTf + \
             self.alphaNoise * self.beta * uNoise - self.u
        y2 = self.tau * self.inputPrimeTf
        y3 = rhoPrime * c
        y4 = self.tau * rhoPrime * tfTools.tf_mat_vec_dot(
                                                 wNoWtaT,
                                                 self.inputPrimeTf)
        y5 = self.beta * self.alphaWna * tfTools.tf_mat_vec_dot(
                                     self.wTfOnlyWta, self.rho)
        y6 = rhoPrime * tfTools.tf_mat_vec_dot(
                                 wNoWtaT,
                                 uNoise - self.uNoiseLowPass)
        y = y1 + y2 + y3 - y4 + y5 - y6

        # Solve the equation for uDot
        uDiff = tf.linalg.solve(A, tf.expand_dims(y, 1))[:, 0]

    updateLowPassActivity = self.rLowPass.assign((self.rLowPass + \
        self.timeStep / self.tauEligibility * self.rho) *\
        tf.exp(-1. * self.timeStep / self.tauEligibility))

    self.eligNowUpdate = self.eligNow.assign(
        tfTools.tf_outer_product(
            self.u - tfTools.tf_mat_vec_dot(self.wTfNoWta, self.rho)\
                   - self.biasTf, self.rho))
```

```
errorUpdate = self.error.assign(self.u - \
    tfTools.tf_mat_vec_dot(self.wTfNoWta, self.rho) \
    - self.biasTf - self.inputTf)

# porpagate the eligibility traces for the main learning
# and for the regularization terms
with tf.control_dependencies([saveOldUDot,
                              updateLowPassActivity,
                              self.eligNowUpdate,
                              errorUpdate]):

    self.updateEligiblity = self.eligibility.assign(
        (self.eligibility + self.timeStep * tfTools.tf_outer_product(
            self.u - tfTools.tf_mat_vec_dot(self.wTfNoWta, self.rho)\
             - self.biasTf - self.inputTf - self.uNoiseLowPass,
             self.rho)) *\
            tf.exp(-1. * self.timeStep / self.tauEligibility)
    )

    self.updateRegEligibility = self.regEligibility.assign(
        (self.regEligibility + self.timeStep * tfTools.tf_outer_product(
            tf.nn.relu(self.uTarget - self.u),
            self.rho)) * tf.exp(-1. * self.timeStep / self.tauEligibility)
    )

    self.updateRegEligibilityBorder = self.regEligibilityBorder.assign(
        (self.regEligibilityBorder + \
            self.timeStep * tfTools.tf_outer_product(
            tf.nn.relu(self.uLow - self.u) -
            tf.nn.relu(self.u - self.uHigh),
            self.rho)) * tf.exp(-1. * self.timeStep / self.tauEligibility)
    )

with tf.control_dependencies([saveOldUDot,
                              updateLowPassActivity,
                              self.eligNowUpdate,
                              errorUpdate,
                              self.updateEligiblity,
                              self.updateRegEligibility,
                              self.updateRegEligibilityBorder]):
    self.applyMembranePot = self.u.assign(self.u + self.timeStep * uDiff)

###############################################
## Node to update the weights of the network ##
###############################################
```

```
    self.updateW = self.wTfNoWta.assign(self.wTfNoWta + \
                   ( 1. / self.tauEligibility) * (
        self.modulator * self.learningRate * self.eligibility *\
        self.Wplastic + tf.math.abs(self.modulator) * self.learningRateH *\
        self.regEligibility * self.noWnaMask + self.learningRateB *\
        self.regEligibilityBorder * self.wNoWtaMask))
```

State-propagation in the simulation with scrambled reward. The logic of the simulation propagates the dynamics from event to event.

```
def runSimulation(self, startFrom=1):

    # fill up the event array with change input events
    self.initEvents()
    self.createInputEvent(0.0)
    counter = startFrom
    # we count the number of iterations
    # in terms of received reward
    counterReward = 0

    while self.events:

        self.logger.debug('The current events are:\
                          {}'.format(self.events))
        # pop next event
        idNext = self.getNextEvent()
        nextEvent = self.events.pop(idNext)
        timeStampNext = nextEvent[0]
        self.logger.debug('The next event with id\
                          {0} is: {1}'.format(idNext,
                                              nextEvent))

        # propagate the network to the next event
        tGlobal = self.simClass.T
        if timeStampNext > tGlobal:
            deltaTime = np.around(timeStampNext - tGlobal, decimals=1)
            self.logger.debug('The delta time to be propagated\
                              is {}'.format(deltaTime))
            self.simClass.run(deltaTime)
        self.logger.debug('The time after propagation\
                          is {}'.format(self.simClass.T))

        # apply event
        eventType = nextEvent[1]
        # The input current at the bottom of the
        # network changes
        if eventType == 'changeInput':
            self.eventChangeInput(self.simClass.T,
                                  counter < self.Niter)
            counter += 1
            self.logger.debug('changeInput event applied')
            if self.checkpointing and (counter % self.checkPerIter == 0):
                self.saveCheckpoint(counter)
                self.logger.info('Checkpoint created at\
                                 {}'.format(counter))
        # The decision of the network is obtained
        # the obtained reward is saved and the next
        # reward event is created in the background
        elif eventType == 'readOut':
            self.eventReadOut(self.simClass.T,
                              nextEvent[2])
            self.logger.debug('readOut event applied')
        # reward arrives at the network
        elif eventType == 'reward':
            self.eventReward(nextEvent[2],
                             nextEvent[3])
            # make report if applicable
            if counter % self.reportFrequency == 0:
                self.plotFinalReport()
                self.saveResults()
            self.logger.debug('Reward event applied')
            counterReward += 1
            self.logger.info('After {0} applied reward\
                the mean reward is {1}'.format(counterReward,
                                                  self.meanReward))
        else:
            self.logger.error(
                'The received event type is not in\
                 [<changeInput>, <readOut>, <reward>].\
                 Received: {}'.format(eventType))

        self.logger.debug('The events after one iteration\
         in the while loop are: {}'.format(self.events))


    self.logger.info('The simulation finished after\
                 presenting {} inputs'.format(counter))
```

B Lists

B.1 List of Figures

B.2 List of Tables

B.3 List of Software

List of the used and the mentioned software. For the specific version in the experiments see the respective parts of the appendix.

brainscales-benchmarks Collection of benchmarks for the BSS-1 system. `https://github.com/electronicvisions/brainscales-benchmarks`

NEST A simulator mainly for spiking neurons with focus on dynamics and structure of large-scale neural systems [Gewaltig and Diesmann, 2007]. `https://www.nest-simulator.org/`

nmpm_software Collection/Module of a built software necessary for using the BSS-1. The standard modules of the official project HEAD are considered as the software to given to the user. Each module contains a date-stamp of the build-date and the collection of the git-id of the underlying software repositories.

Pillow Pillow is the PIL fork by Alex Clark and Contributors. PIL was the Python Imaging Library by Fredrik Lundh and Contributors. `https://github.com/python-pillow/Pillow`

pyhmf BSS-1 specific implementation of the PyNN neural network description language. `https://github.com/electronicvisions/pyhmf`

scipy Python based ecosystem containing software for mathematics, engineering and sciences in general [Jones et al., 2001–]. `https://www.scipy.org/`

sbs Python based API for simulating LIF sampling using different backend simulators, but preferably NEST Gewaltig and Diesmann [2007]. The software is not open-source; it can be obtained upon request from the author: Oliver Breitwieser.

tensorflow Open source software tool for developing and deploying machine learning models and applications [Abadi et al., 2015]. It facilitates the implementation of machine learning models and their execution on special purpose hardware. `https://www.tensorflow.org/`

frickel-dls A gathering of convenience function, which works as low-level abstraction layer for programming the BSS-2 HICANN-DLSv2 neuromorphic chip. It was depreciated in late 2019.

Acronyms

D_{KL} Kullback-Leibler Divergence. 11, 12, 53, 64, 72

ADC Analog-to-Digital Converter. 87, 89

AdEx Adaptive Exponential Integrate-and-Fire. 43, 45

ANN artificial neural network. 37, 120, 146, 148, 150, 162, 166, 167, 170, 208

API Application Programming Interface. 40

ARM Advanced RISC Machine. 34

ASIC application-specific integrated circuit. 32, 34, 35, 83

BSS-2 BrainScaleS-2 system. 3, 37, 82, 83, 84, 85, 87, 85, 88, 89, 91, 92, 96, 98, 99, 101, 105, 106, 108, 109, 181, 190, 208, 210, 216

BSS-1 BrainScaleS-1 system. 3, 42, 43, 45, 46, 47, 49, 50, 57, 64, 65, 66, 68, 72, 75, 79, 80, 83, 84, 85, 87, 85, 88, 91, 179, 207, 210

capmem capacitive memory. 87

CD Contrastive Divergence. 54, 55, 64, 65, 66, 68, 75

CMOS complementary metal-oxide-semiconductor. 35, 45, 83, 84

COBA conductance-based. 20, 21, 43

COBA-LIF conductance-based leaky integrate-and-fire. 58, 61

CPU central processing unit. 28, 30, 32, 36, 41, 46, 84, 106, 179

CUBA current-based. 20, 21, 85, 87, 98

CUBA-LIF current-based leaky integrate-and-fire. 87

denmem dendritic membrane unit. 45, 68

DLS Digital Learning System. 84

DVS Dynamics Vision Sensor. 36

EPS Entry-Level Power Supply. 106

FG Floating Gate. 46, 47, 87

FPGA field-programmable gate array. 32, 45, 85, 92, 108, 110

GPGPU General Purpose Graphical Processing Unit. 30, 32, 36

GPU graphical processing unit. 10, 28, 30, 32, 111, 197

HICANN High Input-Count Analog Neural Network. 43, 45, 49, 68, 84, 85, 108

HICANN-DLSv2 High Input-Count Analog Neural Network Digital Learning System v2. 83, 108

IoT Internet of Things. 29, 35

LIF leaky integrate-and-fire. 18, 19, 21, 42, 43, 50, 57, 58, 61, 62, 80, 82, 85, 87, 88, 98, 103, 105, 207

MSE mean squared error. 77

PPU Plasticity Processing Unit. 84, 85, 87, 88, 89, 92, 94, 101, 106, 109, 110, 192

PrSC-Model preserved synaptic connection model. 162, 169

PSC post-synaptic current. 16, 18, 20, 21, 87, 101

PSP post-synaptic potential. 16, 18, 57, 61, 62, 87, 99

R-STDP reward-modulated spike-time-dependent plasticity. 83, 84, 91, 92, 94, 98, 108, 109

RAM random-access memory. 36

RBM Restricted Boltzmann Machine. 53, 54, 55, 61, 62

ReLu rectified linear unit. 7, 21, 148

SRAM static random-access memory. 43, 46

SSN Stochastic Sampling Network. 62, 63, 64, 66, 71, 74, 77, 78, 79, 80, 108

STDP spike-time-dependent plasticity. 25, 26, 37, 43, 88, 91, 92, 96, 98

STP short-term plasticity. 43

TPU Tensor Processing Unit. 32

USB Universal Serial Bus. 85, 92

VLSI Very Large Scale Integration. 28

WNA winner-nudges-all. 130, 131, 134, 138, 146, 155, 156, 159, 162, 164, 166, 167, 169, 170, 175, 196, 197

WTA winner-takes-all. 112, 170, 173, 174

C Publications and contributions

Peer-reviewed publications

Akos F. Kungl, Sebastian Schmitt, Johann Klähn, Paul Müller, Andreas Baumbach, Dominik Dold, Alexander Kugele, Eric Müller, Christoph Koke, Mitja Kleider, Christian Mauch, Oliver Breitwieser, Luziwei Leng, Nico Gürtler, Maurice Güttler, Dan Husmann, Kai Husmann, Andreas Hartel, Vitali Karasenko, Andreas Grübl, Johannes Schemmel, Karlheinz Meier and Mihai A. Petrovici, Accelerated Physical Emulation of Bayesian Inference in Spiking Neural Networks, *Frontiers in Neuroscience — Neuromorphic Engineering*, doi: 10.3389/fnins.2019.01201; 14 November 2019 Volume 13 pages 1201

Contribution: study design, experiments, evaluation, manuscript

The publication is discussed in detail in chapter 3.

Timo Wunderlich, **Akos F. Kungl**, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, Christian Pehle, Sebastian Billaudelle, Gerd Kiene, Christian Mauch, Johannes Schemmel, Karlheinz Meier and Mihai A. Petrovici, Demonstrating Advantages of Neuromorphic Computation: A Pilot Study, *Frontiers in Neuroscience — Neuromorphic Engineering*, doi: 10.3389/fnins.2019.00260; 26 March 2019 Volume 13 pages 260

Contribution: study design, manuscript

The publication is discussed in detail in chapter 4.

Dominik Dold, Ilja Bytschok, **Akos F. Kungl**, Andreas Baumbach, Oliver Breitwieser, Walter Senn, Johannes Schemmel, Karlheinz Meier and Mihai A. Petrovici, Stochasticity from function Why the Bayesian brain may need no noise, *Neural Networks*, doi: 10.1016/j.neunet.2019.08.002; November 2019, Volume 119, Pages 200-213

Contribution: discussions on study design, experiments on neuromorphic hardware

The publication is not included in the thesis, but it is cited.

Preprints

Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Dominik Dold, Julian Göltz, **Akos F. Kungl**, Timo C. Wunderlich, Andreas Hartel, Eric Müller, Oliver Breitwieser, Christian Mauch, Mitja Kleider, Andreas Grübl, David Stöckel, Christian Pehle, Arthur Heimbrecht, Philipp Spilger, Gerd Kiene, Vitali Karasenko, Walter Senn, Mihai A. Petrovici, Johannes Schemmel, Karlheinz Meier, Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate, *arXiv preprint*, 2019, arXiv:1912.12980

Contribution: implementing spike-based Bayesian inference on the BSS-2 neuromorphic system, contribution to the manuscript

Submitted and accepted to the proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), but at the time of writing not yet published.

The publication is not included in the thesis, but it is cited.

Julian Göltz, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Dominik Dold, Laura Kriener, **Akos F. Kungl**, Walter Senn, Johannes Schemmel, Karlheinz Meier, Mihai A. Petrovici, Fast and deep neuromorphic learning with time-to-first-spike coding, *arXiv preprint*, 2019, arXiv:1912.11443

Contribution: discussion on study design, contribution to the manuscript, contribution to the calculations

The publication is not included in the thesis, but it is cited.

Selected conference contributions

Timo Wunderlich, **Akos F. Kungl**, Eric Müller, Johannes Schemmel, Mihai Petrovici, Brain-Inspired Hardware for Artificial Intelligence: Accelerated Learning in a Physical-Model Spiking Neural Network, 09 September 2019, *In:* Artificial Neural Networks and Machine Learning – ICANN 2019: Theoretical Neural Computation, *Springer International Publishing*, pages 119–122, ISBN: 978-3-030-30487-4

Contribution: study design, manuscript

The publication is the conference contribution version of Wunderlich et al. [2019]. It was presented at the International Conference on Artificial Neural Networks (ICANN) 2019. It is discussed in detail in chapter 3.

Dominik Dold, **Akos F. Kungl**, João Sacramento, Mihai A Petrovici, Kaspar Schindler, Jonathan Binas, Yoshua Bengio, Walter Senn, Lagrangian dynamics of dendritic microcircuits enables real-time backpropagation of errors, Computational and Systems Neuroscience (Cosyne) 2019; Lisbon, Portugal

Contribution: contribution to the calculations; study design, calculations and experiments for the reinforcement learning part

The conference contribution is on the biologically plausible backpropagation in the supervised and reinforcement learning paradigm. It is discussed in detail in chapter 5.

D Bibliography

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 *European Solid-State Circuits Conference, ESSCIRC Conference 2016: 42nd*, pages 71–74. IEEE, 2016.

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*, 65(12): 4299–4312, 2018.

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

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. In *Readings in Computer Vision*, pages 522–533. Elsevier, 1987.

Hannelore Aerts, Wim Fias, Karen Caeyenberghs, and Daniele Marinazzo. Brain networks under attack: robustness properties and the impact of lesions. *Brain*, 139(12):3063–3083, 2016.

Laurence Aitchison and Máté Lengyel. The hamiltonian brain: efficient probabilistic inference with excitatory-inhibitory neural circuit dynamics. *PLoS computational biology*, 12(12):e1005186, 2016.

Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

Omid Alemi, William Li, and Philippe Pasquier. Affect-expressive movement generation with factored conditional restricted boltzmann machines. In *2015 International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 442–448. IEEE, 2015.

Yali Amit. Deep learning with asymmetric connections and hebbian updates. *Frontiers in computational neuroscience*, 13:18, 2019.

Ignacio Arganda-Carreras, Verena Kaynig, Curtis Rueden, Kevin W Eliceiri, Johannes Schindelin, Albert Cardona, and H Sebastian Seung. Trainable weka segmentation: a machine learning tool for microscopy pixel classification. *Bioinformatics*, 33(15):2424–2426, 2017.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Coryn AL Bailer-Jones. *Practical Bayesian Inference*. Cambridge University Press, 2017.

Andreas Baumbach. Magnetic phenomena in spiking neural networks. *MSc thesis, Heidelberg University — Kirchhoff Institute for Physics*, 2016.

Andreas Baumbach. *Working title: Neurons like magnets*. PhD thesis, Heidelberg University — Kirchhoff Institute for Physics, 2020 in preparation.

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.

Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 787–797, 2018.

Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560, 2013.

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, 2014.

Pietro Berkes, Gergő Orbán, Máté Lengyel, and József Fiser. Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. *Science*, 331(6013):83–87, 2011.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In *Advances in Neural Information Processing Systems*, pages 5941–5950, 2018.

Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience*, 18(24):10464–10472, 1998.

Guo-qiang Bi and Mu-ming Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24(1):139–166, 2001.

Elie L Bienenstock, Leon N Cooper, and Paul W Munro. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982.

Sebastian Billaudelle, Benjamin Cramer, Petrovici Mihai A, Korbinian Schreiber, David Kappel, Johannes Schemmel, and Karlheinz Meier. *Structural plasticity on an accelerated analog neuromorphic hardware system.* PhD thesis, 2019a.

Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Dominik Dold, Julian Göltz, Akos F Kungl, Timo C. Wunderlich, Andreas Hartel, Eric Müller, Oliver Breitwieser, Christian Mauch, Mitja Kleider, Andreas Grübl, David Stöckel, Christian Pehle, Arthur Heimbrecht, Philipp Spilger, Vitali Kiene, Gerd abd Karasenko, Walter Senn, Mihai A Petrovici, Johannes Schemmel, and Karlheinz Meier. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. *arXiv preprint; arXiv:1912.12980*, 2019b.

Christopher M Bishop. *Pattern recognition and machine learning.* Springer, 2006.

Hermann Blum, Alexander Dietmüller, Moritz Milde, Jörg Conradt, Giacomo Indiveri, and Yulia Sandamirskaya. A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor. *Robotics Science and Systems, RSS 2017*, 2017.

Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. Neuromorphic hardware learns to learn. *Frontiers in neuroscience*, 13, 2019.

BrainScaleS. Brainscales - brain-inspired multiscale computation in neuromorphic hybrid systems, 2011. URL `http://brainscales.kip.uni-heidelberg.de/`. Accessed: 2019-07-30.

Oliver Breitwieser. Towards a neuromorphic implementation of spike-based expectation maximization. *Masterarbeit, Universität Heidelberg.(Cited on page 52.)*, 2015.

Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642, 2005.

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

Nicolas Brunel and Mark CW van Rossum. Quantitative investigations of electrical nerve excitation treated as polarization. *Biological Cybernetics*, 97(5):341–349, 2007. Translation of Lapisque 1907.

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.

Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS computational biology*, 7(11):e1002211, 2011.

Ilja Bytschok, Dominik Dold, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Spike-based probabilistic inference with correlated noise. In *BMC Neuroscience 2017*, volume 18, page P200. Organization for Computational Neurosciences, 2017.

Fuxi Cai, Justin M Correll, Seung Hwan Lee, Yong Lim, Vishishtha Bothra, Zhengya Zhang, Michael P Flynn, and Wei D Lu. A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations. *Nature Electronics*, 2(7):290–299, 2019.

Lili Cai, Katherine Pizano, Gregory Gundersen, Cameron Hayes, Weston Fleming, and Ilana B Witten. Distinct signals in medial and lateral vta dopamine neurons modulate fear extinction at different times. *bioRxiv*, 2020.

Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019a.

Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Rev. Mod. Phys.*, 91:045002, Dec 2019b. doi: 10.1103/RevModPhys.91.045002. URL `https://link.aps.org/doi/10.1103/RevModPhys.91.045002`.

M Cartiglia, R Kreiser, and Y Sandamirskaya. A neuromorphic approach to path integration: a head direction spiking neural network with visually-driven reset. *IEEE Synposium for Circuits and Systems, ISCAS*, 2018.

Yao-Feng Chang, Burt Fowler, Ying-Chen Chen, Fei Zhou, Chih-Hung Pan, Ting-Chang Chang, and Jack C Lee. Demonstration of synaptic behaviors and resistive switching characterizations by proton exchange reactions in silicon oxide. *Scientific reports*, 6:21268, 2016.

Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

Ran Cheng, Uday S Goteti, and Michael C Hamilton. Superconducting neuromorphic computing using quantum phase-slip junctions. *IEEE Transactions on Applied Superconductivity*, 29(5):1–5, 2019.

Dante R Chialvo. Emergent complex neural dynamics. *Nature physics*, 6(10):744, 2010.

Chris73. sketch of an action potential, 2007. URL `FileURL:https://upload.wikimedia.org/wikipedia/commons/4/4a/Action_potential.svg`. accessed 2020-01-31, Licence: CC BY-SA at https://creativecommons.org/licenses/by-sa/3.0/, Original by en:User:Chris 73, updated by en:User:Diberri, converted to SVG by tiZom.

Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

Clipart. Clipart library. `http://http://clipart-library.com/`, 2020. Accessed: 2020-01-22.

Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

Benjamin Cramer, David Stöckel, Markus Kreft, Johannes Schemmel, Karlheinz Meier, and Viola Priesemann. Control of criticality and computation in spiking neuromorphic networks with plasticity. *arXiv preprint arXiv:1909.08418*, 2019a.

Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking datasets for the systematic evaluation of spiking neural networks. *arXiv preprint arXiv:1910.07407*, 2019b.

Jan Salomon Cramer. The origins of logistic regression. 2002.

Stefanie Czischek, Martin Gärttner, and Thomas Gasenzer. Quenches near ising quantum criticality as a challenge for artificial neural networks. *Physical Review B*, 98(2):024311, 2018.

Henry Hallett Dale. *Adventures in Physiology: with excursions into autopharmacology*. Pergamon Press, 1953.

Mike Davies. Benchmarks for progress in neuromorphic computing. *Nature Machine Intelligence*, 1(9):386–388, 2019.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

Andrew P Davison, Daniel Brüderle, Jochen M Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2:11, 2009.

Peter Dayan and Laurence F Abbott. Theoretical neuroscience: computational and mathematical modeling of neural systems. 2001.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Nicolas Deperrois, Victoria Moiseeva, and Boris Gutkin. Minimal circuit model of reward prediction error computations and effects of nicotinic modulations. *Frontiers in neural circuits*, 12:116, 2019.

Guillaume Desjardins, Aaron Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Parallel tempering for training of restricted boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 145–152. MIT Press Cambridge, MA, 2010.

Alain Destexhe, Michael Rudolph, and Denis Paré. The high-conductance state of neocortical neurons in vivo. *Nature reviews neuroscience*, 4(9):739, 2003.

Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

Dominik Dold. *Harnessing function from form: towards bio-inspired artificial intelligence in neuronal substrates*. PhD thesis, Kirchhoff Institute for Physics, Heidelberg University, 2020. at the time of writing, this thesis has been submitted but not yet published.

Dominik Dold, Ilja Bytschok, Akos F Kungl, Andreas Baumbach, Oliver Breitwieser, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Stochasticity from functionwhy the bayesian brain may need no noise. *Neural Networks*, 119:200–213, 2019.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

Kenji Doya, Shin Ishii, Alexandre Pouget, and Rajesh PN Rao. *Bayesian brain: Probabilistic approaches to neural coding*. MIT press, 2007.

Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11:2007, 2007.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in neural information processing systems*, pages 472–478, 2001.

Gerrit A Ecke, Sebastian A Bruijns, Johannes Hölscher, Fabian A Mikulasch, Thede Witschel, Aristides B Arrenberg, and Hanspeter A Mallot. Sparse coding predicts optic flow specificities of zebrafish pretectal neurons. *Neural Computing and Applications*, pages 1–10, 2019.

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.

Samuel Frederick Edwards and Phil W Anderson. Theory of spin glasses. *Journal of Physics F: Metal Physics*, 5(5):965, 1975.

Electronic Vision(s). `https://github.com/electronicvisions/gcc`, 2017.

Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2004.

Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual international symposium on computer architecture (ISCA)*, pages 365–376. IEEE, 2011.

Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. From the cover: Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences of the United States of America*, 113(41):11441, 2016.

Michael A. Farries and Adrienne L. Fairhall. Reinforcement Learning With Modulated Spike TimingDependent Synaptic Plasticity. *Journal of Neurophysiology*, 98 (6):3648–3665, 12 2007. ISSN 0022-3077. doi: 10.1152/jn.00364.2007.

Jannik Fehre. The effect of asymmetric weight variability on sampling processes based on boltzmann machines in neuromorphic hardware applications. Heidelberg, 2017. Heidelberg University. Bachelor's Thesis.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

J Feldmann, N Youngblood, CD Wright, H Bhaskaran, and WHP Pernice. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, 569(7755):208, 2019.

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.

Ila R Fiete and H Sebastian Seung. Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical review letters*, 97(4):048104, 2006.

Gabriel A Fonseca Guerra and Steve B Furber. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience*, 11:714, 2017.

David J Foster. Replay comes of age. *Annual review of neuroscience*, 40:581–602, 2017.

E Paxon Frady and Friedrich T Sommer. Robust computation with rhythmic spike patterns. *Proceedings of the National Academy of Sciences*, 116(36):18050–18059, 2019.

N. Frémaux, H. Sprekeler, 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.

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.

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.

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, 2017.

Johannes Friedrich, Robert Urbanczik, and Walter Senn. Spatio-temporal credit assignment in neuronal population learning. *PLoS computational biology*, 7(6): e1002092, 2011.

Steve Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001, 2016.

Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2012.

Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

Francesco Galluppi, Sergio Davies, Alexander Rast, Thomas Sharp, Luis A Plana, and Steve Furber. A hierachical configuration system for a massively parallel neural hardware platform. In *Proceedings of the 9th conference on Computing Frontiers*, pages 183–192, 2012.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

Wulfram Gerstner and Werner M Kistler. Mathematical formulations of hebbian learning. *Biological cybernetics*, 87(5-6):404–415, 2002a.

Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002b.

Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12, 2018.

Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

Julian Göltz, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Dominik Dold, Laura Kriener, Akos F. Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Fast and deep neuromorphic learning with time-to-first-spike coding. *arXiv preprint; arXiv:1912.11443*, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied IntroductionC*. Pearson, 2003. ISBN 978-0201726343. Fifth Edition.

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.

Stefan Habenschuss, Johannes Bill, and Bernhard Nessler. Homeostatic plasticity in bayesian spiking networks as expectation maximization with posterior constraints. In *Advances in Neural Information Processing Systems*, pages 773–781, 2012.

Stefan Habenschuss, Helmut Puhr, and Wolfgang Maass. Emergence of optimal decoding of population codes through stdp. *Neural computation*, 25(6):1371–1407, 2013.

Ralf M Haefner, Pietro Berkes, and József Fiser. Perceptual decision-making as probabilistic inference by neural sampling. *Neuron*, 90(3):649–660, 2016.

Julia J Harris, Renaud Jolivet, Elisabeth Engl, and David Attwell. Energy-efficient information transfer by visual pathway synapses. *Current Biology*, 25(24):3151–3160, 2015.

Jennifer Hasler. Opportunities in physical computing driven by analog realization. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene-rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielc-santos, 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.

Donald O Hebb. *The organization of behavior*. LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS, 1949. ISBN 0-8058-4300-0. This edition was published in 2009.

Moritz Helmstaedter. The mutual inspirations of machine learning and neuroscience. *Neuron*, 86(1):25–28, 2015.

Guillaume Hennequin, Laurence Aitchison, and Máté Lengyel. Fast sampling for bayesian inference in neural circuits. *arXiv preprint arXiv:1404.3521*, 2014.

Geoffrey E Hinton. Machine learning for neuroscience. *Neural systems & circuits*, 1 (1):12, 2011.

Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

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, 2013.

Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

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.

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.

Josef Honerkamp. *Stochastic dynamical systems: concepts, numerical methods, data analysis*. John Wiley & Sons, 1993.

Bernd Illing, Wulfram Gerstner, and Johanni Brea. Biologically plausible deep learningbut how far can we go with shallow networks? *Neural Networks*, 118: 90–101, 2019.

Giacomo Indiveri and Timothy K Horiuchi. Frontiers in neuromorphic engineering. *Frontiers in neuroscience*, 5:118, 2011.

Giacomo Indiveri, Elisabetta Chicca, and Rodney J Douglas. A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE transactions on neural networks*, 17(1), 2006.

Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.

iniVation. inivation website, 2020. URL `https://inivation.com/`. Accessed: 2020-03-05.

Intel. Habana labs, 2019. URL `https://habana.ai/`. Accessed: 2020-02-16.

Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971.

E. M. Izhikevich. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex*, 17(10):2443–2452, 10 2007a. ISSN 1047-3211. doi: 10.1093/cercor/bhl152.

Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007b.

Quasar Jarosz. sketch of a neuron, 2009. URL `FileURL:https://upload.wikimedia.org/wikipedia/commons/b/bc/Neuron_Hand-tuned.svg`. accessed 2020-01-31, Licence: CC BY-SA at https://creativecommons.org/licenses/by-sa/3.0/.

Sebastian Jeltsch. *A scalable workflow for a configurable neuromorphic platform*. PhD thesis, 2014.

Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in neuroscience*, 10: 118, 2016.

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.

Jakob Jordan, Mihai A Petrovici, Oliver Breitwieser, Johannes Schemmel, Karlheinz Meier, Markus Diesmann, and Tom Tetzlaff. Deterministic networks for probabilistic computing. *Scientific reports*, 9(1):1–17, 2019.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

Eric R Kandel, James H Schwartz, Thomas M Jessell, Department of Biochemistry, Molecular Biophysics Thomas Jessell, Steven Siegelbaum, and AJ Hudspeth. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

Seymour S Kety. The general metabolism of the brain in vivo. *Metabolism of the nervous system*, pages 221–237, 1957.

Hassan N Khan, David A Hounshell, and Erica RH Fuchs. Science and research policy at the end of moore's law. *Nature Electronics*, 1(1):14–21, 2018.

Mitja Kleider. *Neuron Circuit Characterization in a Neuromorphic System*. PhD thesis, 2017.

Christoph Koke. *Device variability in synapses of neuromorphic circuits*. PhD thesis, 2017.

Harold Köndgen, Caroline Geisler, Stefano Fusi, Xiao-Jing Wang, Hans-Rudolf Lüscher, and Michele Giugliano. The dynamical response properties of neocortical neurons to temporally modulated noisy inputs in vitro. *Cerebral cortex*, 18 (9):2086–2097, 2008.

Alexander Kononov. *Testing of an analog neuromorphic network chip*. PhD thesis, Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2011. HD-KIP-11-83, 2011.

A. Korcsak-Gorzo, L. Leng, A. Baumbach, J. Breitwieser, S. van Albada, W. Senn, K. Meier, and M. A. Petrovici. Spike-based tempering in neural networks. 2020.

Raphaela Kreiser, Timoleon Moraitis, Yulia Sandamirskaya, and Giacomo Indiveri. On-chip unsupervised learning in winner-take-all networks of spiking neurons. In *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Akos Kungl. Sampling with leaky integrate-and-fire neurons on the hicannv4 neuromorphic chip. *Masterarbeit, Universität Heidelberg*, 2016.

Akos Ferenc Kungl, Sebastian Schmitt, Johann Klähn, Paul Müller, Andreas Baumbach, Dominik Dold, Alexander Kugele, Eric Müller, Christoph Koke, Mitja Kleider, et al. Accelerated physical emulation of bayesian inference in spiking neural networks. *Frontiers in Neuroscience*, 13:1201, 2019.

Anna Kutschireiter, Simone Carlo Surace, Henning Sprekeler, and Jean-Pascal Pfister. Nonlinear bayesian filtering and learning: a neuronal dynamics for perception. *Scientific reports*, 7(1):8722, 2017.

Lev Davidovich Landau and Evgenii Mikhailovich Lifshitz. *Course of theoretical physics*. Elsevier, 2013.

Tor Sverre Lande, Hassan Ranjbar, Mohammed Ismail, and Yngvar Berg. An analog floating-gate memory in a standard digital technology. In *Proceedings of fifth international conference on microelectronics for neural networks*, pages 271–276. IEEE, 1996.

Benjamin James Lansdell and Konrad Paul Kording. Spiking allows neurons to estimate their causal effect. *bioRxiv*, page 253351, 2019.

Yann LeCun. 1.1 deep learning hardware: Past, present, and future. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 12–19. IEEE, 2019.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436, 2015.

Erwan Ledoux and Nicolas Brunel. Dynamics of networks of excitatory and inhibitory neurons in response to time-dependent inputs. *Frontiers in computational neuroscience*, 5:25, 2011.

Jong-Ho Lee, Sung Yun Woo, Sung-Tae Lee, Suhwan Lim, Won-Mook Kang, Young-Tak Seo, Soochang Lee, Dongseok Kwon, Seongbin Oh, Yoohyun Noh, et al. Review of candidate devices for neuromorphic applications. In *ESSDERC 2019-49th European Solid-State Device Research Conference (ESSDERC)*, pages 22–27. IEEE, 2019.

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.

Pascal Leimer, Michael Herzog, and Walter Senn. Synaptic weight decay with selective consolidation enables fast learning without catastrophic forgetting. *BioRxiv*, page 613265, 2019.

Luziwei Leng, Roman Martel, Oliver Breitwieser, Ilja Bytschok, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Spiking neurons with short-term synaptic plasticity form superior generative networks. *Scientific reports*, 8(1):10651, 2018.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Harvey S Levin, Steven Mattis, Ronald M Ruff, Howard M Eisenberg, Lawrence F Marshall, Kamran Tabaddor, Walter M High Jr, and Ralph F Frankowski. Neurobehavioral outcome following minor head injury: a three-center study. *Journal of neurosurgery*, 66(2):234–243, 1987.

Anna Levina, J Michael Herrmann, and Theo Geisel. Dynamical synapses causing self-organized criticality in neural networks. *Nature physics*, 3(12):857, 2007.

William B Levy and Robert A Baxter. Energy efficient neural codes. *Neural computation*, 8(3):531–543, 1996.

Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J Joshua Yang. Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *Journal of Physics D: Applied Physics*, 51(50):503002, 2018.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. Mapping spiking neural networks onto a manycore neuromorphic architecture. *ACM SIGPLAN Notices*, 53(4):78–89, 2018.

Chen Liu, Guillaume Bellec, Bernhard Vogginger, David Kappel, Johannes Partzsch, Felix Neumärker, Sebastian Höppner, Wolfgang Maass, Steve B Furber, Robert Legenstein, et al. Memory-efficient deep learning on a SpiNNaker 2 prototype. *Frontiers in neuroscience*, 12:840, 2018.

Jan-Peter Loock. *Evaluierung eines floating gate analogspeichers für neuronale netze in single-poly umc 180nm CMOS-prozess*. PhD thesis, Diploma thesis (English), University of Heidelberg, HD-KIP-06-47, 2006.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Zachary F Mainen and Terrence J Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503–1506, 1995.

N.G. Mankiw. *Principles of Economics*. South-Western Cengage Learning, 2012. ISBN 9780538453424.

Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 10:94, 2016.

Eve Marder and Jean-Marc Goaillard. Variability, compensation and homeostasis in neuron and network function. *Nature Reviews Neuroscience*, 7(7):563, 2006.

Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275 (5297):213–215, 1997.

Henry Markram, Maria Toledo-Rodriguez, Yun Wang, Anirudh Gupta, Gilad Silberberg, and Caizhi Wu. Interneurons of the neocortical inhibitory system. *Nature reviews neuroscience*, 5(10):793–807, 2004.

Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. A history of spike-timing-dependent plasticity. *Frontiers in synaptic neuroscience*, 3:4, 2011a.

Henry Markram, Karlheinz Meier, Thomas Lippert, Sten Grillner, Richard Frackowiak, Stanislas Dehaene, Alois Knoll, Haim Sompolinsky, Kris Verstreken, Javier DeFelipe, et al. Introducing the human brain project. *Procedia Computer Science*, 7:39–42, 2011b.

Susana Martinez-Conde, Stephen L Macknik, Xoana G Troncoso, and David H Hubel. Microsaccades: a neurophysiological analysis. *Trends in neurosciences*, 32 (9):463–475, 2009.

Timothée Masquelier. Neural variability, or lack thereof. *Frontiers in computational neuroscience*, 7:7, 2013.

Luca Mazzucato, Giancarlo La Camera, and Alfredo Fontanini. Expectation-induced modulation of metastable activity underlies faster coding of sensory stimuli. *Nature neuroscience*, 22(5):787–796, 2019.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989. ISBN 978-0201059922.

Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10): 1629–1636, 1990.

Carver Mead and Lynn Conway. *Introduction to VLSI systems*. Addison-Wesley, 1979. ISBN 978-0201043587.

Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 2019.

Roger G Melko, Giuseppe Carleo, Juan Carrasquilla, and J Ignacio Cirac. Restricted boltzmann machines in quantum physics. *Nature Physics*, page 1, 2019.

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, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

Thomas Mesnard, Gaëtan Vignoud, Joao Sacramento, Walter Senn, and Yoshua Bengio. Ghost units yield biologically plausible backprop in deep neural networks. *arXiv preprint arXiv:1911.08585*, 2019.

Risto Miikkulainen, James A Bednar, Yoonsuck Choe, and Joseph Sirosh. *Computational maps in the visual cortex*. Springer Science & Business Media, 2006.

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.

Sebastian Millner. *Development of a multi-compartment neuron model emulation*. PhD thesis, Heidelberg University — Kirchhoff Institute for Physics, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Don Monroe. Neuromorphic computing gets ready for the (really) big time, 2014.

Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

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.

Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7): 3227–3235, 2017.

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 2018a. ISSN 2162-237X. doi: 10.1109/TNNLS.2018.2826721.

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 2018b.

Eric Müller, Christian Mauch, Philipp Spilger, Oliver Julien Breitwieser, Johann Klähn, David Stöckel, Timo Wunderlich, and Johannes Schemmel. Extending BrainScaleS OS for BrainScaleS-2. *arXiv preprint; arXiv:2003.13750*, 2020a.

Eric Müller, Sebastian Schmitt, Christian Mauch, Sebastian Billaudelle, Andreas Grübl, Maurice Güttler, Dan Husmann, Joscha Ilmberger, Sebastian Jeltsch, Jakob Kaiser, Johann Klähn, Mitja Kleider, Christoph Koke, José Montes, Paul Müller, Johannes Partzsch, Felix Passenberg, Hartmut Schmidt, Bernhard Vogginger, Jonas Weidner, Christian Mayr, and Johannes Schemmel. The operating system of the neuromorphic BrainScaleS-1 system. *arXiv preprint; arXiv:2003.13749*, 2020b.

Eric Christian Müller. *Novel operation modes of accelerated neuromorphic hardware*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2015.

Richard Naud and Henning Sprekeler. Burst ensemble multiplexing: A neural code connecting dendritic spikes with microcircuits. *bioRxiv*, page 143636, 2017.

Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

Robert A Nawrocki, Richard M Voyles, and Sean E Shaheen. A mini review of neuromorphic architectures and implementations. *IEEE Transactions on Electron Devices*, 63(10):3819–3829, 2016.

Alexander Neckar, Sam Fok, Ben V Benjamin, Terrence C Stewart, Nick N Oza, Aaron R Voelker, Chris Eliasmith, Rajit Manohar, and Kwabena Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, 2018.

Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in neuroscience*, 7:272, 2014.

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

Yael Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, 2009.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

Gergő Orbán, Pietro Berkes, József Fiser, and Máté Lengyel. Neural variability and sampling-based probabilistic representations in the visual cortex. *Neuron*, 92(2):530–543, 2016.

Randall C O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8 (5):895–938, 1996.

John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library, 2007.

Bente Pakkenberg, Dorte Pelvig, Lisbeth Marner, Mads J Bundgaard, Hans Jørgen G Gundersen, Jens R Nyengaard, and Lisbeth Regeur. Aging and the human neocortex. *Experimental gerontology*, 38(1-2):95–99, 2003.

Felix Constantin Passenberg. Improving the BrainScaleS-1 place and route software towards real world waferscale experiments. Masterarbeit, Universität Heidelberg, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach,

H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Vasilis F Pavlidis, Ioannis Savidis, and Eby G Friedman. *Three-dimensional integrated circuit design*. Newnes, 2017.

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.

Bruno U Pedroni, Srinjoy Das, John V Arthur, Paul A Merolla, Bryan L Jackson, Dharmendra S Modha, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Mapping generative models onto a network of digital spiking neurons. *IEEE transactions on biomedical circuits and systems*, 10(4):837–854, 2016.

Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

Mihai A Petrovici, Johannes Bill, Ilja Bytschok, Johannes Schemmel, and Karlheinz Meier. Stochastic inference with deterministic spiking neurons. *arXiv preprint arXiv:1311.3211*, 2013.

Mihai A. Petrovici, David Stöckel, Ilja Bytschok, Johannes Bill, Thomas Pfeil, Johannes Schemmel, and Karlheinz Meier. Fast sampling with neuromorphic hardware. volume 28, 2015.

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, 2016.

Mihai A Petrovici, Sebastian Schmitt, Johann Klähn, David Stöckel, Anna Schroeder, Guillaume Bellec, Johannes Bill, Oliver Breitwieser, Ilja Bytschok, Andreas Grübl, et al. Pattern representation and recognition with accelerated analog neuromorphic systems. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017a.

Mihai A Petrovici, Anna Schroeder, Oliver Breitwieser, Andreas Grübl, Johannes Schemmel, and Karlheinz Meier. Robustness from structure: Inference with hierarchical spiking networks on analog neuromorphic hardware. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2209–2216. IEEE, 2017b.

Mihai Alexandru Petrovici. *Form Versus Function: Theory and Models for Neuronal Substrates*. Springer, 2016.

Alexander Peyser, Ankur Sinha, Stine Brekke Vennemo, Tammo Ippen, Jakob Jordan, Steffen Graber, 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.

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

Thomas Pfeil, Andreas Grübl, Sebastian Jeltsch, Eric Müller, Paul Müller, Mihai A Petrovici, Michael Schmuker, Daniel Brüderle, Johannes Schemmel, and Karlheinz Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in neuroscience*, 7:11, 2013a.

Thomas Pfeil, Anne-Christine Scherzer, Johannes Schemmel, and Karlheinz Meier. Neuromorphic learning towards nano second precision. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5. IEEE, 2013b.

Thomas Pfeil, Jakob Jordan, Tom Tetzlaff, Andreas Grübl, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Effect of heterogeneity on decorrelation mechanisms in spiking neural networks: A neuromorphic-hardware study. *Physical Review X*, 6(2):021023, 2016.

Jean-Pascal Pfister, Peter Dayan, and Máté Lengyel. Synapses with short-term plasticity are optimal estimators of presynaptic membrane potentials. *Nature neuroscience*, 13(10):1271, 2010.

Alexandre Pouget, Jeffrey M Beck, Wei Ji Ma, and Peter E Latham. Probabilistic brains: knowns and unknowns. *Nature neuroscience*, 16(9):1170, 2013.

Isabella Pozzi, Sander Bohté, and Pieter Roelfsema. A biologically plausible learning rule for deep learning in the brain. *arXiv preprint arXiv:1811.01768*, 2018.

Dimitri Probst, Mihai A Petrovici, Ilja Bytschok, Johannes Bill, Dejan Pecevski, Johannes Schemmel, and Karlheinz Meier. Probabilistic inference in discrete spaces can be implemented into networks of lif neurons. *Frontiers in computational neuroscience*, 9:13, 2015.

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, 2015.

Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, 36(6):97–110, 2019.

Pamela Reinagel and R Clay Reid. Precise firing events are conserved across neurons. *Journal of Neuroscience*, 22(16):6837–6841, 2002.

Oliver Rhodes, Luca Peres, Andrew GD Rowley, Andrew Gait, Luis A Plana, Christian Brenninkmeijer, and Steve B Furber. Real-time cortical simulation on neuromorphic hardware. *arXiv preprint arXiv:1909.08665*, 2019.

Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11):1761–1770, 2019.

Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

Andres Rodriguez, Eden Segal, Etay Meiri, Evarist Fomenko, Y Jim Kim, Haihao Shen, and Barukh Ziv. Lower numerical precision deep learning inference and training. *Intel White Paper*, 3, 2018.

Pieter R Roelfsema and Arjen van Ooyen. Attention-gated reinforcement learning of internal representations for classification. *Neural computation*, 17(10):2176–2214, 2005.

Martin Rolfs. Microsaccades: small steps on a long way. *Vision research*, 49(20):2415–2441, 2009.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, pages 348–358, 2019.

Jaldert O Rombouts, Sander M Bohte, and Pieter R Roelfsema. How attention can create synaptic tags for the learning of working memories in sequential tasks. *PLoS computational biology*, 11(3), 2015.

Frank Rosenblatt. A comparison of several perceptron models. *Self-Organizing Systems*, pages 463–484, 1962.

Tamás Roska. Cellular wave computers for nano-tera-scale technologybeyond boolean, spatial-temporal logic in million processor devices. *Electronics letters*, 43(8):427–429, 2007.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Christopher J Rozell, Don H Johnson, Richard G Baraniuk, and Bruno A Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural computation*, 20(10):2526–2563, 2008.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 533-536:533–536, 1986.

João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, pages 8721–8732, 2018.

Ruslan Salakhutdinov. Learning deep boltzmann machines using adaptive mcmc. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 943–950, 2010.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

P.A. Samuelson. *Economics*. Tata McGraw Hill, 2010. ISBN 9780070700710.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Johannes Schemmel, Johannes Fieres, and Karlheinz Meier. Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 431–438. IEEE, 2008.

Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950. IEEE, 2010.

Johannes Schemmel, Sebastian Billaudelle, Phillip Dauer, and Johannes Weis. Accelerated analog neuromorphic computing. *arXiv preprint; arXiv:2003.11996*, 2020.

Martin Schetzen. The volterra and wiener theories of nonlinear systems. 1980.

Sebastian Schmitt, Johann Klähn, Guillaume Bellec, Andreas Grübl, Maurice Guettler, Andreas Hartel, Stephan Hartmann, Dan Husmann, Kai Husmann, Sebastian Jeltsch, et al. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234. IEEE, 2017.

Michael Schmuker, Thomas Pfeil, and Martin Paul Nawrot. A neuromorphic network for generic multivariate data classification. *Proceedings of the National Academy of Sciences*, 111(6):2081–2086, 2014.

Korbinian Schreiber, Timo Wunderlich, Christian Pehle, Mihai A Petrovici, Johannes Schemmel, and Karlheinz Maier. *Closed-loop experiments on the BrainScaleS-2 architecture*. 2020. doi: 10.1145/3381755.3381776. URL `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=4013`.

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.

Wolfram Schultz. Dopamine reward prediction-error signalling: a two-component response. *Nature Reviews Neuroscience*, 17(3):183, 2016.

Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

Marc-Olivier Schwartz. *Reproducing Biologically Realistic Regimes on a Highly-Accelerated Neuromorphic Hardware System*. PhD thesis, 2013.

Walter Senn, Dominik Dold, Akos F. Kungl, Yoshua Bengio, João Sacramento, and Mihai A. Petrovici. Least action principle for real-time dendritic error-propagationacross cortical microcircuits, in preparation.

William Severa, Ojas Parekh, Kristofor D Carlson, Conrad D James, and James B Aimone. Spiking network algorithms for scientific computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

Claude E Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

Juncheng Shen, De Ma, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. Darwin: a neuromorphic hardware co-processor based on spiking neural networks. *Science China Information Sciences*, 59(2):1–5, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354, 2017.

Eero P Simoncelli, Liam Paninski, Jonathan Pillow, and Odelia Schwartz. Responses with stochastic stimuli. *The cognitive neurosciences*, page 327, 2004.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer, 2012.

Philipp Spilger. Spike-based expectation maximization on the hicann-dlsv2 neuromorphic chip. Bachelorarbeit, Universität Heidelberg, 11 2018.

Thomas Splettstoesser. sketch of a chemical synapse, 2015. URL `FileURL:https://upload.wikimedia.org/wikipedia/commons/7/70/SynapseSchematic_unlabeled.svg`. accessed 2020-01-31, Licence: CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0).

Nelson Spruston. Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience*, 9(3):206–221, 2008.

André Srowig, Jan-Peter Loock, Karlheinz Meier, Johannes Schemmel, Holger Eisenreich, Georg Ellguth, and René Schüffny. Analog floating gate memory in a 0.18 µm single-poly cmos process, 2007.

Richard M. Stallman and GCC Developer Community. *GCC 8.0 GNU Compiler Collection Internals*. 12th Media Services, 2018. ISBN 9781680921878.

Jörg Steidel. Solving map coloring problems on analog neuromorphic hardware. Bsc thesis, Heidelberg University, 2018.

Marcel Stimberg, Romain Brette, and Dan Goodman. Brian 2: an intuitive and efficient neural simulator. *BioRxiv*, page 595710, 2019.

David Stöckel. Boltzmann sampling with neuromorphic hardware. Bachelor's thesis, Heidelberg University, 2015.

Yannik Stradmann. Characterization and calibration of a mixed-signal leaky integrate and fire neuron on hicann-dls. *Bachelor thesis. Ruprecht-Karls-Universität Heidelberg*, 2016.

Yannik Stradmann. Verification and commissioning of mixed-signal neuromorphic substrates. Master, Ruprecht-Karls-Universität Heidelberg, 2019.

Piergiorgio Strata and Robin Harvey. Dale's principle. *Brain research bulletin*, 50 (5-6):349–350, 1999.

D Sulzer and S Rayport. Dale's principle and glutamate corelease from ventral midbrain dopamine neurons. *Amino acids*, 19(1):45–52, 2000.

Ilya Sutskever and Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *Artificial intelligence and statistics*, pages 548–555, 2007.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothee Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

Adam L Taylor, Timothy J Hickey, Astrid A Prinz, and Eve Marder. Structure and visualization of high-dimensional conductance spaces. *Journal of Neurophysiology*, 96(2):891–905, 2006.

Graham W Taylor and Geoffrey E Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning*, pages 1025–1032. ACM, 2009.

Tom Tetzlaff, Moritz Helias, Gaute T Einevoll, and Markus Diesmann. Decorrelation of neural-network activity by inhibitory feedback. *PLoS computational biology*, 8(8):e1002596, 2012.

Chetan Singh Thakur Thakur, Jamal Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Mark Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 12:891, 2018.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

J Vincent Toups, Jean-Marc Fellous, Peter J Thomas, Terrence J Sejnowski, and Paul H Tiesinga. Multiple spike time patterns occur at bifurcation points of membrane potential dynamics. *PLoS computational biology*, 8(10):e1002615, 2012.

Binh Tran. Demonstrationsexperimente auf neuromorpher Hardware. *Bachelor thesis (german), Universität Heidelberg*, 2013.

Martin Trepel. *Neuroanatomie: Struktur und Funktion*, volume 7. Urban & Fischer, 2017. ISBN 978-3-437-41288-2.

Misha V Tsodyks and Henry Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of sciences*, 94(2):719–723, 1997.

Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

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.

Anup Vanarse, Adam Osseiran, and Alexander Rassau. A review of current neuromorphic approaches for vision, auditory, and olfactory sensors. *Frontiers in neuroscience*, 10:115, 2016.

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.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

Christopher S von Bartheld, Jami Bahney, and Suzana Herculano-Houzel. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. *Journal of Comparative Neurology*, 524(18):3865–3895, 2016.

John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993. This is the republication of the original draft that was written 1945.

Mai-Anh T Vu, Tülay Adalı, Demba Ba, György Buzsáki, David Carlson, Katherine Heller, Conor Liston, Cynthia Rudin, Vikaas S Sohal, Alik S Widge, et al. A shared vision for machine learning in neuroscience. *Journal of Neuroscience*, 38 (7):1601–1607, 2018.

M Mitchell Waldrop. The chips are down for Moore's law. *Nature News*, 530(7589): 144, 2016.

Runchun M Wang, Chetan S Thakur, and André van Schaik. An fpga-based massively parallel neuromorphic cortex simulator. *Frontiers in neuroscience*, 12: 213, 2018.

Xiao-Jing Wang. Decision making in recurrent neuronal circuits. *Neuron*, 60(2): 215–234, 2008.

Justin Werfel, Xiaohui Xie, and H Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. In *Advances in neural information processing systems*, pages 1197–1204, 2004.

Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7(May):877–917, 2006.

James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29(5):1229–1262, 2017.

James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 2019.

Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive Behavior*, 6(2): 219–246, 1997.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Timo Wunderlich, Akos Ferenc Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, et al. Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in Neuroscience*, 13:260, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Xiaohui Xie and H Sebastian Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15(2):441–454, 2003.

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

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. In *2017 IEEE 19th Electronics Packaging Technology Conference (EPTC)*, pages 1–8. IEEE, 2017.

Acknowledgments

My deepest heartfelt appreciation goes to Isa for her love and for supporting me and bearing me during the ups and downs of the last four years.

I am indebted to my family, Melinda, Ferenc, Mami and Peti for supporting me while I am still a student for the ninth year in a row.

I would like to express my gratitude to the late Prof. Karlheinz Meier for his support and leadership during the thesis. I had the pleasure to actively work with him on the research behind Wunderlich et al. (2019). Unfortunately, he passed away in the second year of my doctor thesis.

Special thanks to Mihai Petrovici, who organized the TMA group and supported me during not only my doctoral thesis but already my master's thesis. He protected me from all the bureaucratic and financial worries; and gave valuable scientific advise.

I thank Walter Senn for the chance to work with him and his group in Bern. The still ongoing project (chapter 5) offered a lot of possibilities for curious creativity and exciting scientific discussions.

I would like to express my gratitude to Johannes Schemmel, who took over the leadership in the Electronic Vision(s) Group and organized for the continued support.

I thank Ulrich Schwarz for taking the challenging task of reading and evaluating my thesis and for his continuous support from the start of my bachelor's thesis on.

I wish to express my deepest gratitude to Manfred Stärk. He acted as the incredibly generous mecenas of the TMA group. Thanks to his support, I and my colleagues were able to conduct our research and to visit several workshops and conferences. Further, he was and is always an amazing and interested visitor and speaker at our yearly Stärk symposiums.

I would like to thank Andreas Baumbach; a.k.a. Baumie; and Dominik Dold; a.k.a. Dodo; who filled our office with life and lively discussions. We shared all the frustration and happiness of the quest for obtaining a doctoral degree.

I thank the entire Electronic Vision(s) for the lot of fun and support. I do not list your names, because I would just forget some of you and I want to thank all of you.

I owe my deepest gratitude to Jannik Fehre, Timo Wunderlich and Madison Cotteret who did their thesis under my direct supervision. They gave me the chance to discover and develop my own leadership and supervision skills.

Special thanks also to all my friends.

I appreciate the feedback offered by all my proof-readers: Hartmut, Stefff, Jose, Sebastian, Benjamin, Timo, Christian, Dodo, Baumie, Isa, Thomas, Andie Grübl, Peti, Krisztián, Kriszti, Martin, Yannik, Pascal and Eric.

No animals were harmed during the execution of the experiments and the writing of this thesis; not even in a Gedankenexperiment. The cat on figure 1.1 is called Foltán, he volunteered to become immortal by modeling for the image.

Funding statement

The research leading to these results has received funding from the European Union research and innovation programs under grant agreement no 269921 (BrainScaleS) and 604102, 720270 and 785907 (Human Brain Project, HBP) and from the Manfred Stärk Foundation.

Statement of Originality (Erklärung)

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

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

Heidelberg, den 16. März 2020 . . . .