Graduate School for Cellular and Biomedical Sciences

University of Bern

Deep Learning in Neuronal and Neuromorphic Systems

PhD Thesis submitted by
Laura Magdalena Kriener
for the degree of
PhD in Neuroscience

Supervisor
Dr. Mihai A. Petrovici
Department of Physiology
Faculty of Medicine
University of Bern

Supervisor
Prof. Dr. Walter Senn
Department of Physiology
Faculty of Medicine
University of Bern

Co-Advisor
Prof. Dr. Giacomo Indiveri
Institute of Neuroinformatics
ETH Zurich and University of Zurich

Accepted by the Faculty of Medicine, the Faculty of Science and the Vetsuisse Faculty of the University of Bern at the request of the Graduate School for Cellular and Biomedical Sciences

Bern, Dean of the Faculty of Medicine

Bern, Dean of the Faculty of Science

Bern, Dean of the Vetsuisse Faculty Bern

Abstract

The ever-increasing compute and energy requirements in the field of deep learning have caused a rising interest in the development of novel, more energy-efficient computing paradigms to support the advancement of artificial intelligence systems. Neuromorphic architectures are promising candidates, as they aim to mimic the functional mechanisms, and thereby inherit the efficiency, of their archetype: the brain. However, even though neuromorphics and deep learning are, at their roots, inspired by the brain, they are not directly compatible with each other. In this thesis, we aim at bridging this gap by realizing error backpropagation, the central algorithm behind deep learning, on neuromorphic platforms.

We start by introducing the Yin-Yang classification dataset, a tool for neuromorphic and algorithmic prototyping, as a prerequisite for the other work presented. This novel dataset is designed to not require excessive hardware or computing resources to be solved. At the same time, it is challenging enough to be useful for debugging and testing by revealing potential algorithmic or implementation flaws. We then explore two different approaches of implementing error backpropagation on neuromorphic systems. Our first solution provides an exact algorithm for error backpropagation on the first spike times of leaky integrate-and-fire neurons, one of the most common neuron models implemented in neuromorphic chips. The neuromorphic feasibility is demonstrated by the deployment on the BrainScaleS-2 chip and yields competitive results both with respect to task performance as well as efficiency. The second approach is based on a biologically plausible variant of error backpropagation realized by a dendritc microcircuit model. We assess this model with respect to its practical feasibility, extend it to improve learning performance and address the obstacles for neuromorphic implementation: We introduce the Latent Equilibrium mechanism to solve the relaxation problem introduced by slow neuron dynamics. Our Phaseless Alignment Learning method allows us to learn feedback weights in the network and thus avoid the weight transport problem. And finally, we explore two methods to port the rate-based model onto an event-based neuromorphic system.

The presented work showcases two ways of uniting the powerful and flexible learning mechanisms of deep learning with energy-efficient neuromorphic systems, thus illustrating the potential of a convergence of artificial intelligence and neuromorphic engineering research.

Contents

Chapter 1

Introduction

The last decade's tremendous progress in a variety of areas of artificial intelligence was mainly carried by the advances in the field of deep learning. The impact of deep learning on machine intelligence was first felt in computer vision (Krizhevsky et al., 2014) and spread from there to language processing and generation (Brown et al., 2020; OpenAI, 2022a), games of strategical planning (Silver et al., 2017; Vinyals et al., 2019) and photorealistic image generation (Ramesh et al., 2022; OpenAI, 2022b).

This rapid progress came at a cost: Along with their capabilities of performing ever more difficult tasks, the neural network models' complexity and their demand on computational resources have grown accordingly (Thompson et al., 2020; Schwartz et al., 2020). So far, the advancements in chip fabrication and parallelization technologies were able to provide increasingly powerful and efficient computing hardware (Leiserson et al., 2020). Moore's law, for example, famously predicts that the transistor density on chips increases exponentially with every new chip generation (Moore et al., 1965). However, as transistor dimensions are approaching the size of single atoms, the continuation of that trend is uncertain (Leiserson et al., 2020). Meanwhile, the energy efficiency and speed gains that formerly accompanied higher transistor densities have already tapered off (Bohr, 2007). On top of this, even if the exponential growth could be sustained by new fabrication technologies, the growing computational demands of deep learning have already outpaced it (Schwartz et al., 2020). Currently, this is compensated by an increased focus on parallelization techniques and highly specialized hardware systems (Thompson et al., 2020), but those results in an energy consumption that grows in accordance with the required compute. If the model complexity and sizes keep up their growth trend, this parallelization approach will neither be sufficient nor sustainable (Thompson et al., 2020; Schwartz et al., 2020).

In light of this, interest in alternative, more efficient computing paradigms is growing. Originally, the structure of deep neural networks was inspired by the brain. It therefore only seems natural to — once again — consult this archetype for efficient ways to compute with neural networks, especially since we know that the total power consumption of the brain

is around only 20 W (Sokoloff, 1960). This is orders of magnitude below what current state-of-the-art deep learning models typically consume (Economist, 2016; Deepmind, 2020).

The field of neuromorphic engineering aims at building computing hardware that is inspired by the structure and functional principles of the brain and thereby hopes to inherit the efficient mechanisms of its biological archetype (Furber, 2016). While the resulting neuromorphic platforms are quite diverse (Schuman et al., 2017), they typically have in common that computation is distributed across many small and relatively simple compute units (neurons) that are interconnected (via synapses) to form a network (Furber, 2016). The efficiency of neuromorphic systems is often rooted in two fundamental concepts, in-memory computing and event-based communication:

The neural network on a neuromorphic system serves as both the processing unit that operates on input and the memory of the system, which determines and parameterizes the computation itself. This is in accordance with the information processing in the brain, where memories and "computing algorithms" are stored in the same neural structure that is also performing the actual "computation" on the sensory data. It is however in stark contrast to the classical von-Neumann computing paradigm, which separates the computing architecture into a memory and a processing block (Von Neumann, 1945). In the latter setup both data and instructions for the processing block need to be read from the memory and transported between the blocks. This introduces a bottleneck, the von-Neumann bottleneck, and can slow down computation (Backus, 1978). By removing the separation between memory and processing unit, neuromorphic approaches promise to avoid the inefficiencies introduced by the von-Neumann bottleneck on conventional computing systems.

In addition to the low-level architectural differences, the neural networks on neuromorphic hardware platforms also differ from the artificial neural networks (ANNs) commonly used in deep learning, as the former are modelled more closely after their biological counterparts. In particular, the neurons in an ANN have no temporal dynamics, they are functions that calculate an output value when prompted with an input. Information is exchanged between the neurons via the communication of the continuously valued outputs, i.e. typically floating-point numbers. This is in stark contrast to neuromorphic and biological neurons which both have internal temporal dynamics that process input signals. Additionally, these neurons do not communicate with each other at all points in time, they only produce a short output event, a spike, when the internal dynamics fulfill certain conditions, e.g. cross a threshold (Gerstner and Kistler, 2002). Therefore, the communication between neurons is spatially and temporally sparse. This means that of a given set of neurons only a subset is actively communicating at any point in time. This sparsity in communication is believed to be one of the main reasons for the brain's energy efficiency and neuromorphic systems, by mimicking this communication scheme, attempt to inherit the efficiency (Roy et al., 2019).

The prospect of combining the powerful and flexible algorithms of deep learning with an energy-efficient neuromorphic computing system appears highly desirable (Roy et al., 2019). However, while both deep learning and neuromorphic systems draw inspiration

from the brain, for the former the level of inspiration is significantly more abstract and the two are not directly compatible. This discrepancy is rooted in the differences in neuron dynamics and in particular the neuronal communication mechanisms.
In this thesis we aim at bridging the gap between deep learning and neuromorphics by making error backpropagation, the central learning algorithm for artificial neural networks, compatible with neuromorphic neural networks. After covering the required background knowledge from the fields of computational neuroscience, neuromorphic engineering and deep learning in Chapter 2 and outlining the scientific aims of this thesis in Chapter 3, we approach this task from two angles: In Chapter 5 we employ a bottom-up, or "device-up", method, where we take common features of neuromorphic devices, in this case the leaky integrate-and-fire (LIF) neurons, as a starting point and develop a variant of the error backpropagation algorithm designed for these components. Conversely, in Chapter 6 we attempt a top-down, or "algorithm-down", approach, where we start out from an algorithm approximating error backpropagation and improve its practical feasibility and compatibility to neuromorphic systems in multiple step-wise modifications. Additionally, we introduce a tool for algorithmic and neuromorphic prototyping in Chapter 4: During our work on the topics in Chapters 5 and 6 we observed a lack of suitable small-scale benchmarks, which we solved by developing the small but challenging Yin-Yang classification task. Finally, we discuss the results obtained in this thesis in Chapter 7 and outline areas that promise further improvement for both the "device-up" and the "algorithm-down" approach.

Chapter 2

Background

The topics discussed in this thesis are located at the intersection of three fields of research: computational neuroscience, neuromorphic engineering and deep learning. In an effort to keep these introductory sections brief we only discuss those aspects of each field that are required for the understanding of the results presented in this thesis. Where possible, references to standard textbooks are included for further reading.
The research fields of computational neuroscience, neuromorphic engineering and also deep learning have in common that they are, to varying degrees, inspired by or based on the biology of the brain. Therefore, before introducing the fields in Sections 2.2 to 2.4 we first describe the biological systems they are rooted in Section 2.1.

2.1 Biological neurons and synapses

In this section we briefly describe the structure and properties of neurons and synapses. Mathematical and computational models of these biological objects will be detailed in the following sections. For a more in-depth description on the connection between the biological systems and their models we recommend the theoretical and computational neuroscience textbooks by Dayan and Abbott (2005) and Gerstner and Kistler (2002).

2.1.1 Neurons

Neurons are electrically excitable cells that communicate with each other via electrochemical signals. While the exact morphology of neurons varies widely across different neuron types and brain areas, they share a common structure illustrated in Fig. 2.1. A neuron can be roughly divided into three parts: The input signals into the neuron arrive at the dendritic tree (or dendrites). The soma integrates all inputs from the dendrites and potentially generates an output signal. When an output signal, called an action potential or spike, is generated, it travels along the axon. At the end of axon, the axon terminals, the neuron is connected to other neurons, which then receive the spike signal.

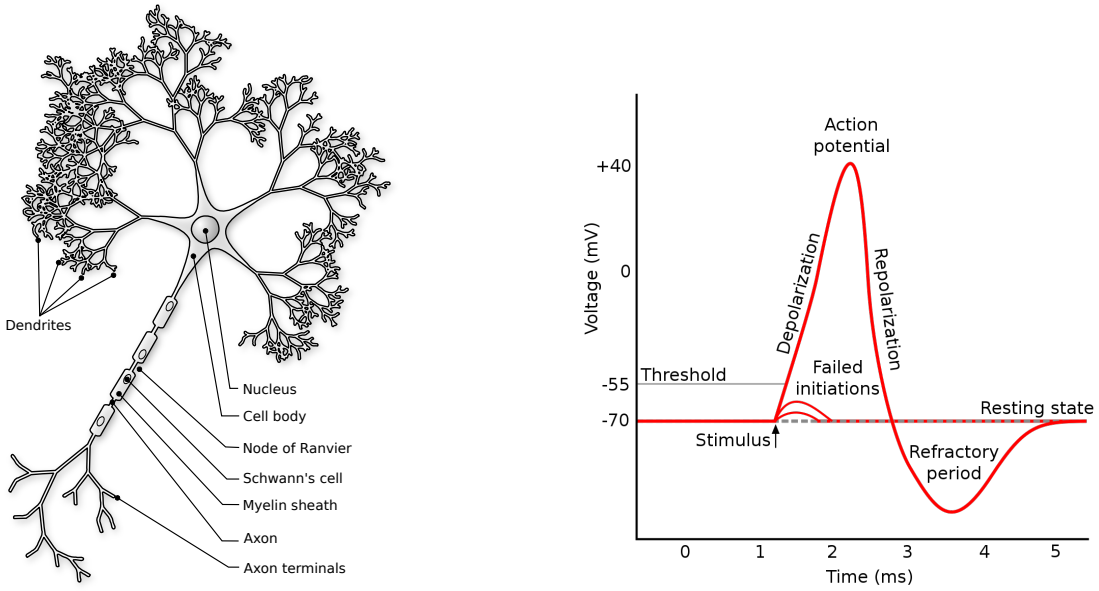

Figure 2.1: Schematic drawing of neuron morphology and action potential. Left: Drawing of a biological neuron by Rougier (2007) (published under the CC BY-SA 3.0 license). The neuron cell consists of three main parts: The dendrites or dendritic tree where input from connected neurons arrives and is transported towards the soma. The cell body or soma where all input is integrated. If the input is sufficiently strong, an action potential is generated. The action potential then travels along the axon which brings the signal to the connected downstream neurons. **Right:** Sketch of an action potential by Chris73 (2007) (published under the CC BY-SA 3.0 license). If the neuron receives a strong enough input stimulus (black arrow), it reacts with a rapid and strong depolarization followed by a quick drop in the potential. During what is called the refractory period the membrane voltage is hyperpolarized, i.e. it is below the resting potential. After the refractory period the membrane potential settles again at the resting potential.

Similar to any other cell type, a neuron is surrounded by a cell membrane which separates its interior (cytoplasm) from the outside (extra-cellular medium). The membrane is semi-permeable, which means that it lets small molecules such as water pass but stops larger molecules or ions. It acts as a capacitor which separates the charged particles inside and outside of the neuron. Even though the membrane itself is not permeable for ions, it contains transmembrane proteins called ion-channels that allow the exchange of ions between the inside and outside of the cell. Ion-channels are specific to one type of ion (e.g. sodium) and can be either passive, which means they are always open and the flow of ions through them is driven by concentration gradients, or gated by, for example, the voltage across the cell membrane. In addition to the ion-channels, ion-pumps actively transport ions across the membrane. Through an interplay between active ion-pumps, passive ion-channels and gated ion-channels, described in detail in e.g. Gerstner and Kistler (2002), neurons actively keep a concentration gradient of sodium, potassium, calcium and chlorine ions between the inside and outside. Due to this imbalance of ion concentrations, there is a charge imbalance

and a voltage across the cell membrane. In its resting state (i.e. if it receives no input) the voltage across the membrane is around −70 mV.

Action potentials

If a neuron receives input, its reaction strongly depends on the input strength. To a weak depolarizing input the membrane voltage reacts with a small deflection from the resting potential, a postsynaptic potential (PSP), that quickly decays. If however a strong enough depolarizing input is received, or multiple PSPs of weaker inputs arriving in short succession stack up to strongly depolarize the membrane potential, an action potential or spike is triggered (Fig. 2.1).
The action potential is produced by the highly non-linear dynamics of the voltage-gated ion channels in the membrane. Even though the exact shape of action potentials differs across neuron types (Bean, 2007), the main features are similar enough that we can describe a "stereotypical shape" of an action potential[1]: It starts off with a run-away mechanism in the voltage-gated sodium channels, where an increase in membrane potential causes more sodium channels to open, which increases the membrane potential even further. This causes the membrane potential to rise above 0 mV within less than 1 ms. The strong depolarization is halted by the deactivation of the sodium channels and the opening of the potassium channels which quickly repolarizes the membrane potential. This is followed by a drop below the resting potential (hyperpolarization), which we call the refractory period. During this time additional input into the neuron can not trigger another action potential. Typically, the refractory period lasts for multiple milliseconds before the membrane voltage returns to the resting potential. This mechanism was first modelled in detail by Hodgkin and Huxley (1952). As the action potential's shape is always approximately the same, all information content of the spike signal is in its timing and not in the exact shape of the potential produced.
Action potentials or spikes are the main mode of communication between neurons. Therefore, the spike signal needs to be transported from the point where it is produced to the connection points to other neurons. These are located at the ends of the axon, the axon terminals (Fig. 2.1). Action potentials are typically generated at the connection between the soma and the beginning of the axon and travel along the axon to their destination point (Clark et al., 2009). The axon consists of multiple segments which are each surrounded by a myelin sheath (formed by Schwann cells which are located around the axon). By isolating the axon from the extra-cellular medium, the myelin sheath increases the speed with which the action potential travels along the axon. The sections between the axon segments are the Ranvier nodes. They are not surrounded by a myelin sheath and reinforce the action potential signal along its route to the axon terminals.

[1] Note that there are other types of spikes with different shapes such as calcium or NMDA spikes (Larkum et al., 1999; Antic et al., 2010) which we do not address here.

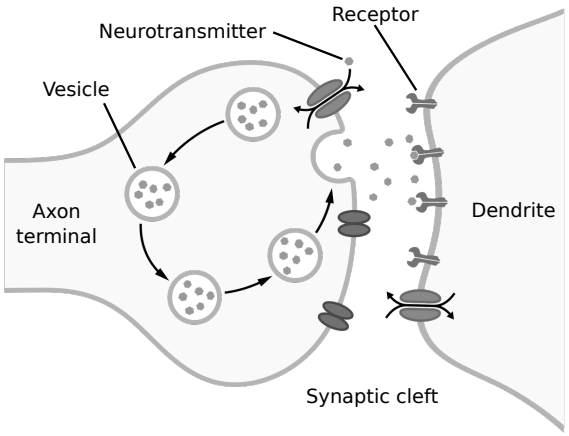

Figure 2.2: Schematic drawing of a chemical synapse. On the left the presynaptic axon terminal is shown. It releases vesicles of neurotransmitters into the synaptic cleft. The neurotransmitters travel through the synaptic cleft to the receptors on the dendrite of the postsynaptic neuron (right side). The presynaptic neuron recycles neurotransmitters from the synaptic cleft into new vesicles that can then be reused later to transmit another spike event. This figure was adapted from Splettstoesser (2015) (published under the CC BY-SA 4.0 license).

2.1.2 Synapses

The information exchange between two neurons takes place at the point of contact between the axon terminals of the one neuron and the dendrite of the other. These points are called synapses. There are chemical and electrical synapses. Electrical synapses, also called gap-junctions, allow neurons to exchange electrical signals in both directions. This is in contrast to the one-way chemical synapses. We will focus on chemical synapses in the following as they are much more common.

A chemical synapse can develop at the point where the axon terminal of one neuron connects to the dendrite of another neuron (Fig. 2.2). Synaptic connections to the soma can also occur, but they are significantly less common than dendritic synapses. The neuron sending the spike signal is called the presynaptic neuron, the receiving one is called the postsynaptic neuron. When an action potential of the presynaptic neuron arrives at the axon terminal, the presynaptic neuron releases neurotransmitters into the narrow space between the neurons. This gap between the pre- and postsynaptic neuron is called the synaptic cleft. Once the neurotransmitters reach the receptors at the postsynaptic neuron they trigger an opening of ion channels. This changes the conductance across the postsynaptic membrane and gives rise to a postsynaptic current (PSC). The PSC then in turn causes a change in the postsynaptic membrane potential, a PSP. If the resulting PSP depolarizes the neuron, we call the synapse that caused it excitatory, if the PSP hyperpolarizes, the synapse is inhibitory.

Plasticity

The strength of the impact that a presynaptic spike has on the membrane potential of the postsynaptic neuron is commonly referred to as the synaptic weight of the connection. The weight of a connection is not constant but can change over time. This is called synaptic plasticity and is believed to be fundamental to learning, memory and the development of neural circuits (Dayan and Abbott, 2005).

Synaptic plasticity can take place on multiple timescales. Short-term plasticity (STP) operates on the timescale of hundreds of milliseconds to seconds and modulates the synaptic weight via the amount of neurotransmitters that are released into the synaptic cleft during a synaptic event (Markram and Tsodyks, 1996). As it takes some time for the neuron to recycle neurotransmitters after they have been released in a synaptic event, many synaptic events occurring in short succession can deplete the amount of available neurotransmitters. Then, the amount of neurotransmitters released in the next event is lower and therefore the synaptic weight is decreased. This effect is called short-term depression. At the same time the opposite effect, short-term potentiation, is also possible. Here, the occurrence of spike events increases the calcium levels inside the presynaptic cell, which in turn increases the release probability of neurotransmitters for the next synaptic event, thereby increasing the synaptic weight. Both of these effects are non-permanent and after a few seconds without spiking activity the synaptic weight decays back to its baseline value.

As the effects of STP persist only over time intervals of a few seconds at most, other mechanisms which operate on longer timescales are more relevant for the study of task learning and memory. Pioneering theoretical work in this direction was performed by Hebb (1949). It is colloquially summarized in the famous Hebb rule "What fires together, wires together". More precisely Hebb (1949) suggests that if one neuron is frequently involved in making another neuron fire, the connection from the first neuron to the second one should be strengthened. Experimental evidence for activity dependent long-term potentiation (LTP) was found by Bliss and Lømo (1973) and shortly after for long-term depression (LTD) by Dunwiddie and Lynch (1978). Later results suggest that not only the correlation of pre- and postsynaptic firing is relevant to the weight change but also the timing of it (Bi and Poo, 1998). The effects of long-term plasticity typically persist on the order of tens of minutes or longer (Dayan and Abbott, 2005).

The most extreme form of synaptic plasticity is structural plasticity. Instead of changing the strength of existing connections in the network, structural plasticity changes the connectivity of the network itself by forming new synaptic connections and removing old ones. It is believed that structural plasticity is involved in the learning of new tasks and the recovery from injuries to the brain (Johansen-Berg, 2007).

2.2 Computational models of neurons and synapses

In computational and theoretical neuroscience we use mathematical neuron and synapse models and simulations of those models to understand the underlying principles of information processing in the brain. In the formation of a model there is always a trade-off between the mathematical/computational complexity and the level of detail with which biological mechanisms are described. There exists a very large collection of models across the whole spectrum from detailed and computationally expensive to highly abstract, simplified and computationally cheap. In this chapter we mainly highlight the ones relevant for this thesis and again refer the reader to the textbooks of Gerstner and Kistler (2002) and Dayan and Abbott (2005) for a more in-depth treatment of the subject.

2.2.1 Spiking neuron models

Spiking neuron models include, with a varying degree of abstraction, the all-or-nothing spike communication mechanism between neurons. A biophysically detailed model of spiking neurons is the Hodgkin-Huxley model, which is based on the recordings from the squid giant axon (Hodgkin and Huxley, 1952). It consists of four non-linear ordinary differential equations describing the dynamics of the membrane voltage as well as sodium and potassium ion channels. This allows it to reproduce the characteristic shape of action potentials (Fig. 2.1), as well as neuronal firing patterns such as adaptation or bursting observed in biology. However, the level of detail at which the Hodgkin-Huxley model reproduces biological phenomena comes at the cost of significant computational complexity, which makes it less suitable for the study of larger networks. For a computationally cheaper model we can leverage the fact that the actual shape of the action potential is stereotypical, which means it is always approximately the same[2]. Therefore, the most information is encoded in the timing of the action potential and not in its voltage dynamics. This allows us to not model the shape of the action potential but to treat is as an abstract event (Gerstner and Kistler, 2002, Chapter 4).

Leaky-integrate-and-fire model

One of the most commonly used neuron models which employs this method to reduce mathematical and computational complexity is the leaky integrate-and-fire (LIF) neuron model (Lapicque, 1907; Abbott, 1999; Dayan and Abbott, 2005). Here the ion channel dynamics are summarized into one passive leak conductance g_ℓ which pulls the membrane voltage u to a resting potential E_ℓ. The differential equation for the membrane voltage is

$$C_\mathrm{m}\frac{\mathrm{d}u}{\mathrm{d}t} = g_\ell\left[E_\ell - u(t)\right] + I_\mathrm{ext}(t) + I_\mathrm{syn}(t) \tag{2.1}$$

[2] Although there are slight variations across different neuron types (Bean, 2007).

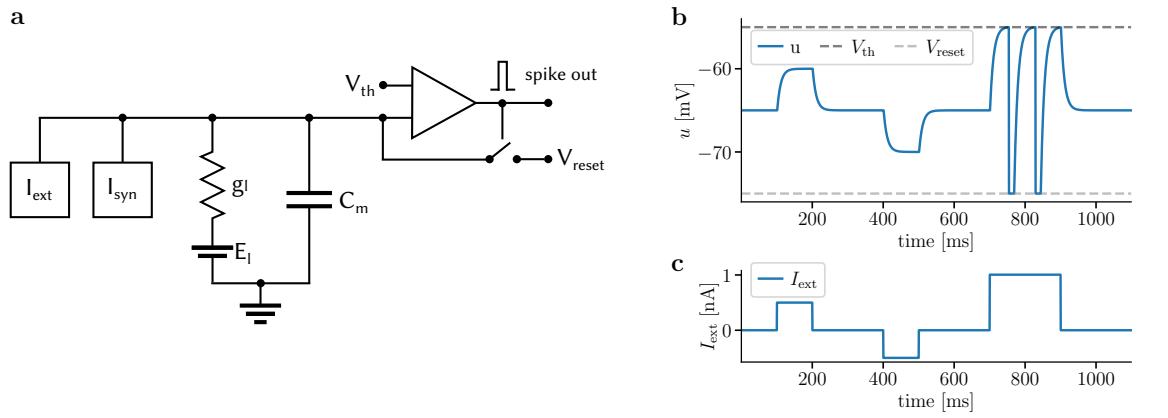

Figure 2.3: LIF neuron schematics and dynamics. (a) Equivalent circuit of the LIF neuron model. The neuron membrane is modelled as a capacitor C_m. All ion channels are summarized into one conductance g_ℓ pulling the membrane voltage u towards the leak potential E_ℓ. A comparator (triangle) compares the membrane voltage to a threshold V_th. If the membrane voltage reaches the threshold, the comparator sends out a spike signal, which triggers a resetting of the membrane to V_reset. The external input current I_ext and the synaptic input current I_syn are treated as black-box current sources here. **(b)** Membrane voltage u of an LIF neuron receiving an external input current (shown in (c)). The first current step lets the membrane rise exponentially to a new and higher membrane voltage, while the second one pulls the membrane below its resting value. The final current step is strong enough to push the membrane above the threshold. The neuron spikes two times, each spike is followed by a refractory period at V_reset. Once the current step ends, the membrane voltage decays back to its resting potential. **(c)** External current I_ext applied to the neuron. Simulation parameters can be found in Table B.1.

where C_m is the membrane capacitance and I_ext and I_syn are input currents to the cell, either from an external source or synaptic events. The action potential dynamics are replaced by a simple thresholding mechanism: If the membrane voltage crosses a threshold value V_th at the time $t = t_\mathrm{spike}$, the membrane voltage is reset to a reset potential V_reset and held at this value for the duration of the refractory period τ_ref:

$$u = V_\mathrm{reset} \quad \text{for} \quad t \in (t_\mathrm{spike}, t_\mathrm{spike} + \tau_\mathrm{ref}] \quad \text{if} \quad u(t_\mathrm{spike}) = V_\mathrm{th} \tag{2.2}$$

The output of the neuron is a sequence of spike events, each at a precise time $t_\mathrm{spike,i}$. This sequence is called a spike train $S(t)$ and can be written as a sum of delta functions

$$S(t) = \sum_{i \in \mathrm{spikes}} \delta\left(t - t_\mathrm{spike,i}\right) \ . \tag{2.3}$$

The dynamics of the LIF neuron when presented with an external step current, e.g. a patch clamp stimulus, are illustrated in Fig. 2.3 b and c. We see that a positive input current step depolarizes the membrane, while a negative current hyperpolarizes it. As long as the input is low enough that the membrane voltage stays below the spiking threshold, the membrane

trace is a low-pass filtered version of the input current (see Eqn. (2.1)). The time constant of the low-pass is given by the neuron parameters $\tau_\mathrm{m} = \frac{C_\mathrm{m}}{g_\ell}$. If the depolarizing current is strong enough, it can push the membrane above the threshold and elicit a spike followed by a reset. In addition to simulating the dynamics described in Eqns. (2.1) and (2.2) as shown in Fig. 2.3 b, we can also find an electrical circuit where the electrical quantities follow the same differential equations (Fig. 2.3 a). This is going to be of particular relevance for the discussion of mixed-signal neuromorphic hardware in Section 2.3.2.

2.2.2 Rate-based neuron models

While spiking neuron models like the LIF neuron describe the neuronal output as distinct events in time, rate-based neuron models take a simplified approach. They describe the neuronal output not as a series of events but as the firing rate: the rate at which the neuron produces output events. The firing rate r can be defined in multiple ways, the simplest of which being just the number of spikes N produced in a time interval ΔT

$$r = \frac{N}{\Delta T}\,. \tag{2.4}$$

However, rate-based neuron models typically employ a quantity called the instantaneous firing rate $r(t)$, which is given for any point in time and can not be based on a spike count in a macroscopic time interval ΔT. Instead, it is typically interpreted as the spike count during a very short (infinitesimal) interval $\mathrm{d}t$ either averaged over multiple stochastic trials or a population of neurons. For an extensive treatment of this see Dayan and Abbott (2005, Chapter 1).

The simplification of no longer treating every single output event, but summarizing them into a firing rate has both advantages and disadvantages. On the one hand a description of the neuronal output using the firing rate is not able to capture effects based on spike timing and spike correlation. On the other hand however rate-based models allow for the easier inclusion of stochasticity on a network level and reduce the required amount of simulated neurons by letting one rate-based neuron represent a population of spiking neurons (Dayan and Abbott, 2005, Chapter 7.1). Additionally, the rate-based models also improve analytical tractability as we will see in Chapter 6.

In addition to their membrane dynamics, rate-based neurons are characterized by their activation function φ which describes the firing rate of a neuron as a function of its membrane potential $r(t) = \varphi\left(u(t)\right)$. As it describes a neuronal firing rate, the activation function is typically positive everywhere and very often bounded on the upper end, as arbitrarily high firing rates are deemed unrealistic. A common choice for φ is the logistic function

$$\varphi(u) = \frac{a}{1 + \exp\left(-\frac{u-b}{c}\right)} \tag{2.5}$$

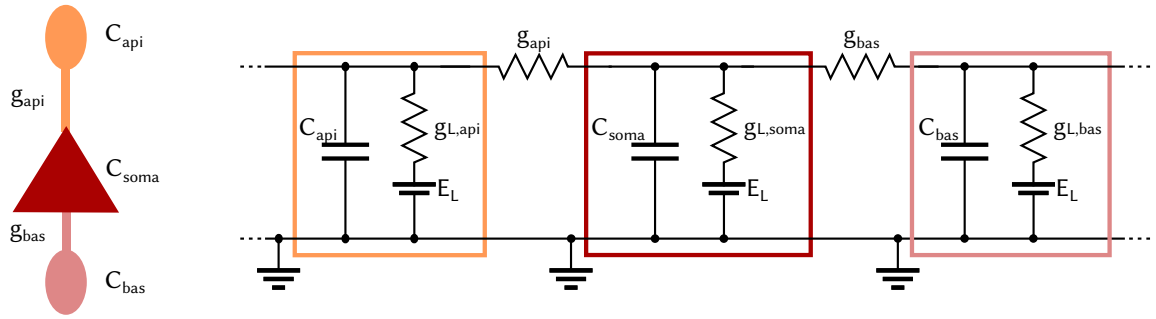

Figure 2.4: Simple multi-compartment neuron model. Left: Illustration of a neuron with 3 compartments: soma, basal dendrite and apical dendrite. Right: The three compartments are modelled by three leaky integrator circuits which are connected by the conductances g_{bas} and g_{api}. Note that for visual clarity we have left out the mechanisms to produce an output or receive input for all compartments in this drawing.

where a determines the maximum firing rate, b is a bias and c scales the steepness of the activation function.

The rate-based neuron model which we will encounter in Chapter 6 has membrane voltage dynamics that are similar to the dynamics of the previously discussed LIF neuron: The membrane voltage u also follows the leaky-integrator dynamics

$$C_{\text{m}} \frac{\mathrm{d}u}{\mathrm{d}t} = g_\ell \left[E_\ell - u(t)\right] + I(t) \tag{2.6}$$

where we have subsumed any input currents (external, synaptic or otherwise) into $I(t)$. In contrast to the LIF model however, there is no thresholding or reset mechanism. The output of the rate-based neuron is $r(t) = \varphi\left(u(t)\right)$.

2.2.3 Multi-compartment neuron models

So far we have focussed our modelling on the voltage dynamics and output signals of a neuron. There is however another (somewhat orthogonal) aspect of biology for which we have to decide on what level of detail we want to model it: the morphology of the neuron. In Sections 2.2.1 and 2.2.2 we have implicitly already chosen the strongest simplification possible by fully neglecting the fact that at different locations in the neuron, the voltage across the membrane can have different values. By describing a neuron with only one membrane voltage $u(t)$, we implicitly reduced the shape of the neuron to a single point. Neuron models employing this simplification are grouped under the term "point neuron models".

A quite flexible and commonly used method to include neuron morphology is to divide the neuron into two or more parts, which we will call compartments (Gerstner and Kistler, 2002,

Chapter 2.6). In the most basic case, we just divide the neuron into a "soma compartment" and a "dendritic compartment" and assume that the membrane voltage is approximately the same everywhere within the soma of the neuron but different from the membrane voltage in the whole dendritic tree. We model the connection between the two compartments as a constant conductance. To model the complex branching structure of the dendritic tree, we can also split up one "dendritic compartment" into multiple separated compartments which can be connected to the soma in an arbitrary tree-like fashion. The voltage dynamics of each compartment is governed by its own differential equation(s), which is coupled to the equations describing the dynamics of the neighboring compartments. For a compartment i which is modelled by leaky-integrator dynamics and connected to a set of other compartments $\mathcal{C}$ the voltage dynamics are

$$C_{\mathrm{m},i}\frac{\mathrm{d}u_i}{\mathrm{d}t} = g_{\ell,i}\left[E_{\ell,i} - u_i(t)\right] + I_{\mathrm{ext},i}(t) + I_{\mathrm{syn},i}(t) + \sum_{j\in\mathcal{C}} g_{i,j}\left[u_j(t) - u_i(t)\right] \tag{2.7}$$

where $g_{i,j}$ is the conductance between the compartments i and j. We see that the dynamics are the same as for the original leaky integrator in Eqn. (2.1) except for an added current originating from the connected compartments.

Figure 2.4 illustrates such a multi-compartment setup consisting of a somatic compartment and two dendritic compartments, the basal dendrites and the apical dendrites. Typically, in a multi-compartment neuron only the somatic compartment is able to produce an output. Whether this output is spiking or rate-based depends on the dynamics chosen for the somatic compartment. Furthermore, it is common (but not strictly necessary) that only the dendritic compartments receive synaptic input and forward it to the somatic compartment.

We will reencounter the multi-compartment neuron illustrated in Fig. 2.4 in Sacramento et al. (2018) and Chapter 6. There however, the dynamics are simplified: While we here treated the dendritic compartments as leaky-integrators with their own temporal dynamics, in Sacramento et al. (2018) the membrane voltages in the dendritic compartments are instantaneous functions of their inputs (Eqn. (6.4) to Eqn. (6.6)). Furthermore, while here each pair of connected compartments is influencing the dynamics of each other, in Sacramento et al. (2018) the dendritic compartments influence the somatic compartment but not the other way around.

2.2.4 Synapse models

In our discussions of neuron models we have so far not detailed the synaptic interactions but rather collected the overall effect of all synaptic interactions into a synaptic input current I_{syn} onto the membrane. In the following we will, for both a spiking and a rate-based scenario, detail the modelled mechanisms and the temporal dynamics of the synaptic currents.

First however, we need to distinguish between two types of synapse models: As discussed in Section 2.1.2, neurotransmitters travelling through the synaptic cleft give rise to the opening of ion-channels once they reach the postsynaptic neuron. This changes the conductance across the cell membrane. We model this as a conductance $g_{\text{syn}}(t)$ which varies over time and pulls the membrane of the neuron towards a synaptic reversal potential E_{rev}. The resulting current

$$I_{\text{syn}}(t) = g_{\text{syn}}(t)\ [E_{\text{rev}} - u(t)] \tag{2.8}$$

depends on the current state of the membrane voltage. Synaptic models that are based on a variation of a synaptic conductance are called conductance-based (CoBa) models. By assuming that any fluctuation of the membrane voltage caused by the synapse is small compared to the distance between the membrane voltage and the synaptic reversal potential, we can simplify this model and remove the dependence of I_{syn} on $u(t)$. Models using this simplification assume the impact of synaptic input not to be a change in conductance but directly a current onto the membrane $I_{\text{syn}}(t)$ and are therefore called current-based (CuBa) models.

Spike transmission

For both CoBa and CuBa synapse models we assume the impact of different synapses of a neuron as well as multiple spikes across the same synapse to sum up linearly[3]:

$$g_{\text{syn}}(t) = \sum_{i \in \text{syn}} w_{i,\text{CoBa}} \sum_{t_s \in \text{spks}(i)} \kappa\left(t - t_s\right) \quad \text{for CoBa} \tag{2.9}$$

$$I_{\text{syn}}(t) = \sum_{i \in \text{syn}} w_{i,\text{CuBa}} \sum_{t_s \in \text{spks}(i)} \kappa\left(t - t_s\right) \quad \text{for CuBa} \tag{2.10}$$

where $\kappa(t)$ is the kernel which describes the temporal shape of the synaptic interaction and $w_{i,\ldots}$ is the synaptic weight. For the CoBa case the synaptic weight is a conductance, while it is a current for the CuBa case. Note that inhibitory synapses are realized differently for the two model types: CuBa inhibitory synapses are modelled via a negative value for the synaptic weight w. For CoBa synapses the value of the weight is always positive and the distinction between excitatory and inhibitory synapses is made via different values for the reversal potential E_{rev} in Eqn. (2.8). The reversal potential E_{rev} is above the resting potential for excitatory and below for inhibitory synapses.

[3]For simplicity in the notation we have in the CoBa case assumed that either all synapses are excitatory or all are inhibitory. If that where not the case, we would need to split the sum over all incoming synapses between inhibitory and excitatory because the resulting $g_{\text{syn, exc}}(t)$ and $g_{\text{syn, inh}}(t)$ are multiplied with different reversals potential when calculating the PSC.

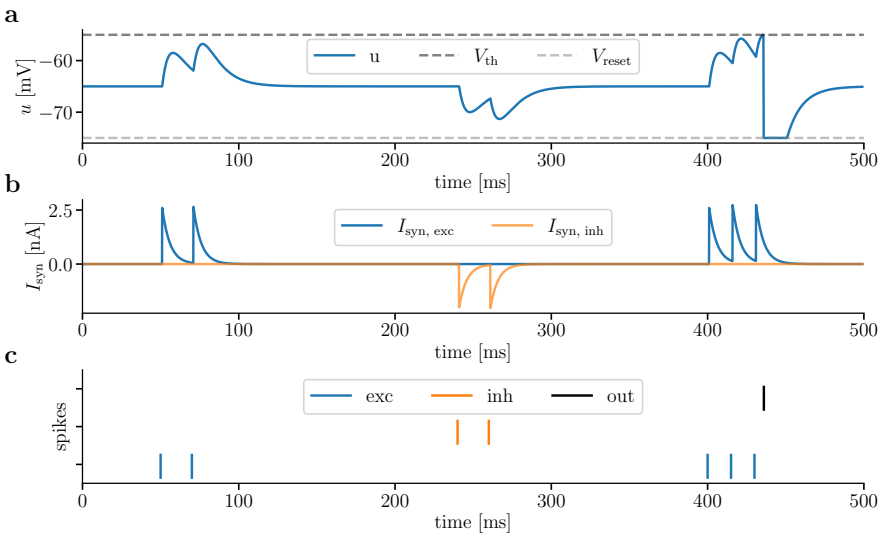

Figure 2.5: LIF neuron with current-based synaptic input. (a) Membrane voltage of an LIF neuron receiving two spike trains of excitatory and inhibitory input spikes. Each input spike causes a PSP on the membrane voltage. PSPs that are close together in time stack up. When enough input spikes arrive in short succession, the PSPs add up to a high enough value that reaches the threshold and the neuron produces and output spike. Afterwards, the membrane voltage is reset to V_{reset} and held there for the duration of τ_{ref}. **(b)** The synaptic input currents I_{syn} caused by the excitatory and inhibitory input spike train (blue and orange respectively). The synapses are current-based with an exponential kernel. **(c)** Raster plot of the spike trains: excitatory input spike train in blue, inhibitory input spike train in orange and the postsynaptic output spike in black. Simulation parameters can be found in Table B.1.

The most common kernel shapes used for describing the synaptic interaction are the

$$\kappa(t) = \delta(t) \qquad \text{delta kernel} \tag{2.11}$$

$$\kappa(t) = \Theta(t)\, \exp\left(-\frac{t}{\tau_{\text{syn}}}\right) \qquad \text{exponential kernel} \tag{2.12}$$

$$\kappa(t) = \Theta(t)\, t\, \exp\left(-\frac{t}{\tau_{\text{syn}}}\right) \qquad \text{alpha kernel} \tag{2.13}$$

where $\Theta(t)$ is the Heaviside-function and τ_{syn} is the synaptic time constant defining the timescale of the synaptic interaction.

Figure 2.5 illustrates the PSC induced by current-based synapses with an exponential kernel for a set of excitatory and inhibitory input spikes as well as the resulting PSPs on the membrane potential of an LIF neuron. The PSPs add up linearly and if the threshold voltage is reached, a postsynaptic spike is triggered. The model illustrated here, an LIF neuron with CuBa synapses with exponential kernels, is also the one employed in Chapter 5 and Section 6.5 of this thesis.

Rate transmission

For rate-based neuron models the synaptic interaction is not based on discrete events in time, but instead the instantaneous firing rate of the presynaptic neuron is sent to the post-synaptic neuron at every point in time. Again, we can make the distinction between CoBa and CuBa synapses:

$$g_{\text{syn,exc}}(t) = \sum_{i \in \text{ exc syn}} w_i \, r_i(t) \quad \text{for CoBa} \tag{2.14}$$

$$g_{\text{syn,inh}}(t) = \sum_{i \in \text{ inh syn}} w_i \, r_i(t) \quad \text{for CoBa} \tag{2.15}$$

$$I_{\text{syn}}(t) = \sum_{i \in \text{ exc + inh syn}} w_i \, r_i(t) \quad \text{for CuBa} \tag{2.16}$$

The resulting synaptic current in the CoBa case is

$$I_{\text{syn}}(t) = g_{\text{syn,exc}}(t) \, [E_{\text{rev,exc}} - u(t)] + g_{\text{syn,inh}}(t) \, [E_{\text{rev,inh}} - u(t)] \, . \tag{2.17}$$

An example of the usage of CoBa rate-based synapses can be found in Urbanczik and Senn (2014). The learning mechanisms introduced in this work form the basis for the learning in the dendritic microcircuit model in Sacramento et al. (2018) which is prominently featured in Chapter 6 of this thesis. However, the synapse model in Sacramento et al. (2018) was simplified to a rate-based CuBa model.

Plasticity

The most commonly modelled form of synaptic plasticity is long-term plasticity (Gerstner and Kistler, 2002; Dayan and Abbott, 2005). In general, we describe the change in a synaptic weight between two neurons i and j[4] with a differential equation

$$\frac{\mathrm{d}w_{ji}}{\mathrm{d}t} = f\left(\theta_i, \theta_j, w_{ji}\right) \tag{2.18}$$

where f is some function of the quantities of the neurons θ_i, θ_j and potentially of the synaptic weight itself. The shape of the function f and the quantities of the neurons that are part of the plasticity rule varies widely.
In a spike-based setting the relevant neuron quantities are typically the spike trains $S(t)$ of the neurons. The arguably most famous example for this is spike-timing-dependent plasticity (STDP), which was first experimentally discovered by Bi and Poo (1998). Here, the

[4] We choose the convention of indexing a weight as $w_{\text{post-idx, pre-idx}}$ to match the notation chosen in the publications presented in Chapters 5 and 6.

weight update is a function of the relative time differences between pre- and postsynaptic spike times.
In a rate-based setting, typically time continuous quantities, such as the firing rates $r(t)$ or the membrane voltages $u(t)$, are used. For example, the most basic form of Hebbian LTP (Hebb, 1949) can be written as

$$\frac{\mathrm{d}w_{ji}}{\mathrm{d}t} = c\, r_i(t)\, r_j(t) \tag{2.19}$$

where c is a constant and positive parameter and r_i and r_j are the pre- and postsynaptic firing rates. Other examples in this category of "Hebbian-like plasticity rules" are Oja's rule (Oja, 1982) and the BCM rule (Bienenstock et al., 1982).
More recently the category of "three-factor learning rules" has grown in popularity (Frémaux and Gerstner, 2016; Gerstner et al., 2018). In addition to the variables from the pre- and postsynaptic neuron, the weight update additionally depends on a third factor. This third factor can for example be a global modulatory signal like reward or surprise but also a neuron specific error signal. An in-depth discussion on three-factor learning rules can be found in a review on this topic by Gerstner et al. (2018). For this thesis however, it suffices to say that this category contains the Urbanczik-Senn learning rule (upon which plasticity mechanisms in Chapter 6 are based), along with many other plasticity mechanisms in models for biologically plausible error backpropagation.

2.3 Neuromorphic engineering

In this chapter we aim to provide an overview of neuromorphic engineering with a specific emphasis on the neuromorphic hardware type and specific chip that we will encounter in Chapters 5 and 6 of this thesis. As the field of neuromorphic engineering is small compared to e.g. theoretical neuroscience and at the same time is quite diverse we do not have established textbooks to refer to. However, the reviews of Indiveri et al. (2011); Furber (2016); Schuman et al. (2017); Thakur et al. (2018); Roy et al. (2019) provide a good overview on neuromorphic hardware platforms. Even though the field is developing rather rapidly and these reviews are missing some of the more recently developed hardware platforms, we will base this chapter on them. Where appropriate, we will additionally point to the newer developments throughout the following sections.
When the term neuromorphic was first coined by Carver Mead, it revolved around the idea that the characteristics of metal-oxide-semiconductor (MOS) transistors were similar to the dynamics of ion channels in neurons and that this could be used to build analog silicon neurons that mimic the behavior of biological neurons (Mead and Ismail, 1989; Mead, 1990). Nowadays, the term "neuromorphic" is used in a much broader sense and encompasses not only hardware platforms but also algorithms and sensor technologies that are in some sense inspired by the mechanisms and dynamics of the nervous system. In this thesis we will

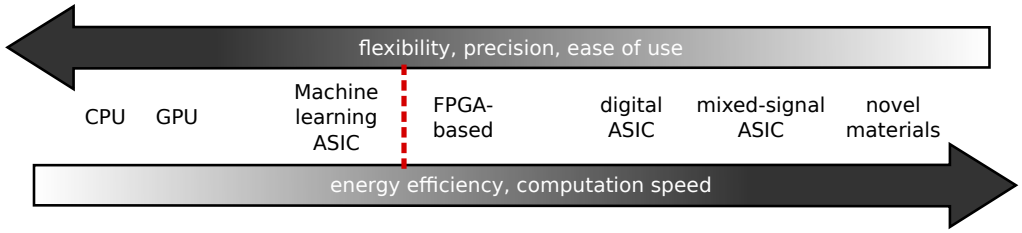

Figure 2.6: Illustration of the range of computing hardware platforms. We go from conventional computing hardware on the left towards novel (neuromorphic) computing paradigms on the right. Generally speaking the more conventional platforms have advantages in one or multiple of the areas of flexibility, precision and ease of use. Conversely, the neuromorphic hardware platforms tend to be geared towards energy efficiency and/or computation speed, while being less flexible or precise or easy to use. The red dashed line marks the area in the spectrum where often the step towards event-based (spiking) communication is made and where often the line between conventional and neuromorphic hardware is drawn.

focus on the neuromorphic computing platforms, of which there is also a great variety that we will cover in Section 2.3.1. In spite of this diversity the different approaches share the brain as a common inspiration and there are several reoccurring design ideas and concepts across different neuromorphic platforms (Schuman et al., 2017):

- In contrast to conventional von-Neumann computing hardware, which separates the memory from the processing unit (Von Neumann, 1945), the brain co-locates memory and processing in the network of neurons and synapses. By inheriting this network structure, neuromorphic platforms aim to avoid the so-called "von-Neumann bottleneck" in information processing.
- Computation is performed by a network of relatively simple compute nodes (neurons). Each compute node typically operates on locally available information. This allows for highly parallel information processing with little need for global synchronization.
- The high energy efficiency of the brain is partly credited to the use of a temporally sparse, event-based communication scheme between neurons (i.e. spikes). Neuromorphic platforms often aim to achieve good energy efficiency by copying this communication mechanism.
- Neuromorphic platforms often aim to mimic the brain's learning and adaptation mechanisms by implementing synaptic plasticity in the hope of being able to continuously adapt to changing environments or tasks.

2.3.1 Variety of neuromorphic platforms

In this section we provide an overview of various types of neuromorphic platforms that have been or are currently being developed. It can be instructive to arrange the different

types from very close to conventional computing systems towards relying on completely novel computing paradigms as done in Fig. 2.6. There, we have also included the conventional (general purpose) computing hardware of central processing units (CPUs) and graphical processing units (GPUs) as a reference. Generally speaking, more conventional hardware types are more flexible, easier to use and/or more precise, while the more novel architectures have advantages in energy efficiency and/or computation speed.
There is an ongoing debate in the fields of neuromorphic engineering and artificial intelligence on what is considered to be neuromorphic and what is not. Sometimes, for example, application-specific integrated circuits (ASICs) built for the acceleration of artificial neural networks (ANNs) like the Tensor Processing Unit (Jouppi et al., 2017) are counted among neuromorphic architectures as they are designed to optimally perform computations for a model that is, at its roots, inspired by the brain. More commonly however the distinction is made based on the employed communication mechanisms. Spiking communication is seen as one of the key methods for neuromorphic architectures to inherit the energy efficiency of the brain (Roy et al., 2019). In this thesis we will focus on spiking hardware systems.

FPGA-based neuromorphic implementations: Field-programmable gate arrays (FPGAs) are commercially available general purpose computing architectures that can be used to achieve a more efficient simulation of neural networks (both spiking or non-spiking) compared to simulations performed on a CPU or GPU. An example for this is the FPGA-based neuromorphic cortex simulator introduced in Wang et al. (2018). In the field of neuromorphics they are also commonly used in the development phase of custom digital ASICs as they allow for cheaper and faster (pre-silicon) testing and benchmarking, as done for example in Frenkel et al. (2020).

Custom digital platforms: In contrast to the commercially available and general purpose FPGAs, custom design digital platforms are highly specialized on the simulation[5] of (spiking) neural networks. Nevertheless, the designers have significant freedom to choose the main goal of their system and the design trade-offs they need to take to achieve it. For example, the TrueNorth system by IBM (Akopyan et al., 2015) is focused on efficient simulations of network dynamics and trades flexibility for it (fixed neuron model, no plasticity), while the Loihi chip by Intel (Davies et al., 2018) favors online learning capabilities and supports on-chip learning with configurable plasticity rules. In contrast to these fully custom platforms, the SpiNNaker system relies on conventional small ARM processors, which are used for the computation of the neuron dynamics and which communicate with each other via a custom spike-routing mechanism. The use of ARM processors as neuromorphic

[5]The terms of simulation and emulation are often not clearly defined in the field of neuromorphics. For this thesis, we will speak of a simulation, if the neuron and synapse dynamics are calculated using numerical methods (e.g. using Euler integration to advance the quantities step by step in time), while in an emulation a physical system (e.g. a circuit) is governed by the same differential equations as the neurons and synapses and we observe the behavior of the physical system.

cores in combination with custom spike routing allows for a large flexibility in network architectures and simulated neuron as well as synapse models. For TrueNorth, Loihi as well as SpiNNaker it is possible to combine multiple of the respective chips on a custom board which connects the individual systems to each other and allows for the simulation of larger networks than can be achieved with a single chip (Furber, 2016).

Mixed-signal ASICs: Mixed-signal neuromorphic platforms contain both analog and digital circuitry. The analog circuits replace the numerical simulation of the neuron and synapse dynamics by a physical emulation. The communication of spikes between neurons is realized using similar digital mechanisms as on the fully digital platforms. The analog emulation of dynamics offers a significant advantage in terms of energy efficiency[6] but comes at the cost of reduced flexibility[7] as well as increased levels of variability and noise. We will describe these platforms in detail in Section 2.3.2.

Neuromorphic architectures based on novel devices: While the neuromorphic platforms discussed so far differ from conventional computing hardware in architecture or technology use, both fundamentally rely on the complementary metal-oxide-semiconductor (CMOS) technology. Recent developments in the field of material sciences have shown that this could be complemented by novel devices, especially in the field of neuromorphic computing. The most prominent of these new devices is the memristor ("memory resistor") (Strukov et al., 2008). Roughly speaking, a memristor is a resistor with a history-dependent resistive value. In a neuromorphic context memristors could be used as a low-power alternative for synaptic circuits, where the memristor functions as an energy efficient storage for the synaptic weight. Some memristors even exhibit properties resembling the STDP plasticity mechanism and could therefore directly include a learning mechanism in the synaptic weight storage (Covi et al., 2015; Acciarito et al., 2016). While promising significant gains in energy efficiency, the current generations of memristors still suffer from large device variability and cycle-to-cycle variability (Pino et al., 2012; Gi et al., 2015), which limits their practical applicability. In addition to the memristor, other novel technologies and their application in neuromorphic computing such as phase-change memory, spintronics and optical electronics are currently being investigated. As a detailed description of all of these technologies is beyond the scope of this thesis, we refer to Schuman et al. (2017, Section V.B) for an extensive summary.

2.3.2 Mixed-signal neuromorphic platforms

Arguably, the mixed-signal approach, where neuron dynamics are emulated in analog circuitry is a natural choice, as in biology also neuron dynamics (e.g. membrane voltages)

[6] and, depending on the platform, also in terms of emulation speed

[7] Different neuron models would require different circuits to emulate them. Therefore, mixed-signal platforms typically restrict themselves to emulate only one neuron model.

evolve as analog values in continuous time. The routing of spikes from presynaptic to postsynaptic neurons is handled by digital circuitry, as this allows the most flexibility, e.g. in the realizable connectivity between the neuron circuits.

As originally proposed by Mead and Ismail (1989), the neuron dynamics are emulated by finding electronic circuits in which the voltages and currents correspond to neuronal quantities and follow the same temporal dynamics. We have already seen an example of this for the LIF neuron in Fig. 2.3. However, for the fabrication of a neuromorphic chip we can not use the common electronic components like the resistors or capacitors shown in Fig. 2.3 as they are simply too large. Instead, we need to rely on transistors and very-large-scale integration (VLSI) technology. Generally speaking, there are two different ways of using the analog dynamics of CMOS transistors to emulate neuron dynamics and the choice of method divides the resulting mixed-signal neuromorphic platforms into two groups.

The first method uses the dynamics of the transistor in the subthreshold regime to emulate neuron dynamics. This regime is characterized by low currents and relatively slowly evolving dynamics which result in a real-time emulation of neurons (i.e. with neuronal time constants similar to the ones of biological neurons). Examples for neuromorphic platforms in this category are ROLLS (Qiao et al., 2015), the DYNAPs (Moradi et al., 2017) and Neurogrid (Benjamin et al., 2014). These platforms operate in real-time, which makes them suitable for the control of robotic devices or for interfacing with neuromorphic sensors. While the sub-threshold regime of the transistors allows them to operate on a low power budget, this comes at the price of more noisy signals, larger (compared to the second mixed-signal device category) device-to-device mismatch as well as larger fixed-pattern noise on a device[8].

In the second category transistors operate in the above-threshold regime. This leads to higher currents and faster evolving dynamics. Neuronal dynamics are emulated faster, with a speed-up of $10^3 - 10^4$ compared to biology, hence these platforms are often called "accelerated". The high emulation speeds make the platforms particularly suited for long-running experiments (e.g. on learning or evolutionary mechanisms) or high throughput applications (e.g. machine learning style image classification) but less suited for real-time robotic control. The higher currents compared to the subthreshold regime make these platforms less susceptible to thermal and fixed-pattern noise but also result in a higher power consumption. The higher power consumption however is, to a certain extent, counterbalanced by the accelerated emulation speed which shortens experiment duration and thereby limits the energy consumed. Examples for neuromorphic systems in the category are the chips in the BrainScaleS series, developed by the Electronic Vision(s) group in Heidelberg, of which we will describe the newest iteration BrainScaleS-2 in the following section (Schemmel et al., 2007, 2010, 2022).

[8]The term fixed-pattern noise describes the fact that due to production imperfections no two transistors on a chip are exactly identical. This leads to variations between e.g. neuron circuits even if they are parameterized identically.

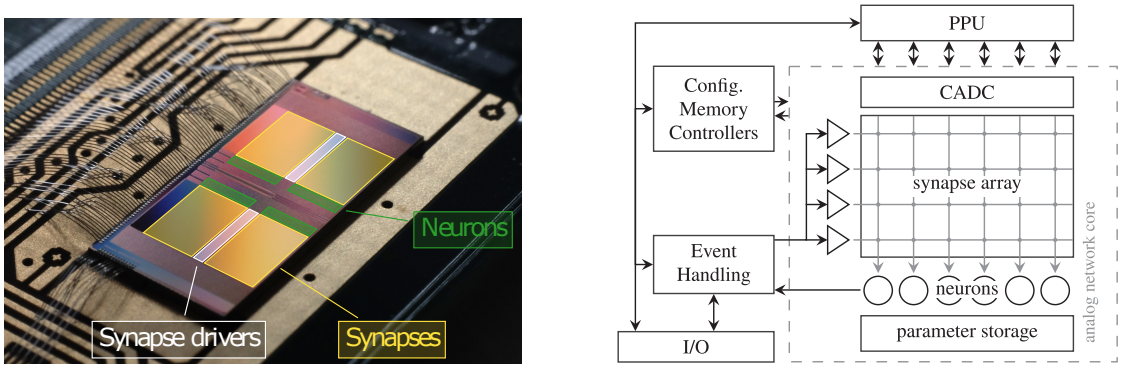

Figure 2.7: BrainScaleS-2 HICANN-X ASIC photograph and block diagram. Left: Photograph of the HICANN-X ASIC. The locations of the neuron circuits (green), the synapse drivers (white) and the synapse array (yellow) are marked on the photograph. Image taken with permission from (Czischek et al., 2022, Figure 5a). **Right:** Block diagram of the main components of the HICANN-X chip. The analog neuromorphic core contains the neuron and synapse circuits. The synapses are arranged in a matrix called the synapse array. Spike signals are sent into the synapse array via the synapse drivers (triangles). The analog parameter storage provides bias currents and voltages for the parametrization of the neuron circuits. The correlation analog-to-digital-converter (CADC) makes synaptic correlation data available as a digital signal. Spike signals produced by the neurons are sent outside of the analog core to the digital circuitry which handles event-routing. The plasticity processing unit (PPU) has access to spike data, traces from the CADC and the synaptic weights and can implement programmable plasticity mechanisms. This diagram was taken with permission from (Billaudelle et al., 2021, Figure 1a).

BrainScaleS-2: HICANN-X

The HICANN-X ASIC is the newest chip in the BrainScaleS-2 series. In contrast to the previous BrainScaleS-1 generation it is fabricated in the newer 65 nm CMOS process and runs with a 1000-fold speed-up compared to biological timescales. Figure 2.7 shows a photograph of the chip as well as a block diagram illustrating the main components of the chip.

The analog network core contains 512 neuron circuits emulating the adaptive exponential leaky integrate-and-fire (AdEx) neuron model (Brette and Gerstner, 2005). The AdEx model is an extension of the LIF neuron model, which adds both an adaptation current and an exponential spike onset. This allows for more biologically plausible voltage dynamics and in particular the reproduction of neuronal firing patterns recorded from biological neurons (Naud et al., 2008). For the applications in this thesis both the adaptation and the exponential circuits were disabled through parametrization such that the standard LIF dynamics were recovered[9]. The synaptic input can be configured to be either CoBa or CuBa

[9]For more details on the analog implementation of the AdEx neurons we recommend Billaudelle (2022); Billaudelle et al. (2022).

with an exponential kernel.[10] The membrane capacitances of the neurons can be coupled, similarly to the illustration in Fig. 2.4, to form multi-compartment neurons. Additionally, a mechanism to model nonlinear dendritic action potentials is implemented (Schemmel et al., 2017).[11] Each neuron can be configured individually using bias currents and voltages that are stored in the analog parameter storage called the "CapMem" (Hock et al., 2013) as well as digital configuration parameters stored in local static random-access memory (SRAM) to enable or disable parts of the circuit.

The synapse circuits are arranged in a grid called the synapse array (see Fig. 2.7). Their main task is to transform the digital event they receive if the presynaptic neuron emits a spike into an analog signal that travels to the postsynaptic neuron. This analog signal is a current pulse. Each synapse has 6 bit of local SRAM storage where the synaptic weight is stored. The larger the synaptic weight, the stronger the current pulse that is sent to the postsynaptic neuron. Additionally, each synapse contains analog circuitry to accumulate correlation measurements between pre- and postsynaptic spike times.

The arrangement of synapse circuits in a grid is to facilitate the routing of spike events from the presynaptic to the postsynaptic neurons. Below each column of synapses in the grid is one neuron circuit (circles in Fig. 2.7). This neuron is the postsynaptic neuron of all synapses in that column. Therefore, once a synapse has emitted a current pulse indicating a presynaptic event, the current pulse only needs to travel down the column to reach the correct postsynaptic neuron. Routing a presynaptic spike event to the correct synapse is more difficult: When a neuron spikes, a digital signal, which carries information about the identity of the spiking neuron, is sent to an event-handler. The event-handler knows which neurons are connected to the neuron that sent the spike and based on this information then forwards the signal to one or multiple synapse drivers (triangles in Fig. 2.7). Each synapse driver is connected to two rows in the synaptic array and can receive spike events from 64 sources. The synapse driver forwards the received signal to the rows in the synapse array it is connected to. Therefore, all synapse circuits in the two rows receive the event. However, each synapse is configured to only react to events coming from one specific presynaptic neuron. As each spike event carries information about the neuron that produced it, each synapse circuit can compare this information to its configuration and only produce a current pulse if the event came from the right source. The presynaptic source of a synapse can be changed while the chip is emulating a network. This allows for a reconfiguration of the network topology, implementing a form of structural plasticity on the chip (Billaudelle et al., 2021).

[10] There were multiple iterations of the HICANN-X chip. Only the newest version (at the time of writing) HICANN-X v3 includes the CoBa synaptic input. For the experiments shown in this thesis the older generation of HICANN-X v2 was used (both the experiments in Chapter 5 and Chapter 6 required CuBa synaptic input). After the publication the experiments shown in Chapter 5 have also been reproduced on the v3 chip version.

[11] This publication describes an older prototype chip version (HICANN DLS3) but the mechanisms are also realized in the current iteration.

The structural as well as synaptic plasticity mechanisms are handled via the plasticity processing unit (PPU) (Friedmann et al., 2013). The PPU is an embedded microprocessor that has access to several observables on the chip including membrane voltages, spike counts and spike-time correlations measured in the synapse circuits. In addition to that the PPU can modify the values of the synaptic weights stored in the SRAM in the synapse circuits. Using this and the available observables, the PPU can perform synaptic weight updates that follow freely programmable plasticity rules.

2.4 Deep learning

In this section we introduce the basic techniques and ideas of deep learning with a particular focus on the relation of ANNs and their biological counterparts. It is important to note that this is not a review of advanced deep learning models and techniques (for this we refer the reader to e.g. Goodfellow et al. (2016)) but rather we cover the fundamental ideas upon which modern deep learning is built.

2.4.1 Artificial neural networks

While modern deep learning uses many different architectures of ANNs (e.g. Krizhevsky et al. (2017); Kingma and Welling (2013); Hochreiter and Schmidhuber (1997)) we will focus here on the most basic one, the fully-connected feedforward network. It consists of multiple layers (groups of artificial neurons) where each layer only receives input from the layer directly adjacent it (Fig. 2.8). All neurons in layer $n-1$ are connected to all neurons in layer n via a weight matrix $\boldsymbol{w}_{n,n-1}$. The first layer is called the input layer, all intermediate ones are hidden layers and the final one is called the output layer. The employed neuron model is highly simplified compared to the more biological ones described in the previous chapters: We call the outputs of the neurons in the n-th layer $\boldsymbol{y}_n$. The ANN equivalent of the membrane voltage we call $\boldsymbol{a}_n$ with

$$\boldsymbol{a}_n = \boldsymbol{w}_{n,n-1}\boldsymbol{y}_{n-1} + \boldsymbol{b}_n \tag{2.20}$$

$$\boldsymbol{y}_n = \varphi\left(\boldsymbol{a}_n\right) \tag{2.21}$$

where $\boldsymbol{b}_n$ is a learnable bias for each neuron and φ is the activation function of the neuron. Common choices for this activation function are a rectified linear unit (ReLU) (Krizhevsky et al., 2012b), a sigmoidal function or $\tanh(.)$. Note that in the machine learning literature this setup is colloquially referred to as multilayer perceptron (MLP) (Goodfellow et al., 2016, Chapter 6), even though this is historically only correct for binary activation functions φ, which nowadays are rarely used (Rosenblatt, 1958).
When the network is presented with an input sample in the lowest layer, the outputs of the neurons are calculated layer by layer until the output layer is reached (Fig. 2.8). The output $\boldsymbol{y}_N$ of the neurons in the output layer N determines the network's response to the given

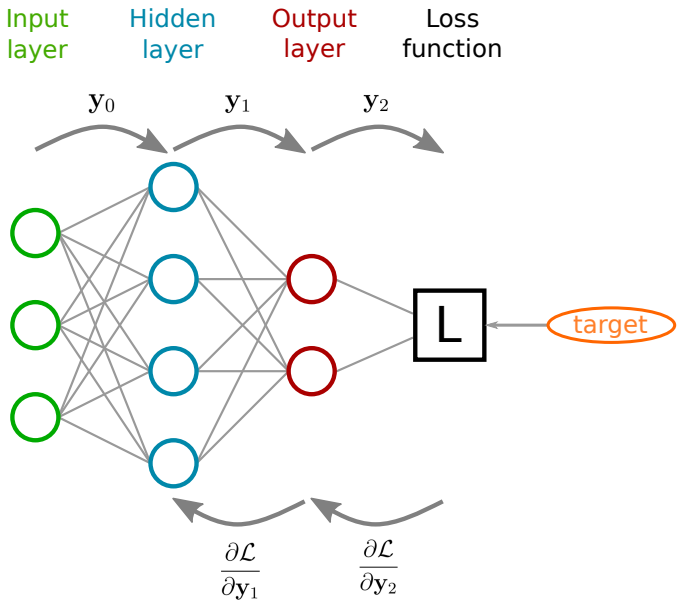

Figure 2.8: Information propagation in ANNs Schematic drawing of a small fully-connected feedforward ANN. The input layer is shown in green, the hidden layer in blue and the output layer in red. When input is presented to the network, activities $\boldsymbol{y}$ travel from left to right to the output layer. The loss function (black box with label L) takes the network's output and evaluates how well the network performs, e.g. by comparing the output to a target. For the learning step "error signals", which indicate how much each activity contributed to the loss $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n}$, are sent backwards through the network.

input sample. This response is fed into a task-dependent loss function $\mathcal{L}$ which determines how good the produced output is for the given input sample.

2.4.2 Error backpropagation

ANNs are trained to solve a task by adjusting their weights and biases using gradient descent on the loss function

$$\Delta \boldsymbol{w}_{n,n-1} = -\eta \, \nabla_{\boldsymbol{w}} \, \mathcal{L} \tag{2.22}$$

$$\Delta \boldsymbol{b}_n = -\eta \, \nabla_{\boldsymbol{b}} \, \mathcal{L} \tag{2.23}$$

with a small learning rate η. Optimally, each update step should be based on the value of the loss $\mathcal{L}$ obtained for all available training examples, however in practice this is typically not feasible. If the number of training samples (the training set) is too large, each update step is calculated on the basis of only a subset of the training samples (a "batch"). This is commonly called stochastic gradient descent (definitions in machine learning literature vary, sometimes stochastic gradient descent only refers to a batch size of 1 sample).

The algorithm employed to realize the gradient descent steps in ANNs is called "error backpropagation" (Rumelhart et al., 1986). In its essence the error backpropagation algorithm is a layer-wise application of the chain-rule for derivatives and results in weight update equations that are based on a layer-wise recursive error signal. Here we present the weight updates and recursive error calculation that are derived in Appendix A.1.
The updates of the synaptic weights from layer $n-1$ to n are derived from gradient descent on some task-dependent loss function $\mathcal{L}$

$$\Delta \boldsymbol{w}_{n,n-1} = -\eta \, \nabla_{\boldsymbol{w}} \, \mathcal{L} \tag{2.24}$$

$$= -\eta \underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi' \left(\boldsymbol{a}_n \right)}_{=\boldsymbol{e}_n} \boldsymbol{y}_{n-1}^{\mathrm{T}} \tag{2.25}$$

$$= -\eta \, \boldsymbol{e}_n \, \boldsymbol{y}_{n-1}^{\mathrm{T}} \tag{2.26}$$

where we define the error signal for the layer n as $\boldsymbol{e}_n = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi' \left(\boldsymbol{a}_n \right)$.
The error signal can be calculated recursively from the error signal in the layer above

$$\boldsymbol{e}_n = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi' \left(\boldsymbol{a}_n \right) \tag{2.27}$$

$$= \left[\boldsymbol{w}_{n+1,n}^{\mathrm{T}} \boldsymbol{e}_{n+1} \right] \odot \varphi' \left(\boldsymbol{a}_n \right) \, . \tag{2.28}$$

Fig. 2.8 illustrates how this can be realized in a network when the derivative of the loss with respect to the layers' activity $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n}$ is propagated backwards through the layers. Note that $\varphi' \left(\boldsymbol{a}_n \right)$ does not need to be sent backwards from layer $n+1$ as it is a local quantity of layer n.

2.4.3 Biological plausibility

Even though ANNs are to a certain extent inspired by the brain, they and especially the error backpropagation mechanism employed to train them have been deemed biologically implausible for a long time (Crick, 1989; Grossberg, 1987). Nevertheless, there has been a large interest both in the field of computational neuroscience and AI to find alternative formulations or approximations of error backpropagation that address the biologically implausible aspects of the original version (Richards et al., 2019; Whittington and Bogacz, 2019; Lillicrap et al., 2020). We detail here the areas of implausibility which need to be addressed by a biologically-plausible variant of deep learning and error backpropagation. These aspects are collected from reviews on this topic by Whittington and Bogacz (2019) and Lillicrap et al. (2020).

- **Symmetry of forward and backward connectivity:** In an ANN information can flow forward and backward through the same synaptic connection. As biological

(chemical) synapses are one-way connections, this information flow would need to be realized by having two biological synapses with opposite directions but the same synaptic strength for each artificial one. This is however problematic since the synaptic strength of the feedforward synapse changes during learning and the feedback synapse, which is physically separate from it, would need to somehow notice this change and adjust itself accordingly. This issue is commonly referred to as the "weight transport problem". A biologically plausible model needs to either not require the symmetry between forward and backward connectivity or find a biologically plausible learning mechanism that keeps the feedback connections symmetric to the forward ones.

- **Phased computation:** During the training of an ANN we alternate between feedforward and feedback phases. In the feedforward phase an input is presented to the network and travels from lowest to highest layer towards the output layer of the network. In the feedback phase an error signal is sent into the network in the output layer and travels backwards. The information flow is strictly unidirectional in both phases and since the phases are separate, the error signals never mix or interfere with the feedforward information flow. Such phased and strictly separated information flow is unrealistic. A biologically plausible model needs to be able to handle simultaneous forward and backward information flow as well as the interference of both.

- **Non-local weight updates:** The synaptic weight updates of an ANN are calculated by an external algorithm (error backpropagation) which has access to all state variables in the network. However, it is assumed that a biological synapse can only rely on quantities represented in the pre- and postsynaptic neurons and maybe a global (i.e. network wide) factor through neuromodulators (Gerstner et al., 2018). Therefore, a biologically plausible model must include mechanisms to represent the required quantities for the weight updates locally in space and time, which means at the time of the synaptic update within the pre- or postsynaptic cell.

- **Signed error signals:** The error signals propagating backwards through the network can be of both positive and negative value. This is in disagreement with the assumption that biological neurons communicate via spikes or firing rates. Neither of them are directly able to communicate a signed value, because spikes are just a 1-bit signal and firing rates are strictly positive. A biologically plausible model should be able to represent, communicate or calculate the signed error signals with only spikes or positive firing rates.

- **Unrealistic neuron model:** The neuron model employed in an ANN is highly simplified, as it has for example no notion of time. A more biologically plausible model should include a neuron model which more closely matches the behavior of biological neurons and exhibits temporal dynamics.

In recent years significant progress has been made in addressing the above-mentioned points. One of the most prominent findings, upon which many subsequent studies rely, is the mechanism of feedback alignment (FA) by Lillicrap et al. (2016). It tackles the weight transport problem by showing that the synaptic weights used for the backward pass through the network do not need to be the transposed of the forward weights but instead can be replaced with a random (and constant throughout the training) set of weights. While losing some performance, the FA networks are still able to learn tasks with a high degree of accuracy. Lillicrap et al. (2016) attributes this to the forward weights roughly aligning themselves (throughout the training) with the randomly chosen backward weights. Due to this alignment the errors propagated using the random feedback weights point roughly in a similar direction as the "correct ones" which the network would have propagated if the transposed of the forward weights would have been used. In practice, it turns out that the errors pointing roughly in the right direction is often good enough to eventually learn to solve a task. Even though later studies have shown that FA is not feasible for all network architectures and sizes (Bartunov et al., 2018; Moskovitz et al., 2018; Max et al., 2022) it is still widely used due to the fact that it completely avoids the weight transport problem while not requiring any architectural changes to the network or any other additional complexity. There are many recent studies attempting to address the biological implausibilities of error backpropagation by suggesting approximative solutions that prevent one or multiple of the above-mentioned implausibilities (Whittington and Bogacz, 2017; Scellier and Bengio, 2017; Guerguiev et al., 2017; Sacramento et al., 2018; Mesnard et al., 2019; Song et al., 2020; Millidge et al., 2020a,b; Pozzi et al., 2020; Payeur et al., 2021; Tang et al., 2021). Among them is the dendritic microcircuit model of Sacramento et al. (2018) which we will summarize, discuss and improve upon in multiple aspects in Chapter 6.

Chapter 3

Hypothesis and Aim

Within the last decade deep learning has revolutionized the field of machine intelligence starting from breakthroughs in computer vision (Krizhevsky et al., 2012a) to nowadays approaching human-level performance in a variety of areas such as complex strategical planning and natural language processing (Vinyals et al., 2019; Brown et al., 2020; Rae et al., 2021). This progress was fueled by the ever-increasing amount of available compute power, mainly carried by the exponential growth of transistor densities on chips, commonly referred to as Moore's law (Moore et al., 1965). However, as transistor dimensions are approaching the sizes of single atoms, this growth is slowing down and the continuation of Moore's law is uncertain. Currently, parallelization techniques in combination with specialized deep learning hardware are employed to fulfill the increasing demand on computational power, but also this approach is expected to struggle to keep up in the future (Thompson et al., 2020; Leiserson et al., 2020). Nonetheless, new deep learning models are tackling increasingly challenging tasks while growing ever larger and requiring more compute power. Already a few years ago it was not out of the ordinary for state-of-the-art models to require the equivalent of several hundred GPUs (consuming dozens of kW of power) to perform a task at human level (Silver et al., 2016; Economist, 2016). This is not sustainable, in particular if we remind ourselves that the original inspiration of these artificial neural networks, the human brain, is able to solve such tasks at a power budget of around 20 W.

The field of neuromorphic engineering aims to develop novel computing architectures that are more closely related to the mechanisms in the brain. They thereby hope to inherit the efficiency of their biological role model. Neuromorphic computing architectures that strive to mimic the brain's ability to learn, need to include mechanisms that allow for the adaption of their neural networks such that they can learn or be trained to perform a large variety of tasks. However, since it is so far not fully understood how learning in the brain works in detail, there is no ready-made blueprint of how the learning mechanisms on neuromorphic architectures should look like. In contrast to the only partially understood learning in biological neural networks, the mechanisms for training artificial neural networks, in par-

ticular the error backpropagation algorithm, are known since the 1980s (Rumelhart et al., 1986). Since then, the error backpropagation algorithm has demonstrated its flexibility and ability to train artificial neural networks in a variety of scenarios and is at the center of the resounding success of deep learning.

We therefore hypothesize that the combination of neuromorphic hardware and error backpropagation can yield powerful and flexible devices for the deployment of neural networks and present an energy-efficient alternative to classical deep learning on conventional computing hardware.

Even though both, classical deep neural networks and current neuromorphic platforms, draw inspiration from the same archetype, they are not directly compatible with each other. In this thesis we aim at bridging this gap. There are two possible angles from which we can approach this task, both of which we will explore in this thesis. The first option is to start from current neuromorphic design philosophies, to identify common components of recent neuromorphic platforms and to develop a method of performing error backpropagation based on these components (Chapter 5). The second option is to start out with an already existing biologically plausible approximation of error backpropagation and to modify it step-by-step to make it amenable to hardware implementation (Chapter 6). Before we can address either of these topics however, we need to solve an often over-looked difficulty (Chapter 4):

Chapter 4: A dataset for algorithmic and neuromorphic prototyping

As it is our aim to implement a variant or approximation of error backpropagation on a neuromorphic platform, we need to ensure, both during the development process of the algorithm and during the deployment on the hardware platform, that the implemented/deployed algorithm actually correctly propagates the errors as intended. Theoretically, this can simply be tested by attempting to learn a task which is only solvable by if appropriate error signals reach all layers of the network. In practice however, this is a difficult problem: Common benchmarks, for which we are confident that they are difficult enough and therefore sensitive to a potential flaw in an implementation, e.g. CIFAR (Krizhevsky et al., 2014) or Imagenet (Deng et al., 2009), are simply too large. They typically require extensive training runs which can hinder quick iterations when developing and testing an algorithm. Additionally, the relatively large size of the networks required to solve them can prohibit the deployment on neuromorphic prototype hardware which often has only limited resources. Conversely, smaller tasks, such as the classification of the MNIST dataset (LeCun et al., 1998) or XOR, are unsuitable because they are not sensitive enough to expose implementation flaws. In Chapter 4 we therefore aim to engineer a task, the Yin-Yang dataset, that is suitable both for algorithmic as well as hardware prototyping. For this we require it to be a difficult enough test case to spot implementation flaws while at the same time being small enough to be deployable on small-scale neuromorphic systems.

Chapter 5: Error backpropagation compatible with neuromorphic LIF neurons

One of the big obstacles for implementing error backpropagation on neuromorphic chips is the fact that the original error backpropagation mechanism is not directly applicable to networks of spiking neurons. This is due to the fact that it requires differentiability of activities and spikes are, in principle, non-differentiable, all-or-nothing signals. Therefore, developing an error backpropagation algorithm compatible with spiking neurons has been an active field of research in recent years (Mostafa, 2017; Neftci et al., 2019). In Chapter 5 we go a step further and not only aim to find an algorithm that is compatible with LIF neurons, one of the most common spiking neuron types on neuromorphic platforms, but is also robust to neuromorphic hardware constraints and distortion effects (Göltz et al., 2021). In particular, we both evaluate our algorithm in an ideal simulated setting and investigate the effects of fixed-pattern noise and quantized weight values. Finally, we demonstrate the algorithm's suitability for neuromorphic deployment by implementing it on the BrainScaleS-2 platform. We show that our algorithm is able to leverage the efficiency and speed of the BrainScaleS-2 platform and achieve competitive results in accuracy, classification speed as well as power consumption.

Chapter 6: Adapting a biologically plausible algorithm for deployment on neuromorphic hardware

In contrast to the original error backpropagation algorithm its biologically plausible variants are more likely to be amenable to an implementation on neuromorphic platforms. This is due to the fact that the biologically plausible variants operate, to varying extend, under similar constraints as those imposed by neuromorphic platforms, such as for example the locality of learning rules. The dendritic microcircuits of Sacramento et al. (2018) are one of the recently developed models that approximate error backpropagation in a local, phase-free and bio-plausible fashion. However, despite their biological plausibility, the dendritic microcircuits are still not directly deployable on current neuromorphic systems. This is due to for example their rate-based exchange of information, which contrasts the spike- or event-based communication on neuromorphic hardware. In Chapter 6 we assess the model with the focus on practical feasibility and neuromorphic implementability. We, furthermore, propose multiple extensions that make a neuromorphic deployment more feasible and can also inform the development of future hardware generations.

Chapter 4

Result I: A proper test case for prototyping

This chapter contains the article *The Yin-Yang dataset*. It was published in the *Association for Computing Machinery's (ACM) Proceedings of the Neuro-inspired Computational Elements Workshop 2022*[1] and a preprint is available on *arXiv*.[2] The format was adapted to the format of this thesis and the references have been included in the main bibliography.

Author contributions

The project idea was developed jointly by LK, JG and MAP. LK and JG designed the experiments, LK wrote the necessary software, prepared it for publication on GitHub[3] and performed the experiments in Fig. 4.1, Fig. 4.2 and Fig. 4.3. The Fig. 4.4 was created by JG. LK, JG and MAP collectively wrote the article.

[1] Kriener et al. (2022)
[2] Kriener et al. (2021a)
[3] Kriener et al. (2021b)

The Yin-Yang dataset

L. Kriener[1], J. Göltz[2,1], M. A. Petrovici[1,2]

[1] Department of Physiology, University of Bern, 3012 Bern, Switzerland.
[2] Kirchhoff-Institute for Physics, Heidelberg University, 69120 Heidelberg, Germany.

Abstract

The Yin-Yang dataset was developed for research on biologically plausible error backpropagation and deep learning in spiking neural networks. It serves as an alternative to classic deep learning datasets, especially in early-stage prototyping scenarios for both network models and hardware platforms, for which it provides several advantages. First, it is smaller and therefore faster to learn, thereby being better suited for small-scale exploratory studies in both software simulations and hardware prototypes. Second, it exhibits a very clear gap between the accuracies achievable using shallow as compared to deep neural networks. Third, it is easily transferable between spatial and temporal input domains, making it interesting for different types of classification scenarios.

4.1 Introduction

We introduce the Yin-Yang dataset for learning in hierarchical networks (Kriener et al., 2021b). It is tailored to the requirements of research on biologically plausible error backpropagation algorithms, learning in spiking neural networks and hierarchical networks on neuromorphic hardware. These fields typically require small but at the same time not trivially solvable datasets to prototype and test network architectures and learning algorithms. Setups commonly used for this purpose involve either elementary logic tasks such as XOR or small-scale datasets such as MNIST or fashion-MNIST (LeCun et al., 1998; Xiao et al., 2017) and reduced versions thereof. However, these setups often do not adequately fulfill their purpose. Binary XOR only has a tiny number of input patterns and therefore a very limited, discrete set of reachable accuracies, making the evaluation and comparison of learning algorithms difficult. In turn, MNIST-type datasets have other drawbacks. For one, they require comparatively large networks, which might not be feasible during prototyping. But even more importantly, and despite this ostensible difficulty, they can nevertheless be classified with high accuracy even by shallow networks or networks without learning in the lower layers. This is problematic because training a deep network with an imperfect learning algorithm can result in performance indistinguishable from that of a shallow network or a network with plasticity only in the last layer. Conversely, a test on the MNIST dataset can fail to reveal the inability of the training algorithm to propagate error signals through the network, as the achieved high accuracies obscure the underlying problem.

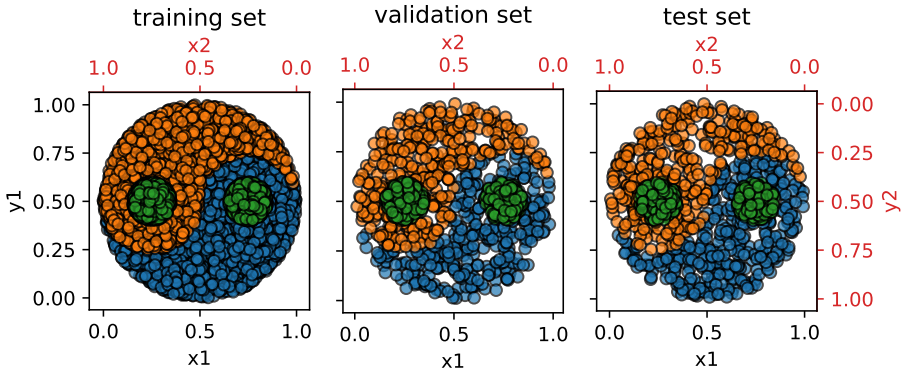

Figure 4.1: Training, validation and test dataset. Each dot in the yin-yang symbol represents one sample of the dataset. The color of the dot denotes its class ("Yin", "Yang" or "Dot"). This figure was generated using the default settings for random seeds and dataset sizes (5000 samples for the training set and 1000 samples each for the validation and test set).

The Yin-Yang dataset can provide an alternative for these testing and prototyping scenarios as it is solvable by smaller networks, contains fewer samples and most importantly exhibits a large gap between the accuracies reached by shallow or partly fixed networks on the one hand and correctly trained deep networks on the other. Note that here, we use "deep" in opposition to "shallow", i.e., any network that has latent variables through which errors need to propagate. We consider a shallow network to be the equivalent of a single-layer perceptron, with only an input layer connected directly to a label layer.

4.2 Dataset

Each sample in the dataset represents a point in a two-dimensional representation of the yin-yang symbol. Depending on their location in the symbol the samples are classified into the "Yin", "Yang" or "Dot" class (Fig. 4.1). Even though the areas in the yin-yang symbol covered by the different classes have different sizes, the dataset is designed to be balanced, which means that all classes are represented by approximately the same amount of samples. Note that therefore the density of samples is higher in the "Dot"-class regions, as the combined area of these regions is smaller than that of the others.

The samples are randomly generated using rejection sampling. The exact version of a generated set of samples is therefore determined by the random seed and dataset size. This makes it possible to produce multiple dataset versions by providing different random seeds and dataset sizes. In the default configuration the training set has 5000 samples while the validation and test sets have 1000 samples respectively, each generated with a different random seed.

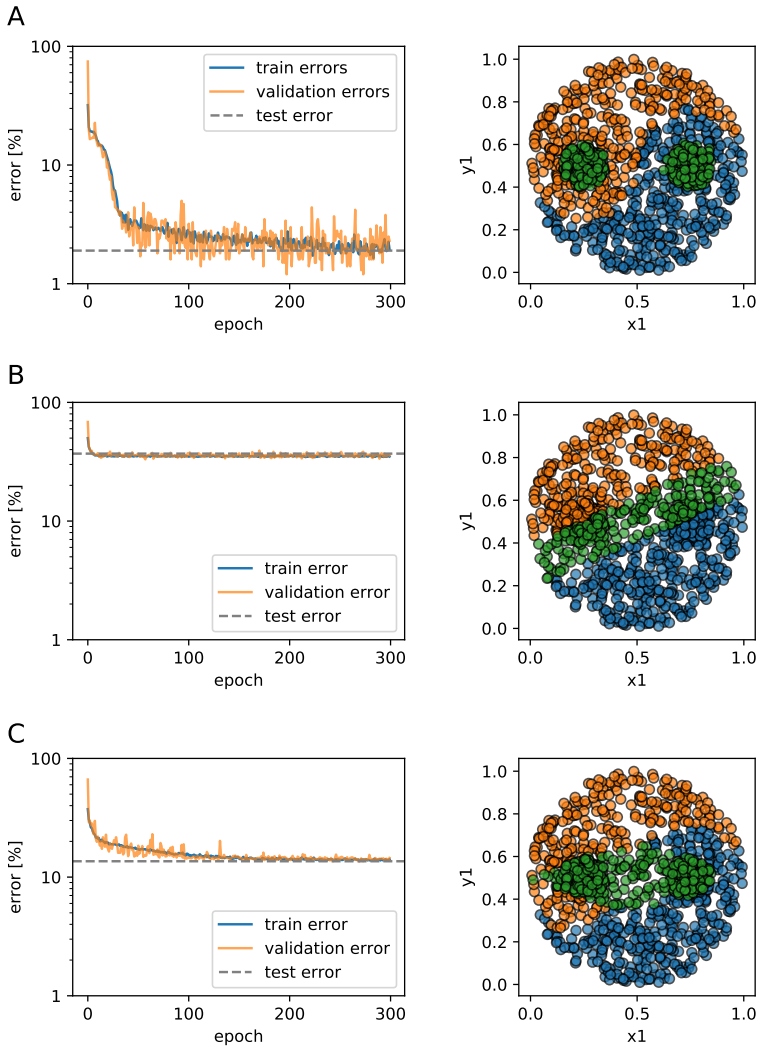

Figure 4.2: Comparison of exemplary training results for different network setups. Network parameters are given in Table 4.2. **Left column:** Evolution of the validation and training error during training. **Right column:** training result illustrated on the test set. **(A)** Network with one hidden layer and fully functional synaptic plasticity via classical error backpropagation. **(B)** Shallow network. **(C)** Network with one hidden layer and frozen lower weights.

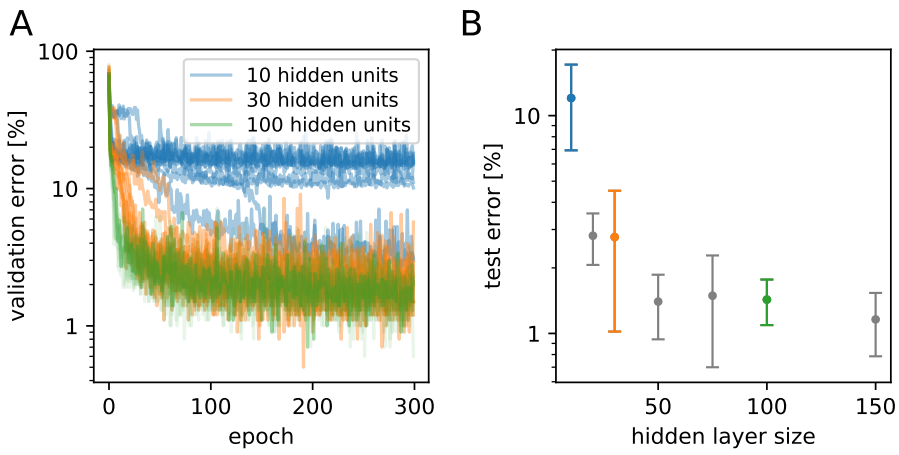

Figure 4.3: Impact of hidden layer size on network performance. (A) Validation errors during training for three different network architectures with different hidden layer sizes. For each architecture, ten training runs with different random weight initialization are overlaid. **(B)** Mean and standard deviation of the final test error depending on hidden layer size of the network. The colored data points correspond to the runs shown in A.

Table 4.1: Mean and standard deviation of the test accuracy for 20 training runs with different random initializations for different network configurations. Training parameters can be found in Table 4.2.

network	**hidden layer with 20 neurons**	**hidden layer with 30 neurons**
deep network	(97.0 ± 1.6) %	(97.6 ± 1.5) %
deep network (frozen lower weights)	(78.3 ± 7.8) %	(85.5 ± 5.8) %
shallow network	(63.8 ± 1.0) %	

As can be seen in Fig. 4.1, values of all samples in the dataset are strictly positive. This is the case to accommodate network models which require positive input values only (common in the field of biologically-plausible networks, as firing rates as well as spike times are typically denoted by positive numbers). Because of that the yin-yang symbol is not centered around zero. This however complicates training in neuron models without intrinsic (learnable) bias. To facilitate training for these models, each sample in the dataset consists not only of the coordinates (x, y) determining the position in the yin-yang symbol but additionally also the values $(1-x, 1-y)$. This effectively symmetrizes the input and removes the need for a bias even though the yin-yang symbol is not centered around the origin of the coordinate system.

4.3 Training results

As a baseline for further applications of this dataset we also provide some training results achieved with classical artificial neural networks. In particular, we compare network performance in three scenarios:

1. a network with one hidden layer and fully functional error backpropagation;
2. a shallow network with only an input and an output layer;
3. a network with one hidden layer, but with frozen weights between the input and the hidden layer to emulate training with a faulty error backpropagation algorithm.

For all scenarios, we use very small network sizes to emulate a model or hardware prototyping environment. Incidentally, this is also helpful in highlighting another problem that is frequently overlooked when increasing the network size: because a large enough hidden layer can mask faulty error backpropagation, larger-scale networks are often inadequate for a quantitative verification of credit assignment (precise error propagation) within the studied network model. This is discussed below in more detail.
The comparison between the three scenarios (Table 4.1 and Fig. 4.2) illustrates a manifest advantage of the Yin-Yang dataset compared to other commonly used datasets of comparable size: both the shallow network and the one with the frozen lower weights are clearly unable to learn the required features to successfully classify the dataset. This leads to a gap of more than 30 % between the accuracies achieved by a shallow and a deep network.
The failure of the partially frozen network highlights another important issue for various proposals of bio-plausible solutions to the credit assignment problem. In large enough networks, the large hidden layers project the input into a very high-dimensional space, which makes classification tasks more easily solvable by the linear classifier embodied by the top layer. This is commonly referred to as the "kernel trick" (see e.g. Scholkopf (2001)). This can easily mask the inability of a network to correctly propagate errors and perform true gradient descent learning. While this issue would become observable when dealing with more complicated classification problems, it would require using large, deep networks that are not only difficult to debug but, more importantly, would lie beyond the capabilities of typical prototype devices or software simulations.
The Yin-Yang dataset addresses both problems simultaneously, by clearly highlighting faulty error backpropagation already within resource-efficient implementations with hidden layer sizes of around 20 to 30 neurons (see Table 4.1). Under these circumstances, the difference between the accuracy reached by a properly trained network and the network where only the top weights are trained lies around 20 % and 12 % respectively. This is a much higher gap than in a comparable example with the MNIST dataset, where networks need several hundred hidden neurons to show significant performance improvements beyond linear classifiers LeCun et al. (1998). However, such sizes automatically introduce the

Table 4.2: Training parameters used to produce the results in Fig. 4.2. Fig. 4.3 uses the same parameters except for the size of the hidden layer.

parameter name	**value**
activation function	ReLU
size input	4
size hidden layer (for deep net)	30
size output layer	3
training epochs	300
batch size	20
optimizer	Adam, (Kingma and Ba, 2014)
Adam parameter β	$(0.9, 0.999)$
Adam parameter ϵ	10^{-8}
learning rate	0.01

kernel trick: a network with 500 hidden units reaches on average 98.3 % on MNIST, while the same network with only training in the top layer reaches 94.8 %. Unmasking these issues can become crucial in research on biologically plausible forms of credit assignment and (local) synaptic plasticity, where exact error backpropagation is notoriously difficult to realize, both for rate-based models and, even more pronouncedly, for spiking networks.

Another advantage of the Yin-Yang dataset over many other commonly used datasets is the dimensionality of its samples and the network sizes required to learn the task. Each sample consists of only four input values (compared to, e.g., the 784 input channels required by MNIST), which significantly reduces the required fan-in for hidden neurons. This can be especially beneficial on neuromorphic platforms, where the number of synaptic connections to a neuron is very often limited by the chip architecture, even more so for early-stage prototypes (e.g. (Binas et al., 2016, Section 3.3), Moradi and Indiveri (2013); Schemmel et al. (2017); Frenkel et al. (2018); Nair and Indiveri (2019); Billaudelle et al. (2020)).

Also, this dataset can be learned with a single hidden layer of reasonably small size (Fig. 4.3). For consistently high final accuracies, a hidden layer of 20 to 30 neurons is required, but for a small proof-of-concept demonstration of a learning algorithm or hardware prototype, even 10 hidden units are enough to achieve results (around 88 % accuracy) that would be impossible with shallow networks, or with algorithms that cannot profit from a network's representational hierarchy. The full set of training parameters can be found in Table 4.2.

In addition to the results shown here, the dataset has already been used to showcase algorithms for error backpropagation in spiking neural networks in (Göltz et al., 2021) and (Wunderlich and Pehle, 2021).

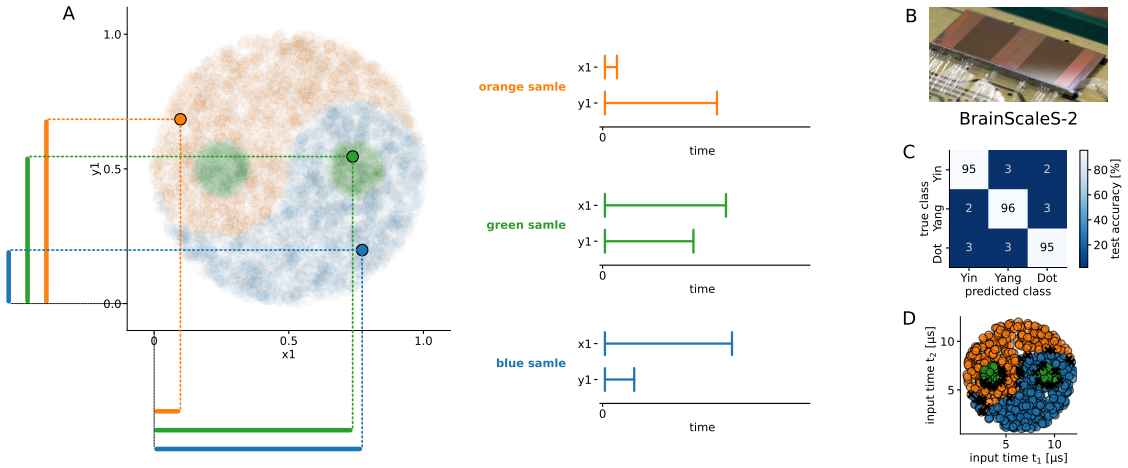

Figure 4.4: Spatio-temporal input encoding scheme and classification results on the neuromorphic chip BrainScaleS-2. Panels **(B-D)** taken from Göltz et al. (2019). **(A)** Encoding of the x, y-coordinates of the Yin-Yang pattern as input spike times t_1 and t_2 illustrated on one sample each for the three classes. **(B)** Image of the BrainScaleS-2 ASIC. **(C)** Confusion matrix after training the BrainScaleS-2 chip to classify the Yin-Yang dataset. **(D)** Classification result of the chip on the test set. For each input sample the color indicates the class determined by the trained network. Wrong classifications are marked with a black X. The wrongly classified samples all lie very close to the border between two classes.

4.4 Input encoding

The Yin-Yang dataset can be adapted to suit the needs of very different network models. Depending on the used network architecture, neuron model and mode of communication between the neurons, different types of information encoding become necessary. In the following, we discuss several encoding methods that are well-suited for a variety of different network and neuron types.

4.4.1 Spatio-temporal input encoding

Using this dataset for spiking neural networks requires an explicit spatio-temporal input encoding. In Göltz et al. (2021) and Wunderlich and Pehle (2021), the four input features of the dataset were directly interpreted as the spike times of 4 input neurons (Fig. 4.4 A). This was done by choosing parameters t_{early} and t_{late} as the earliest and latest possible time the input neurons are allowed to spike. Then the dataset values $\mathbf{x} = (x, y, 1-x, 1-y)$ were translated into the four spike times $\mathbf{t} = (t_1, t_2, t_3, t_4)$ as follows:

$$\mathbf{t} = t_{\text{early}} + \mathbf{x} \cdot (t_{\text{late}} - t_{\text{early}}) \tag{4.1}$$

The choice of $t_{\text{early/late}}$ is dependent on the network architecture and employed learning algorithms. For Göltz et al. (2021) it has proven beneficial to choose t_{early} slightly after the start of the experiment and t_{late} as the sum of the two neuron time constants $t_{\text{late}} \approx \tau_{\text{m}} + \tau_{\text{syn}}$. The classification results achieved with the BrainScaleS-2 chip are shown in Fig. 4.4.
Alternatively, a different spike-based spatio-temporal encoding can be achieved implicitly by manipulating input currents, as proposed for example in Cramer et al. (2020b). Here, each input variable is interpreted as the strength of a constant input current into a leaky-integrate and fire neuron. The timing of the output spike of the input neurons depends on the strength of the input current I with

$$t_{\text{spike}} = \tau_m \log \frac{I}{I - \theta_{\text{I}}} \tag{4.2}$$

where τ_m denotes the membrane time constant and θ_{I} the minimal current necessary to evoke an output spike.

4.4.2 Rate-based input encoding

Many models for biologically plausible error backpropagation are built around rate-based neuron models (e.g. Sacramento et al. (2018); Scellier and Bengio (2017); Haider et al. (2021), for a review see also Whittington and Bogacz (2019)). These approaches use continuous rates as an idealized version of rate coding in spiking neurons. Others build on the same approximations but explicitly use spike-based communication in their neural network implementations (e.g. Schmitt et al. (2017); Esser et al. (2015); Guerguiev et al. (2017)). For such rate-based models, a suitable encoding scheme can be easily realized by designating 4 input neurons and setting their output rates proportional to the values of the respective input feature.
In case of spiking neurons, these four input neurons can produce Poisson spike trains with the same rates as their rate-based counterparts, as, for example, in Schmitt et al. (2017). Alternatively, regular spike trains could also be used to represent firing rates; while more precise than the intrinsicly stochastic Poisson solution, this scheme has its own potential drawback of making the neuronal input-output function dependent on not just the rate, but also the phase of a neuron's afferents. Under certain circumstances, encoding an input as a single neuron may not be viable, for example when synaptic bandwidth or neuron firing rate are limited. In this case, one input can be represented by a population of neurons with a mean firing rate equal to the value of the input.

Code and data availability

Code for the Yin-Yang data set is available at `https://github.com/lkriener/yin_yang_data_set`. The example notebook in the repository includes the plotting of the data

samples (Fig. 4.1) and the training of deep and shallow networks (Fig. 4.2). Additional data available on request from the authors.

Acknowledgment

We wish to thank Sebastian Billaudelle and Benjamin Cramer for valuable discussions, as well as Mike Davies and Intel for their ongoing support. We gratefully acknowledge funding from the European Union under grant agreements 604102, 720270, 785907, 945539 (HBP) and the Manfred Stärk Foundation.
During the development of the dataset some calculations were performed on UBELIX, the HPC cluster at the University of Bern, others were performed on the bwForCluster NEMO, supported by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963–1 FUGG. Additionally, our work has greatly benefitted from access to the Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858.

Chapter 5

Result II: Exact error backpropagation with LIF neurons

This chapter contains the article *Fast and energy-efficient neuromorphic deep learning with first-spike times*. It was published in the journal *Nature machine intelligence*[1] and a preprint is available on *arXiv*.[2] Springer Nature permits the reproduction of the article as part of the author's thesis. The format was adapted to the format of this thesis and the references have been included in the main bibliography.

Author contributions

JG and LK contributed equally and share the first authorship of this publication. JG, AB and MAP designed the conceptual and experimental approach. JG derived the theory and implemented the algorithm. LK embedded the algorithm into a comprehensive training framework. All experiments were performed in close collaboration between JG and LK. In the following the main contributor to each experiment is indicated. JG performed the hardware emulations and theoretical evaluations (Fig. 5.1, Fig. 5.4, Fig. SI.A1, Fig. SI.D1, Fig. SI.E1, Fig. SI.F1). LK performed the software simulations (Fig. 5.2, Fig. 5.3, Fig. 5.5, Table SI.B1, Fig. SI.C1, Fig. SI.E2). AB and OJB offered substantial software support. SB, BC, JG and AFK provided low-level software for interfacing with the hardware. JG, LK, DD, SB and MAP wrote the manuscript.

[1] Göltz et al. (2021)
[2] Göltz et al. (2019)

Fast and energy-efficient neuromorphic deep learning with first-spike times

J. Göltz*,[1,2], L. Kriener*,[2],
A. Baumbach[1], S. Billaudelle[1], O. Breitwieser[1], B. Cramer[1], D. Dold[1,3], A. F. Kungl[1],
W. Senn[2], J. Schemmel[1], K. Meier[1], M. A. Petrovici[2,1]

* These authors contributed equally

[1] Kirchhoff-Institute for Physics, Heidelberg University, 69120 Heidelberg, Germany.
[2] Department of Physiology, University of Bern, 3012 Bern, Switzerland.
[3] Siemens AI lab, Siemens AG Technology, 80331 Munich, Germany.

Abstract

For a biological agent operating under environmental pressure, energy consumption and reaction times are of critical importance. Similarly, engineered systems are optimized for short time-to-solution and low energy-to-solution characteristics. At the level of neuronal implementation, this implies achieving the desired results with as few and as early spikes as possible. With time-to-first-spike coding both of these goals are inherently emerging features of learning. Here, we describe a rigorous derivation of a learning rule for such first-spike times in networks of leaky integrate-and-fire neurons, relying solely on input and output spike times, and show how this mechanism can implement error backpropagation in hierarchical spiking networks. Furthermore, we emulate our framework on the BrainScaleS-2 neuromorphic system and demonstrate its capability of harnessing the system's speed and energy characteristics. Finally, we examine how our approach generalizes to other neuromorphic platforms by studying how its performance is affected by typical distortive effects induced by neuromorphic substrates.

5.1 Introduction

In recent years, the machine learning landscape has been dominated by deep learning methods. Among the benchmark problems they managed to crack, some were thought to still remain elusive for a long time (Krizhevsky et al., 2012a; Silver et al., 2017; Brown et al., 2020). It is thus not exaggerated to say that deep learning dominates our understanding of "artificial intelligence" (Brooks et al., 2012; Ng, 2016; Hassabis et al., 2017; Sejnowski, 2018; Richards et al., 2019).

Compared to abstract neural networks used in deep learning, their more biological archetypes — spiking neural networks — still lag behind in performance and scalability (Pfeiffer and Pfeil, 2018). Reasons for this difference in success are numerous; for instance, unlike abstract neurons, even an individual biological neuron represents a complex system, with finite response times, membrane dynamics and spike-based communication (Gerstner,

2001; Izhikevich, 2004), making it more challenging to find reliable coding and computation paradigms (Gerstner, 1998; Maass, 2016; Davies, 2019). Furthermore, one of the major driving forces behind the success of deep learning, the backpropagation of errors algorithm (Linnainmaa, 1970; Werbos, 1982; Rumelhart et al., 1986), remained incompatible with spiking neural networks until only very recently (Tavanaei et al., 2018a; Neftci et al., 2019).

Despite these challenges, spiking neural networks promise to hold some important advantages. The time information inherent to spikes allows a coding scheme for spike-based communication that utilizes both spatial and temporal dimensions (Gütig and Sompolinsky, 2006), unlike spike-count-based approaches (Cao et al., 2015; Diehl et al., 2016; Schmitt et al., 2017; Wu et al., 2019), where the information of spike times is at least partially diluted due to temporal or population averaging. Owing to the inherent parallelism of all biological, as well as many biologically-inspired, spiking neuromorphic systems Thakur et al. (2018), this promises fast, sparse and energy-efficient information processing, and provides a blueprint for computing architectures that could one day rival the efficiency of the brain itself (Mead, 1990; Roy et al., 2019; Pfeiffer and Pfeil, 2018; Thakur et al., 2018). This makes spiking neural networks implemented on specialized neuromorphic devices potentially more powerful — at least in principle — than the "conventional", simple machine learning models currently used on von-Neumann machines, even though this potential still remains mostly unexploited (Pfeiffer and Pfeil, 2018).

Many attempts have been made to reconcile spiking neural networks with their abstract counterparts in terms of functionality, e.g., featuring spike-based inference models (Petrovici et al., 2013; Neftci et al., 2014; Petrovici et al., 2016; Neftci et al., 2016; Leng et al., 2018; Kungl et al., 2019; Dold et al., 2019; Jordan et al., 2019; Hunsberger and Eliasmith, 2016) and deep models trained on target spike times by shallow learning rules (Kheradpisheh et al., 2018; Illing et al., 2019) or using spike-compatible versions of the error backpropagation algorithm (Bohte et al., 2000; Zenke and Ganguli, 2018; Huh and Sejnowski, 2018). Especially for tasks operating on static information, a particularly elegant way of utilizing the temporal aspect of exact spike times is the time-to-first-spike (TTFS) coding scheme (Thorpe et al., 2001). Here, a neuron encodes its real-valued response to a stimulus as the time elapsed before its first spike in reaction to that stimulus. Such single-spike coding enables fast information processing by explicitly encouraging the emission of as few spikes as early as possible, which meets physiological constraints and reaction times observed in humans and animals (Thorpe et al., 1996, 2001; Johansson and Birznieks, 2004; Gollisch and Meister, 2008). Apart from biological plausibility, such a fast and sparse coding scheme is a natural fit for neuromorphic systems that offer energy-efficient and fast emulation of spiking neural networks (Schemmel et al., 2010; Akopyan et al., 2015; Billaudelle et al., 2019; Davies et al., 2018; Mayr et al., 2019; Pei et al., 2019; Moradi et al., 2017).

For hierarchical TTFS networks, a gradient-descent-based learning rule was proposed in (Mostafa, 2017; Kheradpisheh and Masquelier, 2020), using error backpropagation on a continuous function of output spike times. However, this approach is limited to a neuron model

without leak, which is neither biologically plausible, nor compatible with most analog very-large-scale integration (VLSI) neuron dynamics (Thakur et al., 2018). We propose a solution for leaky integrate-and-fire (LIF) neurons with current-based (CuBa) synapses — a widely-used dynamical model of spiking neurons with realistic integration behavior (Rauch et al., 2003; Gerstner and Naud, 2009; Teeter et al., 2018). An early version of this work was presented in Göltz (2019).

For several specific configurations of time constants, we provide analytical expressions for first-spike timing, which, in turn, allow the calculation of exact gradients of any differentiable cost function that depends on these spike times. In hierarchical networks of LIF neurons using the TTFS coding scheme, this enables exact error backpropagation, allowing us to train such networks as universal classifiers on both continuous and discrete data spaces.

As our algorithm only requires knowledge about afferent and efferent spike times of all neurons, it lends itself to emulation on neuromorphic hardware. The accelerated, yet power-efficient BrainScaleS-2 platform (Friedmann et al., 2017; Billaudelle et al., 2019) pairs especially well with the sparseness and low latency already inherent to TTFS coding. We show how an implementation of our algorithm on BrainScaleS-2 can obtain similar classification accuracies to software simulations, while displaying highly competitive time and power characteristics, with a combination of 48 µs and 8.4 µJ per classification.

By incorporating information generated on the hardware for updates during training, the algorithm automatically adapts to potential imperfections of neuromorphic circuits, as implicitly demonstrated by our neuromorphic implementation. In further software simulations, we show that our model deals well with various levels of substrate-induced distortions such as fixed-pattern noise and limited parameter precision and control, thus providing a rigorous algorithmic backbone for a wide range of neuromorphic substrates and applications. Such robustness with respect to imperfections of the underlying neuronal substrate represents an indispensable property for any network model aiming for biological plausibility and for every application geared towards physical computing systems (Prodromakis and Toumazou, 2010; Esser et al., 2015; van De Burgt et al., 2018; Wunderlich et al., 2019; Kungl et al., 2019; Dold et al., 2019; Feldmann et al., 2019).

In the following, we first introduce the CuBa LIF model and the TTFS coding scheme, before we demonstrate how both inference and training via error backpropagation can be performed analytically with such dynamics. Finally, the presented model is evaluated both in software simulations and neuromorphic emulations, before studying effects of several types of substrate-induced distortions.

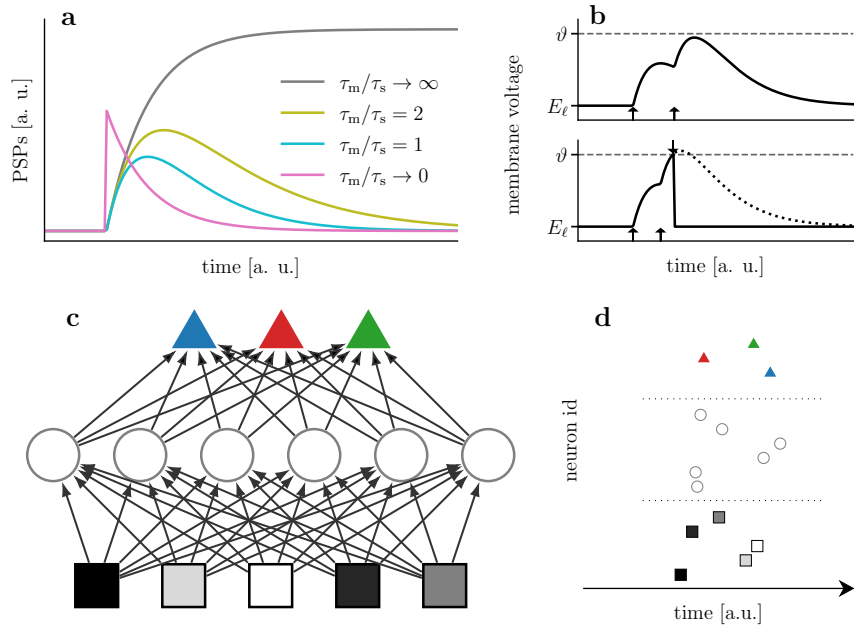

Figure 5.1: Time-to-first-spike coding and learning. Top: single neurons. (a) Postsynaptic potential (PSP) shapes for different ratios of time constants τ_s and τ_m. The finiteness of time constants causes the neuron to gradually forget prior input. **(b)** One key challenge of this finite memory arises when small variations of the synaptic weights result in disappearing/appearing output spikes, which elicits a discontinuity in the function describing output spike timing. **Bottom: application to feedforward hierarchical networks. (c)** Network structure. The geometric shape of the neurons represents a notation of their respective types (input □, hidden ○, label △). The shading of the input neurons is a representation of the corresponding data, such as pixel brightness (■, . . . , ■, . . . , □). The color of the label neurons represents their respective class (▲, ▲, ▲). **(d)** Time-to-first-spike (TTFS) coding exemplified in a raster plot. As an example of input encoding, the brightness of an input pixel is encoded in the lateness of a spike. Note that in our framework, TTFS coding simultaneously refers to two individual aspects, namely the input-to-spike-time conversion and the determination of the inferred class by the identity of the first label neuron to fire (▲). In all figures we denote units in square brackets; in particular, we use [a. u.] for arbitrary units, and [1] for dimensionless quantities, and $[\tau_s]$ for times that are measured in multiples of the synaptic time constant τ_s.

5.2 Results

Leaky integrate-and-fire dynamics The dynamics of an LIF neuron with CuBa synapses are given by

$$C_{\mathrm{m}}\dot{u}(t) = g_\ell[E_\ell - u(t)] + \sum_i w_i \sum_{t_i} \theta(t - t_i) \exp\left(-\frac{t - t_i}{\tau_{\mathrm{s}}}\right) , \tag{5.1}$$

with membrane capacitance C_{m}, leak conductance g_ℓ (from which the membrane time constant $\tau_{\mathrm{m}} = C_{\mathrm{m}}/g_\ell$ follows), presynaptic weights w_i and spike times t_i, synaptic time constant τ_{s} and θ the Heaviside step function. The first sum runs over all presynaptic neurons while the second sum runs over all spikes for each presynaptic neuron. The neuron elicits a spike at time T when the presynaptic input pushes the membrane potential above a threshold ϑ. After spiking, a neuron becomes refractory for a time period τ_{ref}, which is modeled by clamping its membrane potential to a reset value ϱ: $u(t') = \varrho$ for $T \leq t' \leq T + \tau_{\mathrm{ref}}$. For convenience and without loss of generality, we set the leak potential $E_\ell = 0$. Eqn. (5.1) can be solved analytically and yields subthreshold dynamics as described by Eqn. (5.9). The choice of τ_{m} and τ_{s} ultimately influences the shape of a postsynaptic potential (PSP), starting from a simple exponential ($\tau_{\mathrm{m}} \ll \tau_{\mathrm{s}}$), to a difference of exponentials (with an alpha function for the special case of $\tau_{\mathrm{m}} = \tau_{\mathrm{s}}$) to a graded step function ($\tau_{\mathrm{m}} \gg \tau_{\mathrm{s}}$) (Fig. 5.1a). Note that all of these scenarios are conserved under exchange of τ_{s} and τ_{m}, as is apparent from the symmetry of the analytical solution (Eqn. 5.9).

The first two cases with finite membrane time constant τ_{m} are markedly different from the last one, which is also known as either the non-leaky integrate-and-fire (nLIF) or simply integrate-and-fire (IF) model and was used in previous work (Mostafa, 2017). In the nLIF model, input to the membrane is never forgotten until a neuron spikes, as opposed to the LIF model, where the PSP reaches a peak after finite time and subsequently decays back to its baseline. In other words, presynaptic spikes in the LIF model have a purely local effect in time, unlike in the nLIF model, where only the onset of a PSP is localized in time, but the postsynaptic effect remains forever, or until the postsynaptic neuron spikes. A pair of finite time constants thus assigns much more importance to the time differences between input spikes and introduces discontinuities in the neuronal output that make an analytical treatment more difficult (Fig. 5.1b).

First-spike times Our spike-timing-based neural code follows an idea first proposed in (Mostafa, 2017). Unlike coding in artificial neural networks (ANNs) and different from spike-count-based codes in spiking neural networks (SNNs), this scheme explicitly uses the timing of individual spikes for encoding information. In time-to-first-spike (TTFS) coding, the presence of a feature in a stimulus is reflected by the timing of a neuron's first spike after the onset of the stimulus, with earlier spikes representing a more strongly mani-

fested feature. This has the effect that important information inherently propagates quickly through the network, with potentially only few spikes needed for the network to process an input. Consequently, this scheme enables efficient processing of inputs, both in terms of time-to-solution and energy-to-solution (assuming the latter depends, in general on the total number of spikes and the time required for the network to solve, e.g., an input classification problem).

In order to formulate the optimization of a first-spike time T as a gradient-descent problem, we derive an analytical expression for T. This is equivalent to finding the time of the first threshold crossing by solving $u(T) = \vartheta$ for T. Even though there is no general closed-form solution for this problem, analytical solutions exist for specific cases. For example, we show that (see Methods)

$$T = \tau_\mathrm{s} \left\{ \frac{b}{a_1} - \mathcal{W}\left[-\frac{g_\ell \vartheta}{a_1} \exp\left(\frac{b}{a_1}\right) \right] \right\} \qquad \text{for } \tau_\mathrm{m} = \tau_\mathrm{s} \tag{5.2}$$

and

$$T = 2\tau_\mathrm{s} \ln\left[\frac{2a_1}{a_2 + \sqrt{a_2^2 - 4a_1 g_\ell \vartheta}} \right] \qquad \text{for } \tau_\mathrm{m} = 2\tau_\mathrm{s} \ , \tag{5.3}$$

where $\mathcal{W}$ is the Lambert W function and using the shorthand notations a_n and b for sums over the set of causal presynaptic spikes $C = \{i \mid t_i < T\}$ (see Eqns. (5.11) and (5.12)). We note that, when calculating the output spike time for a large number of input neurons, determining C can be computationally intensive (see Methods). One inherent advantage of physical emulation is the reduction of this calculational burden.

The above equations are differentiable with respect to synaptic weights and presynaptic spike times. As will be shown in the following, this directly translates to solving the credit assignment problem and thus allows exact error propagation through networks of spiking neurons. For easier reading, we focus on one specific case ($\tau_\mathrm{m} = \tau_\mathrm{s}$), but the others can be treated analogously.

Exact error backpropagation with spikes Learning in SNNs requires the ability to relate efferent spiking to both afferent weights and spike times. For the output spike time of a neuron k with presynaptic partners i, the first relationship can be formally described by the derivative of the output spike time with respect to the presynaptic weights (Eqn. 5.22). Using certain properties of $\mathcal{W}$, we can find a simple expression that can, additionally, be made to depend on the output spike time t_k itself:

$$\frac{\partial t_k}{\partial w_{ki}} = -\frac{1}{a_1} \frac{\exp\left(\frac{t_i}{\tau_\mathrm{s}}\right)}{\mathcal{W}(z) + 1} \left(t_k - t_i\right) \ , \tag{5.4}$$

with a_1 and z representing functions of w_{ki} and t_i as defined in Eqns. (5.11) and (5.18). Using the output spike time as additional information optimizes learning in scenarios where the exact neuron parameters are unknown and the real output spike time differs from the one calculated under ideal assumptions, as discussed later.

Second, the capability to relate errors in the output spike time to errors in the input spike times allows us to recursively propagate changes from neurons to their presynaptic partners.

$$\frac{\partial t_k}{\partial t_i} = -\frac{1}{a_1}\frac{\exp\left(\frac{t_i}{\tau_\mathrm{s}}\right)}{\mathcal{W}(z)+1}\frac{w_{ki}}{\tau_\mathrm{s}}\left(t_k - t_i - \tau_\mathrm{s}\right) . \tag{5.5}$$

Together, Eqns. (5.4) and (5.5) effectively and exactly solve the credit assignment problem in appropriately parametrized LIF networks of arbitrary architecture.

We can now apply the findings above to study learning in a layered network. Figure 5.1c shows a schematic of our feedforward networks and their spiking activity. The input uses the same coding scheme as all other neurons: more prominent features are encoded by earlier spikes. The output of the network is defined by the identity of the label neuron that spikes first (Fig. 5.1d).

We denote by $t_k^{(l)}$ the output spike time of the kth neuron in the lth layer; for example, in a network with N layers, $t_n^{(N)}$ is the spike time of the nth neuron in the label layer. The weight projecting to the kth neuron of layer l from the ith neuron of layer $l-1$ is denoted by $w_{ki}^{(l)}$.

To apply the error backpropagation algorithm (Linnainmaa, 1970; Rumelhart et al., 1986), we choose a loss function that is differentiable with respect to synaptic weights and spike times. During learning, the objective is to maximize the temporal difference between the correct and all other label spikes. The following loss function fulfills the above requirements:

$$\begin{aligned} L[\mathbf{t}^{(N)}, n^*] &= \mathrm{dist}\left(t_{n^*}^{(N)}, t_{n\neq n^*}^{(N)}\right) \\ &= \log\left[\sum_n \exp\left(-\frac{t_n^{(N)} - t_{n^*}^{(N)}}{\xi\tau_\mathrm{s}}\right)\right] , \end{aligned} \tag{5.6}$$

where $\mathbf{t}^{(N)}$ denotes the vector of label spike times $t_n^{(N)}$, n^* the index of the correct label and $\xi \in \mathbb{R}^+$ is a scaling parameter. This loss function represents a cross entropy between the true label distribution and the softmax-scaled label spike times produced by the network (see Methods). Reducing its value therefore increases the temporal difference between the output spike of the correct label neuron and all other label neurons. Notably, it only depends on the spike time difference and is invariant under absolute time shifts, making it independent of the concrete choice of the experiment start which defines $t = 0$. In case of a non-spiking label neuron we treat its spike time as $t_n^{(N)} = \infty$. In this case however, the

equation Eqn. (5.2) is not defined and neither are its derivatives. We therefore introduce a simple, local heuristic to encourage spiking behavior in large portions of the network (see Methods). In some scenarios, learning can be facilitated by the addition of a spike-time-dependent regularization term (see Methods).
Gradient descent on the loss function Eqn. (5.6) can now be easily performed by repeated application of the chain rule. Using the exact derivatives Eqns. (5.4) and (5.5), this yields the synaptic plasticity rule

$$\begin{aligned}\Delta w_{ki}^{(l)} &\propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ki}^{(l)}} \\ &= -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}} \underbrace{\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_k^{(l)}}}_{\delta_k^{(l)}} = -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}} \sum_j \frac{\partial t_j^{(l+1)}}{\partial t_k^{(l)}} \delta_j^{(l+1)} .\end{aligned} \tag{5.7}$$

A compact formulation for hierarchical networks that highlights the backpropagation of errors can be found in Eqns. (5.38) to (5.40). In either form, only the label layer error and the neuron spike times are required for training, which can either be calculated using Eqn. (5.2) or by simulating (or emulating) the LIF dynamics (Eqn. 5.1).
The computational complexity of the synaptic plasticity rule — a potential limiting factor for on-chip implementations — can be drastically reduced by appropriate approximations. In the Supplementary Information SI.D we present early results using such an approach. Note that the simplification is only used in Supplementary Information SI.D and all other results we report in the following were produced using the full analytical equations Eqns. (5.4) and (5.5).

5.2.1 Simulations

After deriving the learning algorithm in the previous chapter, we show its classification capabilities in software simulations. In these simulations we demonstrate successful learning and provide a baseline for the hardware emulations that follow.
We use two data sets that emphasize different aspects of interesting real-world scenarios. As an example for low-dimensional, "continuous" data spaces, in which points belonging to different classes can be arbitrarily close together (thus making separation particularly challenging), we chose the Yin-Yang data set (Kriener et al., 2021a). For higher-dimensional, discrete input, we used the MNIST data set (LeCun et al., 1998) as a small-scale image classification scenario.
The results in this section are based on Eqn. (5.2) for calculating the spike times in the forward pass, and Eqn. (5.40) for calculating weight updates; for details regarding implementation see Methods. For hyperparameters of the discussed experiments see Tables SI.F1 and SI.F2.

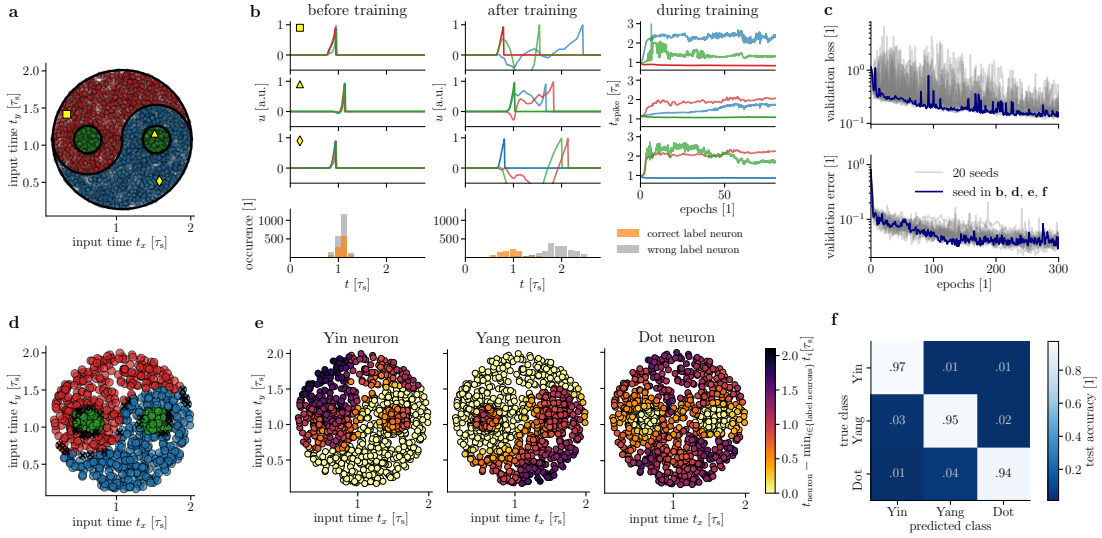

Figure 5.2: Classification of the Yin-Yang data set. (a) Illustration of the Yin-Yang data set. The samples are separated into three classes, Yin (●), Yang (●) and Dot (●). The yellow symbols (□, △, ◊) mark samples for which the training process is illustrated in (b). The input times t_x and t_y correspond to the spike time of the inputs associated with the x and y coordinates of individual samples. **(b)** Training mechanism for three exemplary data samples (cf. (a)). For the first three rows, the left and middle columns depict voltage dynamics in the label layer before and after training for 300 epochs, respectively. The voltage traces of the three label neurons are color-coded according to their corresponding class as in (a). Before training, the random initialization of the weights causes the label neurons to show similar voltage traces and almost indistinguishable spike times. After training there is a clear separation between the spike time of the correct label neuron and all others, with the correct neuron spiking first. The evolution of the label spike times during training is shown in the right column for the first 70 epochs. Bottom row: spike histograms over all training samples. Our learning algorithm induces a clear separation between the spike times of correct and wrong label neurons. **(c)** Training progress (validation loss as given in Eqn. (5.6) and error rate) over 300 epochs for 20 training runs with random initializations (gray). The run shown in panels b and d-f is plotted in blue. **(d)** Classification result on the test set (1000 samples). The color of each sample indicates the class determined by the trained network. The wrongly classified samples (marked with black X) all lie very close to the border between classes. **(e)** Spike times of the Yin, Yang and Dot neurons for all test samples after training. For each sample, spike times were normalized by subtracting the earliest spike time in the label layer. Bright yellow denotes zero difference, i.e., the respective label neuron was the first to spike and the sample was assigned to its class. The bright yellow areas resemble the shapes of the Yin, Yang and Dot areas, reflecting the high classification accuracy after training. **(f)** Confusion matrix for the test set after training.

Yin-Yang classification task: The first data set consists of points in the yin-yang figure (Fig. 5.2a). Each point is defined by a pair of Cartesian coordinates $(x, y) \in [0, 1]^2$. To build in redundancy and capture the intrinsic symmetry of the yin-yang motive, the data set is augmented with mirrored coordinates $(1 - x, 1 - y)$ enabling networks of neurons without trainable bias to learn the task (Kriener et al., 2021a). The three classes are labeled as per the respective area they occupy, i.e., Yin, Yang or Dot. This augmented data set was specifically designed to require latent variables for classification: a shallow non-spiking classifier reaches $(64.3 \pm 0.2)\%$ test accuracy, an ANN with one hidden layer of size 120 typically around $(98.7 \pm 0.3)\%$. Due to this large gap, our Yin-Yang data set represents an expressive test of error backpropagation in our hierarchical spiking networks. At the same time, it can be learned by networks that are compatible in size with the current revision of BrainScaleS-2 (Schemmel et al., 2020).

After translation of the four features to spike times (see Fig. 5.1 and Methods for more details), they were joined with a bias spike at fixed time, and these five spikes served as input to a network with 120 hidden and 3 label neurons. We illustrate the training mechanism with voltage traces for three samples belonging to different classes (Fig. 5.2b). The algorithm changes the weights to create a separation in the label spike times (cf. left and middle column) that corresponds to correct classification. Note that the voltage traces were just recorded for illustration, as only spike times are required for calculating weight updates. After 300 epochs our networks reached $(95.9 \pm 0.7)\%$ test accuracy for training with 20 different random seeds (Fig. 5.2c). The classification failed only for samples that were extremely close to the border between two classes (Fig. 5.2d). Figure 5.2e shows the spike times of the label neurons. These vary continuously for inputs belonging to other classes, but drop abruptly at the boundary of the area belonging to their own class, which denotes a clear separation — see, for example, the abrupt change from red (late spike time) to yellow (early spike time) of the Yin-neuron when moving from Yang to Yin (Fig. 5.2e, left panel).

MNIST classification task: To study the scalability of our approach to larger and more high-dimensional data sets, we applied it to the classification of MNIST handwritten digits (LeCun et al., 1998). Figure 5.3 shows training results for networks with 784–350–10 neurons, where pixel intensities were translated to spike times. During training, noise was added to the input samples to aid generalization, but no bias spikes were used. As seen in Fig. 5.3a, training converges for 10 different initial random seeds, reaching a final test accuracy of $(97.1 \pm 0.1)\%$. Similar results are also achieved for deeper architectures with multiple hidden layers (see Table SI.B1 for additional simulation runs with different network architectures).

For reference, we consider several other results obtained with spiking-time coding. In Mostafa (2017), a maximum test accuracy of 97.55% using a network with a hidden layer of 800 neurons is reported; note that this work uses non-leaky neurons with effectively infinite membrane memory. Also for non-leaky neurons, but using an approximative approach for calculating gradients, Kheradpisheh and Masquelier (2020) report 97.4% using 400 hid-

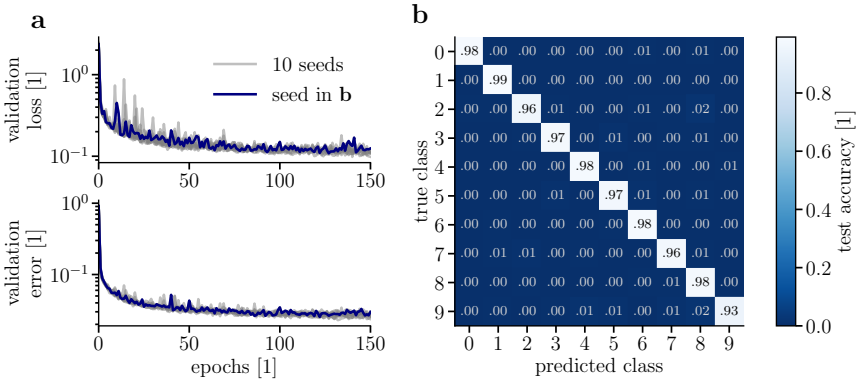

Figure 5.3: Classification of the MNIST data set. (a) Training progress of a network over 150 epochs for 10 different random initializations. The run drawn in blue is the one which produced the results in (b). **(b)** Confusion matrix for the test set after training.

den neurons. In Comsa et al. (2020), a maximum test accuracy of 97.96% was achieved using 340 hidden neurons, supported by a regular spike grid and extensive hyperparameter search.
We note that there also exist trial-averaging and spike-count-based approaches that have the benefit of more straight-forward learning rules, but these approaches sacrifice precision, neuronal real-estate or time-to-solution in comparison to frameworks based on the precise timing of single output spikes.
For example, Esser et al. (2015) report 92.7% using 512 neurons, while Tavanaei et al. (2018b) require 1000 hidden neurons to achieve 96.6%.

5.2.2 Fast neuromorphic classification

In our framework, the time to solution is a function of the network depth and the time constants τ_m and τ_s. Assuming typical biological timescales, most input patterns in the above scenario are classified within several milliseconds. By leveraging the speedup of neuromorphic systems such as BrainScaleS (Schemmel et al., 2010, 2020), with intrinsic acceleration factors of 10^3 to 10^4, the same computation can be achieved within microseconds. In the following, we present an implementation of our framework on BrainScaleS-2 and discuss its performance in conjunction with the achieved classification speed and energy consumption. For a proof-of-concept implementation on its predecessor BrainScaleS-1, we refer to Supplementary Information SI.A.
The advantages of such a neuromorphic implementation come at the cost of reduced control. Training needs to cope with phenomena such as spike jitter, limited weight range and granularity, as well as neuron parameter variability, among others. In general, an important aspect of any theory aiming for compatibility with physical substrates, be they biological or artificial, is its robustness to substrate imperfections; our results on BrainScaleS-2 im-

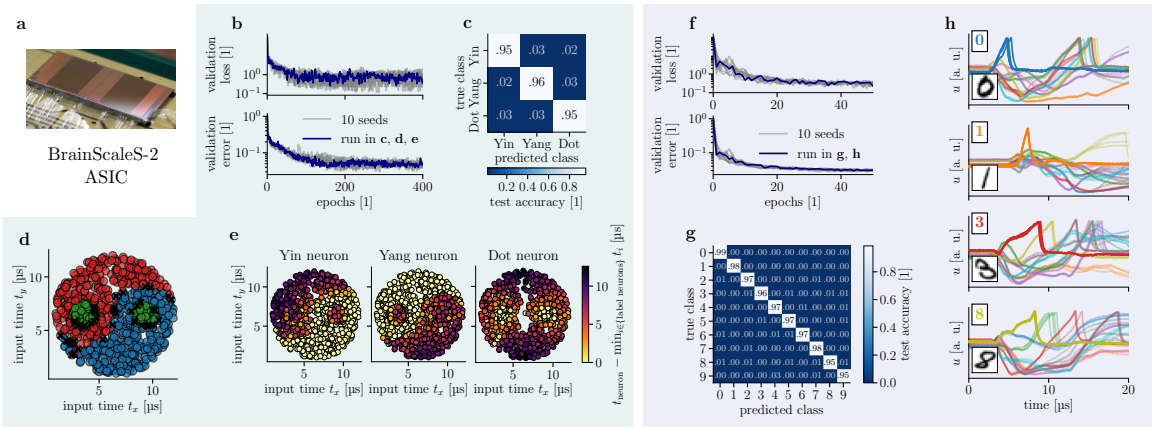

Figure 5.4: Classification on the BrainScaleS-2 neuromorphic platform. **(a)** Photograph of a BrainScaleS-2 chip. **(b-e) Yin-Yang data set (b)** Training progress over 200 epochs for 11 different random initializations. The run drawn in blue also produced the results shown in panel (b-d). **(c)** Confusion matrix for the test set after training. **(d)** Classification result on the test set. For each input sample the color indicates the class determined by the trained network. Wrong classifications are marked with a black X. The wrongly classified samples all lie very close to the border between two classes. **(e)** Separation of label spike times (cf. Fig. 5.2e). For each of the label neurons, bright yellow dots represent data samples for which it was the first to spike, thereby assigning them its class. Similarly to the software simulations, the bright yellow areas align well with the shapes of the Yin, Yang and Dot areas of the data set. **(f-h) MNIST data set (f)** Evolution of training over 50 epochs for 10 different random initializations. The run drawn in blue is the one which produced the results shown in panel (g) and (h). **(g)** Confusion matrix for the test set after training. **(h)** Exemplary membrane voltage traces on BrainScaleS-2 after training. Each panel shows color-coded voltage traces of four label neurons for one input that was presented repeatedly to the network (inlays show the input and its correct class). Each trace was recorded four times to point out the trial-to-trial variations.

plicitly represent a powerful demonstration of this property. To further substantiate the generalizability of our algorithm to different substrates, we complement our experimental results with a simulation study of various substrate-induced distortive effects.

Learning on BrainScaleS-2: BrainScaleS-2 is a mixed-signal accelerated neuromorphic platform with 512 physical neurons, each being able to receive inputs via 256 configurable synapses. These neurons can be coupled to form larger logical neurons with a correspondingly increased number of inputs. At the heart of each neuron is an analog circuit emulating LIF neuronal dynamics with an acceleration factor of 10^3 to 10^4 compared to biological timescales.

Due to variations in the manufacturing process, the realized circuits systematically deviate from each other (fixed-pattern noise). Although these variations can be reduced by calibrating each circuit (Aamir et al., 2018b), considerable differences remain (standard deviation on the order of 5 % on BrainScaleS-2) and pose a challenge for possible neuromorphic algorithms — along with other features of physical model systems such as spike time jitter

Table 5.1: Comparison of pattern recognition models on the MNIST data set emulated on neuromorphic back-ends, sorted by classification speed. For reference, an ANN running on GPU is included in the top row. Note that we include only references which present measurements for both energy and throughput in addition to accuracy. An extended table containing results with partial or estimated measurements can be found in Supplementary Information, Table SI.F3.

platform	type	technology	coding	input resolution	network size/structure	data augmentation/ regularization	energy per classification	classifications per second[1]	test accuracy	reference
Nvidia Tesla P100	digital	14 nm	ANN	28×28	CNN[2]	dropout	852 µJ	125 000	99.2 %	see SI.E
SpiNNaker	digital	130 nm	rate	28×28	784–600–500–10	noisy input encoding	3.3 mJ	91	95.0 %	Stromatias et al. (2015)
True North	digital	28 nm	rate	28×28	CNN	noisy input encoding	0.27 µJ	1000	92.7 %	Esser et al. (2015)
True North	digital	28 nm	rate	28×28	CNN	noisy input encoding	108 µJ	1000	99.4 %	Esser et al. (2015)
Loihi	digital	14 nm	bin. rate	(20×20)[3]	400–400–10	n.a.	2.5 µJ	5917	96.2 %	Renner et al. (2021)
unnamed (Intel)	digital	10 nm	temporal	(28×28)[4]	236–20	stochastic spike loss	1.0 µJ	6250	88.0 %	Chen et al. (2018)
BrainScaleS-2	mixed	65 nm	temporal	16×16	256–246–10	input noise	8.4 µJ	20 800	96.9 %	this work, see also SI.E

[1] Note that some platforms achieve a high number of classifications per second simply by processing a large number of samples in parallel, while other platforms rely on the sequential (but fast) processing of individual samples.
[2] Standard architecture given as an example in the PyTorch repository, for details see Supplementary Information SI.E.
[3] Four (empty) pixels on each margin are cropped to yield the 20×20 center from the 28×28 image.
[4] The 28×28 image is preprocessed using 5×5 Gabor-filters and 3×3 pooling before being sent into the chip.

or spike loss (Petrovici et al., 2014; Wunderlich et al., 2019; Kungl et al., 2019; Dold et al., 2019).

The chip's synaptic arrays were configured to support arbitrary fully-connected networks of up to 256 emulated neurons with a maximum of 256 inputs per neuron. Each such logical connection was realized via two physical synapses in order to allow transitions between an excitatory and an inhibitory regime. Synaptic weights on the chip are configurable with 6 bit precision. More details about our setup can be found in the Methods section.

We used an in-the-loop training approach (Schmitt et al., 2017; Kungl et al., 2019; Cramer et al., 2020a), where inference runs emulated on the neuromorphic substrate were interleaved with host-based weight update calculations. For emulating the forward pass, the spike times for each sample in a mini-batch were joined sequentially into one long spike train and then injected into the neuromorphic system via a field-programmable gate array (FPGA). The latter was also used to record the spikes emitted by the hidden and label layers.

Figure 5.4a-d shows the results of training a spiking network with 120 hidden neurons on BrainScaleS-2 on the Yin-Yang data set. The system quickly learned to discriminate between the presented patterns, with an average test accuracy of $(95.0 \pm 0.9)\%$.

The hardware emulation performs similarly to the software simulations (Fig. 5.2), with the wrong classifications still only happening along the borders of the areas with different labels (Fig. 5.4c). The remaining difference in performance after training is attributable to the substrate variability (cf. also Fig. 5.4h). Considering that one of the specific challenges built into the Yin-Yang data set resides in the continuity of its input space and abrupt class switch between bordering areas, this result highlights the robustness of our approach.

To classify the MNIST data set using the BrainScaleS-2 system, we emulated and trained a network of size 256–246–10 (Fig. 5.4f-h). Due to the restrictions imposed by the hardware on the input dimensionality, we used downsampled images of 16×16 pixels. Across multiple initializations, we achieved a test accuracy of $(96.9 \pm 0.1)\%$; similarly to the Yin-Yang data set, this is only slightly lower than in software simulations of equally sized networks (Table 5.2). The ability of our framework to achieve reliable classification despite such substrate-induced distortions is well-illustrated by post-training membrane dynamics measured on the chip Fig. 5.4h. In all cases shown here, the correct label neuron spikes before 10 µs and is clearly separable from all other label neurons.

Due to its short intrinsic time constants and overall energy efficiency, the BrainScaleS-2 system enables very fast and energy-efficient acquisition of classification results. Classification of the 10 000 MNIST test samples takes a total of 0.937 s, including data transmission, emulation of dynamics and return of the classification results. The total time on the BrainScaleS-2 chip was 480 ms, a detailed breakdown of the execution time is shown in Supplementary Information SI.E. The power consumption of the chip, measured during runtime, including all chip components needed for spike generation and processing (i.e., excluding the host and FPGA) amounted to 175 mW. For measurement details and scalability considerations we refer to Supplementary Information SI.E. This results in an average energy consumption

Table 5.2: Summary of the presented results. Accuracies are given as mean value and standard deviation. For comparison, on the Yin-Yang data set a linear classifier achieves $(64.3\ \ 0.2)$ % test accuracy, while a (non-spiking, not particularly optimized) ANN with 120 hidden neurons achieves $(98.7\ \ 0.3)$ %. As a reference for the MNIST data set we trained a 784–350–10 fully connected ANN which reached an average test accuracy of $(98.2\ \ 0.1)$ %. The results in this table were obtained without extensive hyperparameter tuning.

data set	**hidden neurons**	**accuracy** [%] **test**	**train**
Yin-Yang			
in SW	120	95.9 ± 0.7	96.3 ± 0.7
on HW	120	95.0 ± 0.9	95.3 ± 0.7
MNIST			
in SW	350	97.1 ± 0.1	99.6 ± 0.1
in SW ($\tau_s = 2\tau_m$)	350	97.2 ± 0.1	99.7 ± 0.1
MNIST 16×16			
in SW	246	97.4 ± 0.2	99.2 ± 0.1
on HW	246	96.9 ± 0.1	98.2 ± 0.1

of 8.4 µJ per classification. For a comparison to other neuromorphic platforms, we refer to Table 5.1.

Note that the networks on the other neuromorphic platforms differ in their architectures, coding schemes and training methods, and while we list some of these differences in the table, a direct comparison in terms of individual numbers remains difficult.

This table only includes references in which measurements for both classification rate and energy are reported. A more comprehensive overview, including studies that lack some of the above measurements, can be found in the Supplementary Information, Table SI.F3.

Our current experimental setup leaves room for significant optimization. For an estimation of possible improvements and their potential effect on classification rate and energy consumption, we refer to Supplementary Information SI.E and (Cramer et al., 2020a). With these improvements we expect to increase the classification rate by up to a factor of four while simultaneously decreasing the energy-per-classification value by up to a factor of 3.

5.2.3 Robustness of time-to-first-spike learning

As noted earlier, a learning scheme operating only on spike times combined with our coding represents a natural fit for neuromorphic hardware, both for requiring commonly accessible observables (i.e., spike times, as opposed to, e.g., membrane potentials or synaptic currents) and due to its intrinsic efficiency, as it emphasizes few and early spikes. An important indicator of a model's feasibility for neuromorphic emulation is its robustness

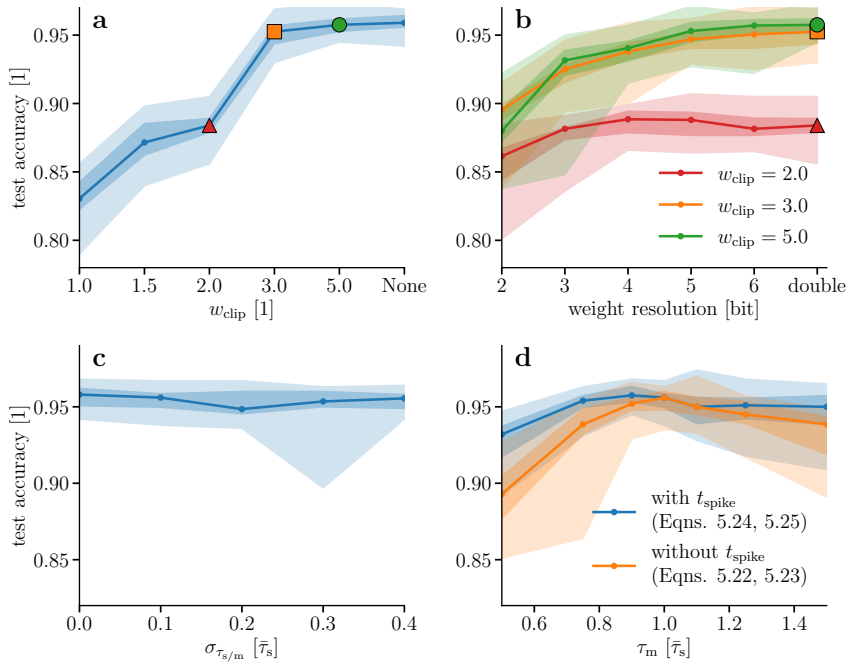

Figure 5.5: Effects of substrate imperfections. Modeled constraints were added artificially into simulated networks. All panels show median, quartiles, minimum, and maximum of the final test accuracy on the Yin-Yang data set for 20 different initializations. **(a)** Limited weight range. The weights were clipped to the range $[-w_{\text{clip}}, w_{\text{clip}}]$ during training and evaluation. The triangle, square and circle mark the clip values that are used in panel (b). **(b)** Limited weight resolution. For the three weight ranges marked in (a) the weight resolution was reduced from a double precision float value down to 2 bits. Here, n-bit precision denotes a setup where the interval $[-w_{\text{clip}}, w_{\text{clip}}]$ is discretized into $2 \cdot 2^n - 1$ samples (n weight bits plus sign). **(c)** Time constants with fixed-pattern noise. For these simulations each neuron received a random τ_s and τ_m independently drawn from the distribution $N(\bar{\tau}_s, \sigma_{\tau_{s/m}})$. This means that the ratio of time constants was essentially never the one assumed by the learning rule. **(d)** Systematic shift between time constants. Here τ_s was drawn from $N(\bar{\tau}_s, \sigma_{\tau_{s/m}})$ while τ_m was drawn from $N(\bar{\tau}_m, \sigma_{\tau_{s/m}})$ for each neuron for varying mean $\bar{\tau}_m$ and fixed $\sigma_{\tau_{s/m}} = 0.1\bar{\tau}_s$. The orange curve illustrates a training where the backward pass performs "naive" gradient descent, without using explicit information about output spike times. The blue curve, as all other panels, has the output spike time as an observable.

towards substrate-induced distortions. By experimentally demonstrating its capabilities on BrainScaleS-2, we have implicitly provided one substantive data point for our framework. Here, we present a more comprehensive study of the robustness of our approach.

Most physical neuronal substrates have several forms of variability in common (Petrovici, 2016, Chapter 5). In both digital and mixed-signal systems, synaptic weights are typically limited in both range and resolution. Additionally, parameters of analog neuron and synapse circuits exhibit a certain spread.

To study the impact of these effects, we included them in software simulations of our model applied to the Yin-Yang classification task.

In this context, we highlight the importance of a detail mentioned in the derivation of Eqn. (5.4). The output spike time given in Eqn. (5.2) depends only on neuron parameters, presynaptic spike times and weights, thus its derivatives share the same dependencies (Eqns. 5.22 and 5.23). With some manipulations, the equation for the actual output spike time can be inserted (Eqns. 5.24 and 5.25), producing a version of the learning rule that directly depends on the output spike time itself. This version thus allows the incorporation of additional information gained in the forward pass and is therefore expected to be significantly more stable, which is confirmed below.

Using dimensionless weight units (scaled by the inverse threshold), we observe that an upper weight limit of approximately 3 is sufficient for achieving peak performance (Figure 5.5a). This weight value is equivalent to a PSP that covers the distance between leak potential and firing threshold.

If this is not achievable within the typical parametrization range of a neuromorphic chip, the effective maximum weight to the hidden layer can be increased by multiplexing each input into the network (cf. Methods).

In the experiments with limited weight resolution (both in software and on hardware), a floating-point-precision "shadow" copy of synaptic weights was kept in memory. The forward and backward pass used discretized weight values, while the calculated weight updates were applied to the shadow weights (Hubara et al., 2017). Our model shows approximately constant performance for weight resolutions down to 5 bit, followed by gradual degradation below (Figure 5.5b).

Interestingly, adding variability to the synapse and membrane time constants has no discernible effects (Figure 5.5c). This is a direct consequence of having used the true output spike times for the learning rule in the backward pass. A comparison to "naive" gradient descent without this information is shown in (Figure 5.5d). These simulations show that the algorithm can be expected to adequately cope with a large amount of fixed-pattern noise on the time constants if the mean of the distributions for τ_m and τ_s match reasonably well with the values assumed by the learning rule (up to 10–20% difference).

Additionally, in Supplementary Information SI.C we investigate trained networks regarding their robustness to adverse effects that appear only after training, such as temperature-induced parameter variations or inactivation of neurons. Our simulations show that trained

networks can cope with such effects, suggesting that our training algorithm develops network structures robust even to distortions not present during training.
Finally, we note that all of the effects addressed above also have biological correlates. While not directly reflecting the variability of biological neurons and synapses, our simulations do suggest that biological variability does not present a fundamental obstacle to our form of TTFS computation.

5.3 Discussion

We have proposed a model of first-spike-time learning that builds on a rigorous analysis of neuro-synaptic dynamics with finite time constants and provides exact learning rules for optimizing first-spike times. The resulting form of synaptic plasticity operates on pre- and postsynaptic spike times and effectively solves the credit assignment problem in spiking networks; for the specific case of hierarchical feedforward topologies, it yields a spike-based form of error backpropagation. In this manuscript, we have applied this algorithm to networks with one and two hidden layers. Given the reported results, we are confident that our approach scales to even larger and deeper networks.
While TTFS coding is an exceptionally appealing paradigm for reasons of speed and efficiency, our approach is not restricted to this particular coding scheme. Our learning rules enable a rigorous manipulation of spike times and can be used for a variety of loss functions that target other relationships between spike timings. The time-to-first-spike scenario studied here merely represents the simplest, yet arguably also the fastest and most efficient paradigm for spike-based classification of static patterns. Additionally, our derived theory is applicable to more complex, e.g., recurrent, network structures and multi-spike coding schemes which are needed for processing temporal data streams.
First-spike coding schemes are particularly relevant in the context of biology, where decisions often have to be taken under pressure of time. The action to be taken in response to a stimulus can be considerably sped up by encoding it in first-spike times. In turn, such fast decision making on the order of $\sim 100\,\mathrm{ms}$ (Thorpe et al., 1996, 2001) will have a particularly sensitive dependence on exact spike times and thus require a corresponding precision of parameters.
At first glance, demands for precision appear at odds with the imperfect, variable nature of microscopic physical substrates, both biological and artificial. We met this challenge by incorporating output spike times directly into the backward pass. With this, the theoretical requirement of exact ratios of membrane to synaptic time constants is significantly softened, which greatly extends the applicability of our framework to a wide range of substrates, including, in particular, BrainScaleS-2.
By requiring only spike times, the proposed learning framework has minimal demands for neuromorphic hardware and becomes inherently robust towards substrate-induced distortions. This further enhances its suitability for a wide range of neuromorphic platforms.

Bolstered by the design characteristics of the BrainScaleS-2 system, our implementation achieves a time-to-classification of about 10 µs after receiving the first spike. Including relaxation between patterns and communication, the complete MNIST test set with 10 000 samples is classified in less than 1 s with an energy consumption of about 8.4 µJ per classification, which compares favorably with other neuromorphic solutions for pattern classification. The time characteristics of this implementation do not deteriorate for increased layer sizes because neurons communicate asynchronously and their dynamics are emulated independently. For the current incarnation of BrainScaleS-2, an increase in spiking activity only has a negligible effect on power consumption. Furthermore, for larger numbers of neurons we would expect only a weak increase of the power drain.

We also stress that, in contrast to, e.g., GPUs, our system was used to process input data sequentially. Our reported classification speed is thus a direct consequence of our coding scheme combined with the system's accelerated dynamics. Further increasing the throughput by parallelization (simultaneously using multiple chips) is straightforward and would not affect the required energy per classification.

Due to the complexity of our exact gradient-based rules, our hardware networks were trained using updates calculated off-chip based on emulated spike times. Early, promising simulations using a significantly simplified learning rule, however, suggest the possibility of an on-chip implementation of our framework. Furthermore, we note that our learning rules require three components that can all be made available at the locus of the synapse: pre- and post-synaptic spikes, as in classical spike-timing-dependent plasticity, and an error term, which could be propagated by mechanisms such as those proposed in, e.g., (Payeur et al., 2020; Sacramento et al., 2018). This raises the intriguing possibility for our framework to help explain learning in biological substrates as well.

Since, compared to the von-Neumann paradigm, artificial brain-inspired computing is only in its infancy, its range of possible applications still remains an open question. This is reflected by most state-of-the-art neuromorphic approaches to information processing, which, in order to accommodate a wide range of spike-based computational paradigms, aim for a large degree of flexibility in network topology and parametrization. Despite the obvious efficiency trade-off of such general-purpose platforms, we have shown that an embedded version of our framework can achieve a powerful combination of performance, speed, efficiency and robustness. This gives us confidence that a more specialized neuromorphic implementation of our model represents a competitive alternative to current solutions based on von-Neumann architectures, especially in edge computing scenarios.

5.4 Methods

Preliminaries In this section we derive the equations from the main manuscript, starting with the learning rule for $\tau_\mathrm{m} \to \infty$, then $\tau_\mathrm{m} = \tau_\mathrm{s}$, Eqn. (5.2) and finally $\tau_\mathrm{m} = 2\tau_\mathrm{s}$, Eqn. (5.3). The case $\tau_\mathrm{m} \to \infty$ has already been discussed in Mostafa (2017) and was reproduced here for completeness and comparison. Due to the symmetry in τ_m and τ_s of the PSP (Eqn. 5.14), the $\tau_\mathrm{m} = 2\tau_\mathrm{s}$ case describes the $\tau_\mathrm{m} = \frac{1}{2}\tau_\mathrm{s}$ case as well.
For each, a solution for the spike time T, defined by

$$u(T) = \vartheta, \tag{5.8}$$

has to be found, given LIF dynamics

$$u(t) = \frac{1}{C_\mathrm{m}} \frac{\tau_\mathrm{m}\tau_\mathrm{s}}{\tau_\mathrm{m} - \tau_\mathrm{s}} \sum_{\text{spikes } t_i} w_i \kappa(t - t_i)\,, \tag{5.9}$$

$$\kappa(t) = \theta(t) \left[\exp\left(-\frac{t}{\tau_\mathrm{m}}\right) - \exp\left(-\frac{t}{\tau_\mathrm{s}}\right) \right], \tag{5.10}$$

with membrane time constant $\tau_\mathrm{m} = C_\mathrm{m}/g_\ell$ and the PSP kernel κ given by a difference of exponentials. Here we already assumed our TTFS use case in which each neuron only produces one relevant spike and the second sum in Eqn. (5.1) reduces to a single term.
For convenience, we use the following definitions

$$a_n := \sum_{i \in C} w_i \exp\left(\frac{t_i}{n\tau_\mathrm{s}}\right), \tag{5.11}$$

$$b := \sum_{i \in C} w_i \frac{t_i}{\tau_\mathrm{s}} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right), \tag{5.12}$$

with summation over the set of causal presynaptic spikes $C = \{i \mid t_i < T\}$.
In practice, this definition of the causal set C is not a closed-form expression because the output spike time T depends explicitly on C. However, it can be computed straightforwardly by iterating over the ordered sets of input spike times (for n presynaptic spikes there are n sets $\tilde{C}_i$ each comprising of the i first input spikes). For each set $\tilde{C}_i$ one calculates an output spike time T_i and determines if this happens later than the last input of this set and before the next input (the $i+1$th input spike). The earliest such spike T_i is the actual output spike time and the corresponding $\tilde{C}_i$ is the correct causal set. If no such causal set $\tilde{C}_i$ exists, the neuron did not spike and we assign it the spike time $T = \infty$.

nLIF learning rule for $\tau_\mathrm{m} \to \infty$ With this choice of τ_m, the first term in Eqn. (5.10) becomes 1 and we recover the nLIF case discussed in (Mostafa, 2017). Given the existence

of an output spike, in Eqn. (5.8) the spike time T appears only in one place and simple reordering yields

$$\frac{T}{\tau_\mathrm{s}} = \ln\left[\frac{a_1}{a_\infty - \vartheta C_\mathrm{m}/\tau_\mathrm{s}}\right], \tag{5.13}$$

where we used Eqn. (5.11) for $n = 1$ and $n = \infty$, the latter being the sum over the weights.

Learning rule for $\tau_\mathrm{m} = \tau_\mathrm{s}$ According to l'Hôpital's rule, in the limit $\tau_\mathrm{m} \to \tau_\mathrm{s}$ Eqn. (5.9) becomes a sum over α-functions of the form

$$u(t) = \frac{1}{C_\mathrm{m}} \sum_i w_i \theta(t - t_i) \cdot (t - t_i) \exp\left(-\frac{t - t_i}{\tau_\mathrm{s}}\right). \tag{5.14}$$

Using these voltage dynamics for the equation of the spike time Eqn. (5.8), together with the definitions Eqns. (5.11) and (5.12) and $\tau_\mathrm{m} = C_\mathrm{m}/g_\ell$, we get the equation

$$0 = g_\ell \vartheta \exp\left(\frac{T}{\tau_\mathrm{s}}\right) + \underbrace{b - a_1 \frac{T}{\tau_\mathrm{s}}}_{=:y}. \tag{5.15}$$

The variable y is introduced to bring the equation into the form

$$h \exp(h) = z \tag{5.16}$$

which can be solved with the differentiable Lambert W function $h = \mathcal{W}(z)$. The goal is now to bring Eqn. (5.15) into this form, this is achieved by reformulation in terms of y

$$0 = g_\ell \vartheta \exp\left(\frac{b}{a_1}\right) \exp\left(-\frac{y}{a_1}\right) + y \tag{5.17}$$

$$\underbrace{\frac{y}{a_1}}_{=:\,h} \exp\left(\frac{y}{a_1}\right) = \underbrace{-\frac{g_\ell \vartheta}{a_1} \exp\left\{\left(\frac{b}{a_1}\right)\right\}}_{=:\,z}. \tag{5.18}$$

With the definition of the Lambert W function the spike time can be written as

$$\frac{T}{\tau_\mathrm{s}} = \frac{b}{a_1} - \mathcal{W}\left[-\frac{g_\ell \vartheta}{a_1} \exp\left(\frac{b}{a_1}\right)\right]. \tag{5.19}$$

Branch choice: Given that a spike happens, there will be two threshold crossings: One from below at the actual spike time, and one from above when the voltage decays back to the leak potential (Fig. SI.F1a,b). Correspondingly, the Lambert W function (Fig. SI.F1c,d) has two real branches (in addition to infinite imaginary ones), and we need to choose the branch

that returns the earlier solution. In case the voltage is only tangent to the threshold at its maximum, the Lambert W function only has one solution.

For choosing the branch in the other cases we need to look at h from the definition, i.e.

$$h = \frac{y}{a_1} = \frac{b}{a_1} - \frac{T}{\tau_\mathrm{s}}. \tag{5.20}$$

In a setting with only one strong enough input spike, the summations in a_n and b reduce to yield $h = (t_i - T)/\tau_\mathrm{s}$. Because the maximum of the PSP for $\tau_\mathrm{m} = \tau_\mathrm{s}$ occurs at $t_i + \tau_\mathrm{s}$, we know that the spike must occur at $T \leq t_i + \tau_\mathrm{s}$ and therefore

$$-1 \leq \frac{t_i - T}{\tau_\mathrm{s}} = h. \tag{5.21}$$

This corresponds to the branch cut of the Lambert W function meaning we must choose the branch with $h \geq -1$. For a general setting, if we know a spike exists, we expect a_n and b to be positive. In order to get the earlier threshold crossing, we need the branch that returns the larger $\mathcal{W}$ (Fig. SI.F1d), that is where $\mathcal{W} = h > -1$.

Derivatives: The derivatives for t_i in the causal set $i \in C$ come down to

$$\begin{aligned}
&\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) \qquad (5.22)\\
&\quad = \frac{\tau_\mathrm{s}}{a_1} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) \left[z\mathcal{W}'(z) + \left(\frac{t_i}{\tau_\mathrm{s}} - \frac{b}{a_1}\right)(1 - z\mathcal{W}'(z))\right], \\
&\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) \qquad (5.23)\\
&\quad = \frac{w_i}{a_1} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) \left[1 + \left(\frac{t_i}{\tau_\mathrm{s}} - \frac{b}{a_1}\right)(1 - z\mathcal{W}'(z))\right].
\end{aligned}$$

A crucial step is to reinsert the definition of the spike time where it is possible (cf. Fig. 5.5d). For this we need the derivative of the Lambert W function $z\mathcal{W}'(z) = \frac{\mathcal{W}(z)}{\mathcal{W}(z)+1}$ that follows from differentiating its definition Eqn. (5.16) with $h = \mathcal{W}(z)$ with respect to z. With this equation one can calculate the derivative of Eqn. (5.19) with respect to incoming weights and times as functions of presynaptic weights, input spike times and output spike time:

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}, T) = -\frac{1}{a_1} \frac{1}{\mathcal{W}(z) + 1} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right)(T - t_i), \tag{5.24}$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}, T) = -\frac{1}{a_1} \frac{1}{\mathcal{W}(z) + 1} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) \frac{w_i}{\tau_\mathrm{s}}(T - t_i - \tau_\mathrm{s}). \tag{5.25}$$

These equations are equivalent to the Eqns. (5.4) and (5.5) shown in the main text.

Learning rule for $\tau_\mathrm{m} = 2\tau_\mathrm{s}$ Inserting the voltage (Eqn. 5.9) into the spike time (Eqn. 5.8) yields

$$g_\ell \vartheta = \exp\left(-\frac{T}{\tau_\mathrm{m}}\right) \sum_{i \in C} w_i \exp\left(\frac{t_i}{\tau_\mathrm{m}}\right) - \exp\left(-\frac{T}{\tau_\mathrm{s}}\right) \sum_{i \in C} w_i \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) . \tag{5.26}$$

Reordering and rewriting this in terms of a_1, a_2, and τ_s (with $\tau_\mathrm{m} = 2\tau_\mathrm{s}$) we get

$$0 = -a_1 \left[\exp\left(-\frac{T}{2\tau_\mathrm{s}}\right)\right]^2 + a_2 \exp\left(-\frac{T}{2\tau_\mathrm{s}}\right) - g_\ell \vartheta \,. \tag{5.27}$$

This is written such that its quadratic nature becomes apparent, making it possible to solve for $\exp(-T/2\tau_\mathrm{s})$ and thus

$$\frac{T}{\tau_\mathrm{s}} = 2 \ln \left[\frac{2a_1}{a_2 + \sqrt{a_2^2 - 4a_1 g_\ell \vartheta}}\right] . \tag{5.28}$$

Branch choice: The quadratic equation has two solutions that correspond to the voltage crossing at spike time and relaxation towards the leak later; again, we want the earlier of the two solutions. It follows from the monotonicity of the logarithm that the earlier time is the one with the larger denominator. Due to an output spike requiring an excess of recent positively weighted input spikes, a_n are positive, and the $+$ solution is the correct one.

Derivatives: Using the definition $x = \sqrt{a_2^2 - 4a_1 g_\ell \vartheta}$ for brevity, the derivatives of Eqn. (5.28) are

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) = 2\tau_\mathrm{s} \left[\frac{1}{a_1} + \frac{2g_\ell \vartheta}{(a_2 + x)x}\right] \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) - \frac{2\tau_\mathrm{s}}{x} \exp\left(\frac{t_i}{2\tau_\mathrm{s}}\right) , \tag{5.29}$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) = 2w_i \left[\frac{1}{a_1} + \frac{2g_\ell \vartheta}{(a_2 + x)x}\right] \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) - \frac{w_i}{x} \exp\left(\frac{t_i}{2\tau_\mathrm{s}}\right) . \tag{5.30}$$

Again, inserting the output spike time yields

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}, T) \tag{5.31}$$
$$= \frac{2\tau_\mathrm{s}}{a_1}\left[1 + \frac{g_\ell \vartheta}{x}\exp\left(\frac{T}{2\tau_\mathrm{s}}\right)\right]\exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) - \frac{2\tau_\mathrm{s}}{x}\exp\left(\frac{t_i}{2\tau_\mathrm{s}}\right),$$
$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}, T) \tag{5.32}$$
$$= \frac{2w_i}{a_1}\left[1 + \frac{g_\ell \vartheta}{x}\exp\left(\frac{T}{2\tau_\mathrm{s}}\right)\right]\exp\left(\frac{t_i}{\tau_\mathrm{s}}\right) - \frac{w_i}{x}\exp\left(\frac{t_i}{2\tau_\mathrm{s}}\right).$$

Error backpropagation in a layered network Our goal is to update the network's weights such that they minimize the loss function $L[\mathbf{t}^{(N)}, n^*]$. For weights projecting into the label layer, updates are calculated via

$$\Delta w_{ni}^{(N)} \propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ni}^{(N)}} = -\frac{\partial t_n^{(N)}}{\partial w_{ni}^{(N)}}\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_n^{(N)}}. \tag{5.33}$$

The weight updates of deeper layers can be calculated iteratively by application of the chain rule:

$$\Delta w_{ki}^{(l)} \propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ki}^{(l)}} = -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}}\delta_k^{(l)}, \tag{5.34}$$

where the second term is a propagated error that can be calculated recursively with a sum over the neurons in layer $(l+1)$:

$$\delta_k^{(l)} := \frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_k^{(l)}} = \sum_j \frac{\partial t_j^{(l+1)}}{\partial t_k^{(l)}}\delta_j^{(l+1)}. \tag{5.35}$$

In the following we treat the $\tau_\mathrm{m} = \tau_\mathrm{s}$ case but the calculations can be performed analogously for the other cases. Rewriting Eqns. (5.24) and (5.25) in a layer-wise setting, the derivatives

of the spike time for a neuron k in arbitrary layer l are

$$\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) \tag{5.36}$$
$$= -\frac{1}{a_1} \exp\left(\frac{t_i^{(l-1)}}{\tau_\mathrm{s}}\right) \frac{1}{\mathcal{W}(z)+1} \left(t_k^{(l)} - t_i^{(l-1)}\right) ,$$
$$\frac{\partial t_k^{(l)}}{\partial t_i^{(l-1)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) \tag{5.37}$$
$$= -\frac{1}{a_1} \exp\left(\frac{t_i^{(l-1)}}{\tau_\mathrm{s}}\right) \frac{1}{\mathcal{W}(z)+1} \frac{w_{ki}^{(l)}}{\tau_\mathrm{s}} \left(t_k^{(l)} - t_i^{(l-1)} - \tau_\mathrm{s}\right) .$$

Inserting Eqns. (5.35) to (5.37) into Eqns. (5.33) and (5.34) yields a synaptic learning rule which implements exact error backpropagation on spike times.
This learning rule can be rewritten to resemble the standard error backpropagation algorithm for ANNs:

$$\boldsymbol{\delta}^{(N)} = \frac{\partial L}{\partial \boldsymbol{t}^{(N)}} , \tag{5.38}$$
$$\boldsymbol{\delta}^{(l-1)} = \left(\widehat{\boldsymbol{B}}^{(l)} - 1\right) \odot \boldsymbol{\rho}^{(l-1)} \odot \left(\boldsymbol{w}^{(l),T} \boldsymbol{\delta}^{(l)}\right) , \tag{5.39}$$
$$\Delta \boldsymbol{w}^{(l)} = -\eta \tau_\mathrm{s} \left(\boldsymbol{\delta}^{(l)} \boldsymbol{\rho}^{(l-1),T}\right) \odot \widehat{\boldsymbol{B}}^{(l)} , \tag{5.40}$$

where $\odot$ is the element-wise product, the T-superscript denotes the transpose of a matrix and $\boldsymbol{\delta}^{(l-1)}$ is a vector containing the backpropagated errors of layer $(l-1)$. The individual elements of the tensors above are given by

$$\rho_i^{(l)} = -\frac{1}{a_1} \exp\left(\frac{t_i^{(l)}}{\tau_\mathrm{s}}\right) \frac{1}{\mathcal{W}(z)+1} , \tag{5.41}$$
$$\widehat{B}_{ki}^{(l)} = \frac{t_k^{(l)} - t_i^{(l-1)}}{\tau_\mathrm{s}} . \tag{5.42}$$

BrainScaleS-2 The application-specific integrated circuit (ASIC) is built around an analog neuromorphic core which emulates the dynamics of neurons and synapses. All state variables, such as membrane potentials and synaptic currents, are physically represented in their respective circuits and evolve continuously in time. Considering the natural time constants of such integrated analog circuits, this emulation takes place at 1000-fold accelerated time scales compared to the biological nervous system. One BrainScaleS-2 chip features 512 adaptive exponential leaky integrate-and-fire (AdEx) neurons, which can be freely config-

ured; these circuits can be restricted to LIF dynamics as required by our training framework (Aamir et al., 2018a). Both the membrane and synaptic time constants were calibrated to 6 µs.

Each neuron circuit is connected to one of four synapse matrices on the chip, and integrates stimuli from its column of 256 CuBa synapses (Friedmann et al., 2017). Each synapse holds a 6 bit weight value; its sign is shared with all other synapses located on the same synaptic row. The presented training scheme, however, allows weights to continuously transition between excitation and inhibition. We therefore allocated pairs of synapse rows to convey the activity of single presynaptic partners, one configured for excitation, the other one for inhibition.

Synapses receive their inputs from an event routing module allowing to connect neurons within a chip as well as to inject stimuli from external sources. Events emitted by the neuron circuits are annotated with a time stamp and then sent off-chip. The neuromorphic ASIC is accompanied by a FPGA to handle the communication with the host computer. It also provides mechanisms for low-latency experiment control including the timed release of spike trains into the neuromorphic core. The FPGA is furthermore used to record events and digitized membrane traces originating from the ASIC. BrainScaleS-2 only permits recording one membrane trace at a time. Each membrane voltage shown in Fig. 5.4h therefore originates from a different repetition of the experiment.

The ASIC is controlled by a layered software stack (Müller et al., 2020) which exposes the necessary interfaces to a high-level user via Python bindings. These were used in our framework that is described in the following.

Simulation software Our experiments were performed using custom modules for the deep learning library PyTorch (Paszke et al., 2019). The network module implements layers of LIF neurons whose spike times are calculated according to Eqn. (5.2). This method of determining the spike times of the neurons is fastest, but also memory-intensive. An alternative implementation integrates the dynamical equations of the LIF neurons in a layer, which also yields the neuron spike times. Even though both approaches are technically equivalent, this method is slower and should only be employed if the computing resources are limited.

The activations passed between the layers during the forward pass are the spike times. The equations describing the weight updates for the network (Eqn. 5.40) are realized in a custom backward-pass module for the network.

Training and regularization methods In order to train a given data set using our learning framework, the input data has to be translated into spike times first. We do this by defining the times of the earliest and latest possible input spike t_{early} and t_{late} and mapping the range of input values linearly to the time interval $[t_{\text{early}}, t_{\text{late}}]$.

If the data set requires a bias to be solvable, our framework allows its addition. These bias spikes essentially represent additional input spikes for a layer, which have the same spike time for any input. The weights from the neurons to these "bias sources" is learned in the same way as all the other synaptic weights. For the Yin-Yang data set, the addition of a bias spike facilitated training. For some samples, due to the low number of inputs, the relatively low activity that is received by the network is spread out over a long time interval. The additional spike in the middle of the available interval decreases the maximum distance between input spikes for the hidden layer. In contrast, the MNIST data set has a much higher input dimensionality and the spikes are more distributed over the input time interval. Therefore, the activity provided to the hidden layer at any point in time is high even without additional bias.

Implementing our learning algorithm as custom PyTorch modules allows us to use the training architecture provided by the library. The simulations were performed using mini-batch training in combination with with the Adam optimizer (Kingma and Ba, 2014) and learning rate scheduling (the parameters can be found in Tables SI.F1 and SI.F2).

To assist learning we employ several regularization techniques. The term

$$+\alpha\left[\exp\left(t_{n^*}^{(N)}/\beta\tau_{\mathrm{s}}\right)-1\right]$$

with scaling parameters $\alpha, \beta \in \mathbb{R}^+$, can be added to the loss in Eqn. (5.6). This regularizer further pushes the correct neuron towards earlier spike times.

Gaussian noise on the input spike times can be used to combat overfitting. This proved beneficial for the training of the MNIST data set.

Weight updates Δw with absolute value larger than a given hyperparameter are set to zero to compensate divergence for vanishing denominator in Eqn. (5.40).

As noted previously, the weight update equations are only defined for neurons that elicit a spike. To prevent fully quiescent networks we add a hyperparameter which controls how many neurons without an output spike are allowed. If the portion of non-spiking neurons is above this threshold, we increase the input weights of the silent neurons. In case of multiple layers where this applies, only the first such layer with insufficient spikes is boosted. If neurons in a layer are too inactive multiple times in direct succession, the boost to the weights increases exponentially.

Training on hardware In principle our training framework can be used to train any neuromorphic hardware platform that (i) can receive a set of input spikes and yield the output spike times of all neurons in the emulated network and (ii) can update the weight configuration on the hardware according to the calculated weight updates. In our framework the hardware replaces the computed forward-pass through the network. For the calculation of the loss and the following backward pass, the hardware output spikes are treated as if they

had been produced by a forward pass in simulation. The backward pass is identical to pure simulation.
As accessible value ranges of neuron parameters are typically determined by the hardware platform in use, a translation factor between the neuron parameters and weights in software and the parameters realized on hardware needs to be determined. In our experiments with BrainScaleS-2 the translation between hardware and software parameter domain was determined by matching of PSP shapes and spike times predicted by a software forward pass to the ones produced by the chip.
The implicit assumption of having only the first spike emitted by every neuron be relevant for downstream processing can effectively be ensured by using a long enough refractory period. Since the only information-carrying signal that is not reset upon firing is the synaptic current, which is forgotten on the time scale of τ_s, we found that, in practice, setting the refractory time $\tau_{ref} > \tau_s$ leads to most neurons eliciting only one spike before the classification of a given input pattern.
For training the Yin-Yang data set on BrainScaleS-2, having only five inputs proved insufficient due to the combination of limited weights and neuron variability. We therefore multiplexed each logical input into five physical spike sources, totalling 25 inputs spikes per pattern. Adding further copies of the inputs effectively increased the weights for each individual input. This method has the added benefit of averaging out some of the effects of the fixed-pattern noise on the input circuits as multiple of them are employed for the same task.

Data availability

We used the MNIST LeCun et al. (1998) and the Yin-Yang data set Kriener et al. (2021a), for the latter code is available at `https://github.com/lkriener/yin_yang_data_set`.

Code availability

Code for the simulations Göltz et al. (2021) is available at `https://github.com/JulianGoeltz/fastAndDeep`.

Acknowledgment

We wish to thank Jakob Jordan and Nico Gürtler for valuable discussions, Sebastian Schmitt for his assistance with BrainScaleS-1, Vitali Karasenko, Philipp Spilger and Yannik Stradmann for taming physics, as well as Mike Davies and Intel for their ongoing support (LK, WS, MAP). Some calculations were performed on UBELIX, the HPC cluster at the University of Bern. Our work has greatly benefitted from access to the Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858. Some

simulations were performed on the bwForCluster NEMO, supported by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963–1 FUGG. We gratefully acknowledge funding from the European Union for the Human Brain Project under grant agreements 604102 (JS, KM, MAP), 720270 (SB, OB, BC, JS, KM, MAP), 785907 (SB, OB, BC, WS, JS, KM, MAP), 945539 (LK, AB, SB, OB, BC, WS, JS, MAP) and the Manfred Stärk Foundation (JG, AB, DD, AFK, KM, MAP).

Author contributions

JG, AB and MAP designed the conceptual and experimental approach. JG derived the theory, implemented the algorithm, and performed the hardware experiments. LK embedded the algorithm into a comprehensive training framework and performed the simulation experiments. AB and OJB offered substantial software support. SB, BC, JG and AFK provided low-level software for interfacing with the hardware. JG, LK, DD, SB and MAP wrote the manuscript.

Competing Interests statement

The authors declare no competing interests.

5.5 Supplementary Information

SI.A Learning with time-to-first-spike (TTFS) coding on BrainScaleS-1

To demonstrate the applicability of our approach to different neuromorphic substrates, we also tested it on the BrainScaleS-1 system (Schemmel et al., 2010). This version of BrainScaleS has a very similar architecture to BrainScaleS-2, but its component chips are interconnected through post-processing on their shared wafer (wafer-scale integration). More importantly for our coding scheme and learning rules, its circuits emulate conductance-based (CoBa) instead of CuBa neurons. Furthermore, due to the different fabrication technology and design choices (in particular, the floating-gate parameter memory, see Srowig et al., 2007; Schemmel et al., 2010; Koke, 2017), the parameter variability and spike time jitter are significantly higher than on BrainScaleS-2 (Schmitt et al., 2017).
The training procedure was analogous to the one used on BrainScaleS-2 although using a different code base. To accommodate the CoBa synapse dynamics, we introduced global weight scale factors that modeled the distance between reversal and leak potentials and the total conductance, which were multiplied to the synaptic weights to achieve a CuBa. This approximation could then be trained with our learning rules. Despite this approximation and the considerable substrate variability, our framework was able to compensate well and classify the data set (Fig. SI.A1) correctly after only few training steps.

SI.B Additional experiments

In addition to the simulation results collected in Table 5.2 we provide additional training results on the MNIST data set here (Table SI.B1). We quantify the effect of noisy input spike times on generalization by comparing a noiseless training run and a run with input noise, both using the hyperparameters shown in Table SI.F1. Additionally, we train a network with a larger hidden layer as well as a deeper network with two hidden layers. Finally, we illustrate the effect of the weight quantization on the training of the MNIST data set by using the same 6-bit quantization as on the BrainScaleS-2.

SI.C Robustness to post-training variations

We have already shown that our learning mechanism is able to cope well with noise and parameter variability which are present during training (Figs. 5.4 and 5.5). In addition to these distortions which can be accounted for by the learning mechanism, it is interesting to measure the performance of the trained network under adverse effects that were not present during training. This is especially relevant for analog circuits where, for example, temperature changes can lead to shifts in the analog neuron parameters. We model this effect by training 10 networks on the MNIST data set using the ideal parameters of $\vartheta = 1$ and $\tau_\mathrm{s} = \tau_\mathrm{m} = 1$ for the neuron threshold and time constant and then evaluating their performance on the test data set for shifted values of the threshold and time constant (Fig. SI.C1

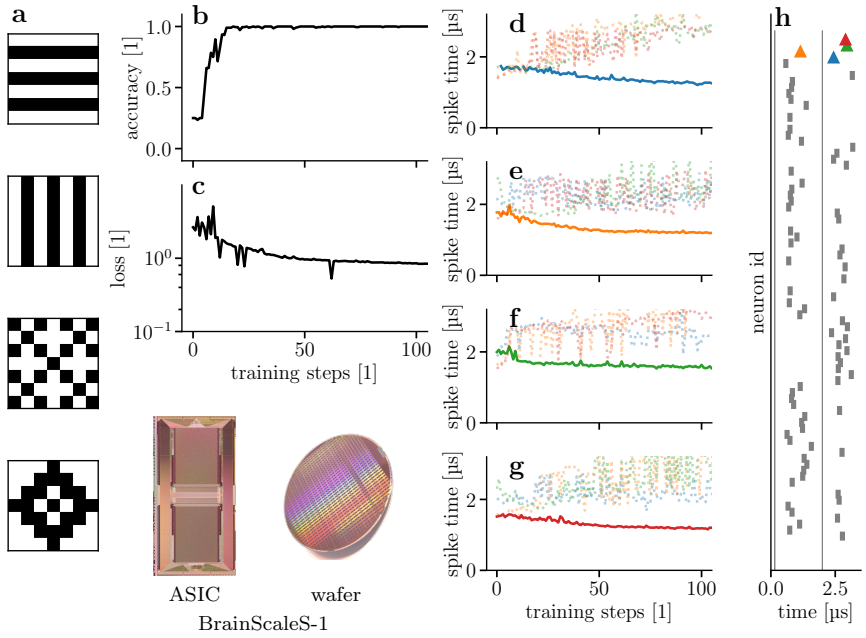

Figure SI.A1: Training a spiking network on the wafer-scale BrainScaleS-1 system. (a) Simple data set consisting of 4 classes with 7×7 input pixels. Accuracy **(b)** and loss **(c)** during training of the four pattern data set. **(d-g)** Evolution of the spike times in the label layer for the four different patterns. In each, the neuron coding the correct class is shown with a solid line and in full color. **(h)** Raster plot for the second pattern (e, correct class ▲) after training.

Table SI.B1: Additional simulation runs on the MNIST data set. The values given as the baseline are taken from Table 5.2. With the noted exception of training length. Apart from the number of training epochs (see footnotes), the hyperparameters for simulations with the input resolution of 28×28 are the same as in Table SI.F1 and the simulations for the input resolution of 16×16 used the hyperparameters given in Table SI.F2.

simulation	**input resolution**	**hidden neurons**	**accuracy** [%] **test**	**train**
baseline	28×28	350	97.1 ± 0.1	99.6 ± 0.1
without noise	28×28	350	95.7 ± 0.3	99.7 ± 0.1
larger hidden layer	28×28	800	97.3 ± 0.1	99.8 ± 0.1
two hidden layers[1]	28×28	400–400	97.1 ± 0.1	99.5 ± 0.1
baseline[2]	16×16	246	97.4 ± 0.2	99.2 ± 0.1
6-bit weight resolution[2]	16×16	246	97.3 ± 0.1	99.1 ± 0.1

[1] This network was trained for 300 epochs.
[2] This network was trained for 150 epochs.

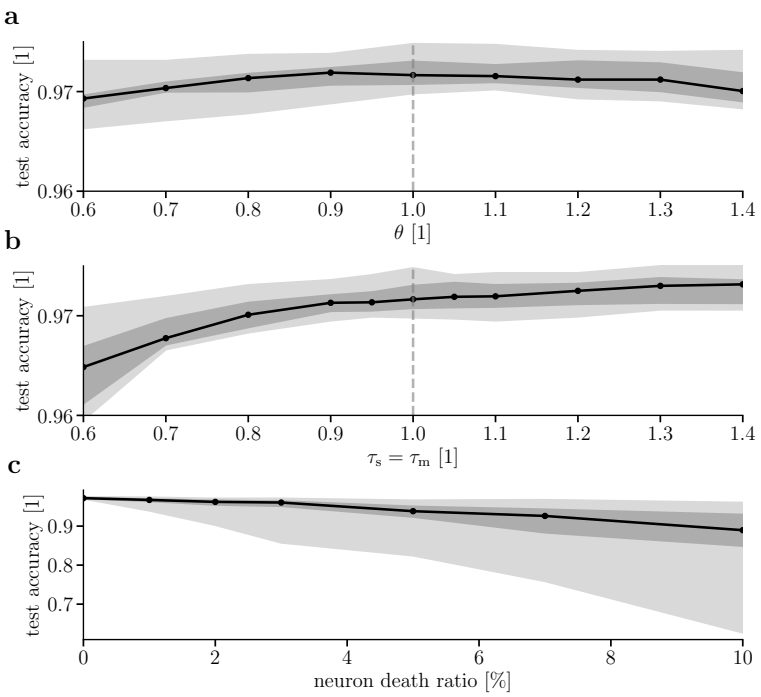

Figure SI.C1: Robustness to variations not present during training. All panels show median (black), quartiles (dark gray), as well as the entire range between minimum and maximum (light gray) in the shaded regions. **(a)** Dependence of test accuracy for evaluation for 10 trained networks with shifted threshold value θ. **(b)** Test accuracies for shifts in the neuron time constant τ_s and τ_m. **(c)** Influence of random deletion of hidden neurons on test accuracies. For each neuron death ratio, 10 different random sets of hidden neurons were deleted. These ten deletion sets were applied to the same ten networks as in (a) and (b).

a, b). These simulations show that the trained networks cope well, even if the relative shifts to the parameters are much larger than what can be typically expected due to temperature changes on BrainScaleS-2.

Furthermore, we consider a scenario which is less likely on neuromorphic platforms, but may be more relevant in biological networks. In biology, neural networks have to be robust against the death of neuron cells within the network. For each of the 10 fully trained networks we delete a percentage of its hidden population and evaluate the performance on the test set. As the consequences of this procedure strongly depend on exact choice of the deleted neurons, we repeat each deletion scenario for each network 10 times with different random seeds. Figure SI.C1c shows that networks trained with our learning mechanism exhibit stability towards sudden neuron death after training and even for 5 % neuron death the bound of the second quartile is still at 92.3 % accuracy. Note also that if plasticity is ongoing, the network can learn to recover much of its performance after apoptosis.

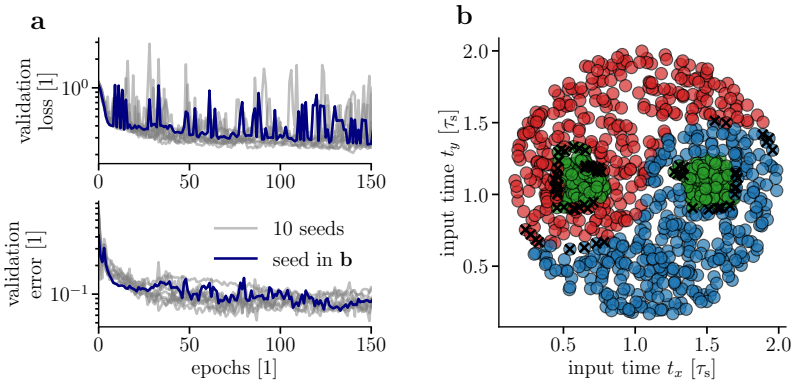

Figure SI.D1: Training on the Yin-Yang data set with a simplified learning rule. We approximated the learning rule to have less complex updates (Eqns. SI.D1 and SI.D2). **(a)** shows the training process of 150 epochs. We reach a test accuracy of (91.7 ± 1.4) % and training accuracy (91.7 ± 1.2) % averaged over 10 seeds. **(b)** shows the classification as in Fig. 5.2 after training for the highlighted training in (a).

SI.D Simplification of the learning rule

The learning rule for $\tau_\mathrm{m} = \tau_\mathrm{s}$ described in the main paper and derived in the Methods is computationally rather demanding: it needs multiple evaluations of the exponential function as well as an evaluation of the Lambert W function $\mathcal{W}$, for which no closed form exists. As the computational complexity of plasticity mechanisms on many neuromorphic chips is limited, we investigate the possibility of approximating the derivatives Eqns. (5.4) and (5.5) by replacing the exponential functions as well as $\mathcal{W}$ by a constant λ[3]:

$$\frac{\partial t_k}{\partial w_{ki}} = -\lambda \left(t_k - t_i\right) , \tag{SI.D1}$$

$$\frac{\partial t_k}{\partial t_i} = -\lambda \frac{w_{ki}}{\tau_\mathrm{s}} \left(t_k - t_i - \tau_\mathrm{s}\right) . \tag{SI.D2}$$

The approximated version consists only of simple differences and multiplications making this learning rule more amenable for on-chip implementations.
To examine the approximated learning rule in the standard setup with $\tau_\mathrm{m} = \tau_\mathrm{s}$ we chose $\lambda = 0.0192$ by evaluating $\frac{1}{a_1}\frac{1}{\mathcal{W}(z)+1}$ for a few inputs into the hidden layer. Using this extreme simplification we trained a network to classify the Yin-Yang data set (Fig. SI.D1). While the network learned the task correctly and achieved a test accuracy of (91.7 ± 1.4) %, this represents a small but noticeable drop in accuracy compared to the full learning rule (Table 5.2). We also observed that these simplified rules led to more instability for longer training periods (not shown here). Nonetheless, these promising results give us confidence

[3] This effectively leads to ρ being a constant in Eqns. (5.39) and (5.40).

that that a more careful choice of the constant or a replacement with a simple, but non-constant term can alleviate these problems while retaining a simple form of the learning rule.
Note, in particular, that Eqn. (SI.D2) explicitly contains the term $t_i + \tau_s$. This term represents the maximum of a PSP and changes sign when the output spike at T happens before versus after the maximum. This simple difference captures the major non-monotonic relationship in the time derivative. As the maximum of the PSP is given by a closed form solution $t_{max} = t_i + \frac{\tau_m \tau_s}{\tau_s - \tau_m} \log \frac{\tau_s}{\tau_m}$ for arbitrary combinations of τ_s and τ_m, it seems natural to investigate a slightly altered learning rule for different time constants.

SI.E Power consumption and execution time measurements

Table 5.1 in the main manuscript compares the energy consumption and classification speed of our model on BrainScaleS-2 with other neuromorphic platforms and an ANN on a GPU. This section details how the power and classification speed measurements were performed, as well as their implications for the scalability of and potential improvements to our setup. Additionally, we present our measurement technique for the GPU reference.

BrainScaleS-2

Power breakdown The full BrainScaleS-2 chip consumed a total of 175 mW measured during runtime with the INA219 chip Tex (2015). This overall figure encompasses the chip's high-speed communication links (approx. 60 mW), the digital periphery as well as its clocking infrastructure (approx. 80 mW), and the biasing of analog circuits (approx. 35 mW). Importantly, we could not observe a significant change in power consumption between the network during inference and an emulation of an inactive network. This implies that the cost of event transport and synaptic processing is negligible on the reported scales and that the overall figure would not be impacted by increased activity levels. As inactive synapses mostly contribute to the overall power consumption through negligible leakage currents, the power consumption would not be impacted by an increase of the neuron circuit's fan-in that would allow the training on larger input spaces.

Execution time breakdown We define the round-trip time for an on-chip inference run as starting before the forward pass through the network in our PyTorch implementation and ending when all classification results produced by the chip are available in PyTorch. For the classification of the full MNIST test data set on BrainScaleS-2 we measured a round-trip time of 0.937 s.
Due to this conservative definition of the round-trip time, our measurement includes a significant amount of time on the host (for data pre- and post-processing) and for communication between host and the neuromorphic system. Fig. SI.E1 shows a detailed breakdown of the execution time. We see that once the data arrives on the chip, it takes 480 ms to process

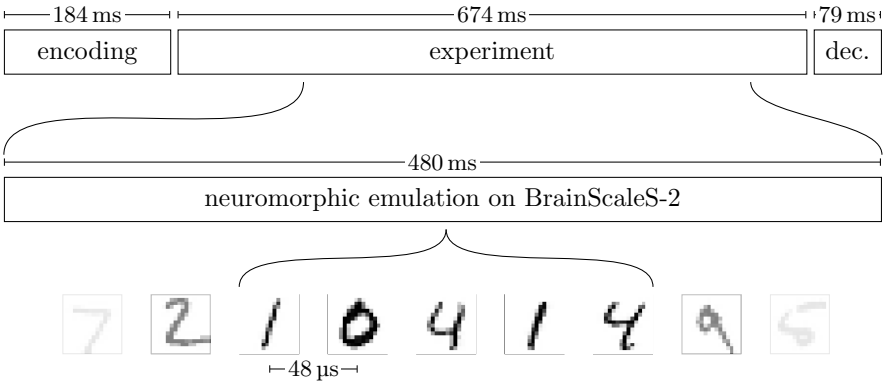

Figure SI.E1: Breakdown of the execution time for inference on the MNIST test set. The total time of about one second consists of an encoding, an experiment and a decoding phase. The encoding phase includes the translation of PyTorch tensors into spike trains and the encoding of the spike trains into instructions for the neuromorphic chip. In the experiment phase the instructions are sent from the host to the chip, the emulation is performed and the results are read out from the chip and communicated back to the host. In the final decoding phase the emulations results are converted back to PyTorch tensors.

the 10 000 images of the test set. This results in a classification every 48 µs or equivalently a classification rate of 20 800 images per second.
Considering the relevant hardware time constants of $\tau_s \approx \tau_m \approx 6\,\mu s$ and the typical time to solution of around $1\,\tau_s$ to $1.5\,\tau_s$, a classification duration per sample of 48 µs seems surprisingly long. This is owed to the sequential presentation of data samples to the network, for which we need to ensure that all residual activity — membrane voltages as well as synaptic currents — from the last sample has fully decayed before the next sample is presented. Currently, this is achieved by simply waiting between samples, but Cramer et al. (2020a) present an alternative: The plasticity processing unit (PPU) is able to trigger a reset of all membrane voltages and synaptic currents on the chip. Using this mechanism, Cramer et al. (2020a) shorten the classification time per image to 11.8 µs. The usage of artificial resets would also be a viable optimization for our model. It would require the previously switched off PPU to be activated and would therefore slightly increase the power consumption (by approximately 20 mW). This increase in power would however be more than compensated by the approximately quadrupled sample throughput.

GPU

For comparison to conventional computing hardware we add power and classification speed measurements on a Nvidia Tesla P100 GPU to Table 5.1. For the measurements on the GPU we use the convolutional neural network given as standard example in the PyTorch repository (`https://github.com/pytorch/examples/blob/507493d7b5fab51d55af88c5df9eadceb144fb67/mnist/main.py`).

The power measurements are performed using the tool nvidia-smi which is running in the background while in the foreground we run a PyTorch program which repeats the classification of the MNIST test data set (in one batch of size 10 000) for 10 times. Figure SI.E2 shows the power consumption over the full program runtime. We see that the GPU is only active for 10 short periods, the duration of which match the measured times during which the PyTorch program uses the GPU (Fig. SI.E2 b). The power consumption is calculated as an average over these intervals, resulting in 106.5 W.
The speed measurements were performed using time measurements in Python and averaged over the 10 classifications, resulting in a classification time per image of 8 µs. This amounts to an energy-per-classification value of 852 µJ.
As an additional reference we attempted to determine the power consumption and classification speed for a fully connected network with a hidden layer of 246 neurons (same size as the hidden layer on BrainScaleS-2) on GPU. However, due to the fact that the classification was a factor of 20 to 25 faster than for the CNN, we were not able to measure the power in a fine enough resolution with nvidia-smi to yield reliable values. To estimate a lower-bound for the energy per classification in this case, we can take the power consumption of the GPU in the phases where it was not actively used in the CNN measurement (i.e. power values between the peaks in Fig. SI.E2a) which is approximately 34 W. This "idle" power consumption for the CNN case seemed to approximately match the averaged power drain for the fully connected network. This amounts to a lower-bound estimate of the energy-per-classification value on the order of 10 µJ.

SI.F Additional data

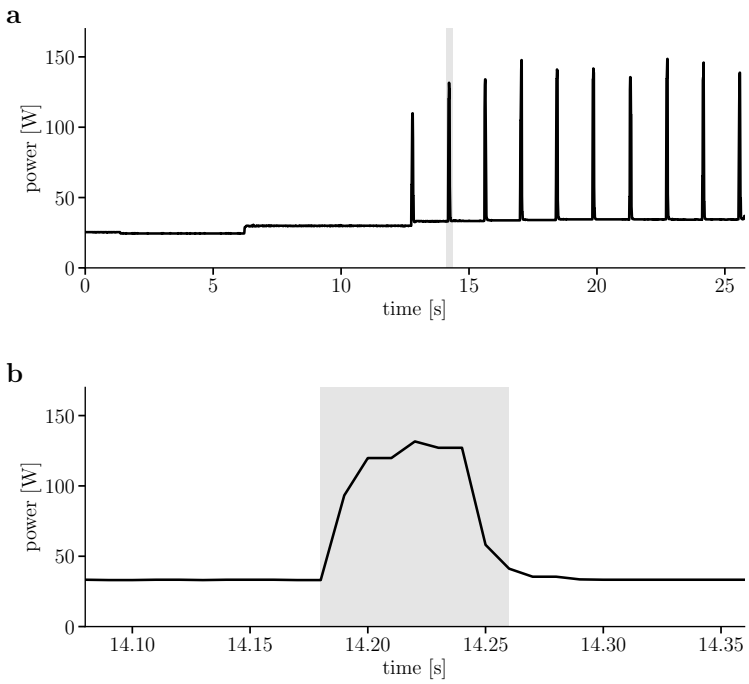

Figure SI.E2: Power consumption of Nvidia Tesla P100 GPU during classification of MNIST test data. (a) Power consumption of a standard PyTorch network for MNIST classification while running inference on the test data set for 10 times. **(b)** Zoom on a peak in the power consumption. The shaded area corresponds to the time during which the GPU is actively used (measured from within Python).

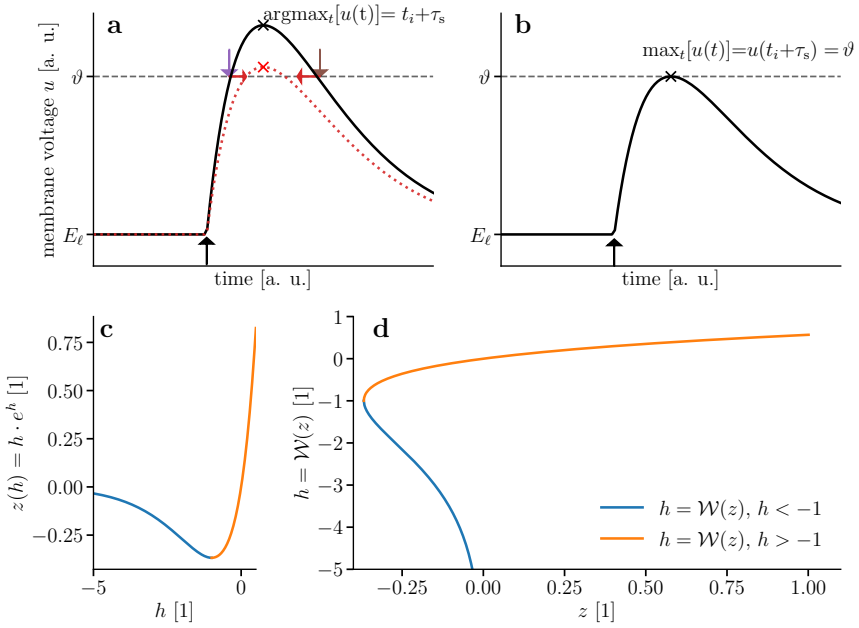

Figure SI.F1: **(a)** Membrane dynamics for one strong input spike at t_i (upward arrow) with two threshold crossings due to leak pullback (earlier violet, later brown). The change induced by a reduction of the input weight is shown in red. **(b)** Edge case without crossing and exactly one time where $u(t) = \vartheta$. **(c)** Defining relation for the Lambert W function $\mathcal{W}$, evidently not an injective map. **(d)** Distinguishing between $h \lessgtr -1$ allows to define the inverse function of (c), the Lambert W function $\mathcal{W}$.

Table SI.F1: Neuron, network and training parameters used to produce the results in Figs. 5.2 and 5.3.

Parameter name	Yin-Yang	MNIST
Neuron parameters		
g_ℓ	1.0	1.0
E_ℓ	0.0	0.0
ϑ	1.0	1.0
τ_m	1.0	1.0
τ_s	1.0	1.0
Network parameters		
size input	5	784
size hidden layer	120	350
size output layer	3	10
bias time[1]	$[0.9\tau_\mathrm{s}, 0.9\tau_\mathrm{s}]$	no bias
weight init mean[1]	$[1.5, 0.5]$	$[0.05, 0.15]$
weight init stdev[1]	$[0.8, 0.8]$	$[0.8, 0.8]$
t_early	$0.15\tau_\mathrm{s}$	$0.15\tau_\mathrm{s}$
t_late	$2.0\tau_\mathrm{s}$	$2.0\tau_\mathrm{s}$
Training parameters		
training epochs	300	150
batch size	150	80
optimizer	Adam	Adam
Adam parameter β	$(0.9, 0.999)$	$(0.9, 0.999)$
Adam parameter ϵ	10^{-8}	10^{-8}
learning rate	0.005	0.005
lr-scheduler	StepLR	StepLR
lr-scheduler step size	20	15
lr-scheduler γ	0.95	0.9
input noise σ	no noise	0.3
max ratio missing spikes[1]	$[0.3, 0.0]$	$[0.15, 0.05]$
max allowed Δw	0.2	0.2
weight bump value	0.0005	0.005
α	0.005	0.005
ξ [2]	0.2	0.2

[1] Parameter given layer wise [hidden layer, output layer].

[2] ξ implemented differently in code-base developed by the authors.

Table SI.F2: Network and training parameters for training on BrainScaleS-2 used to produce the results in Fig. 5.4. In contrast to Table SI.F1, the neuron parameters are not given here, as they are determined by the used chip.

Parameter name	Yin-Yang	16×16 MNIST
Network parameters		
size input	25	256
size hidden layer	120	246
size output layer	3	10
bias time[1]	$[0.9\tau_s, \text{no bias}]$	no bias
weight init mean[1]	$[0.1, 0.075]$	$[0.01, 0.006]$
weight init stdev[1]	$[0.12, 0.15]$	$[0.03, 0.1]$
t_{early}	$0.15\tau_s$	$0.15\tau_s$
t_{late}	$2.0\tau_s$	$2.0\tau_s$[3]
Training parameters		
training epochs	400	50
batch size	40	50
optimizer	Adam	Adam
Adam parameter β	$(0.9, 0.999)$	$(0.9, 0.999)$
Adam parameter ϵ	10^{-8}	10^{-8}
learning rate	0.002	0.003
lr-scheduler	StepLR	StepLR
lr-scheduler step size	20	10
lr-scheduler γ	0.95	0.9
input noise σ	no noise	0.3
max ratio missing spikes[1]	$[0.3, 0.05]$	$[0.5, 0.5]$
max allowed Δw	0.2	0.2
weight bump value	0.0005	0.005
α	0.005	0.005
ξ [2]	0.2	0.2

[1] Parameter given layer wise [hidden layer, output layer].

[2] ξ implemented differently in code-base developed by the authors.

[3] After translation of pixel values to spike times, inputs spikes with $t_{\text{input}} = t_{\text{late}}$ were not sent into the network.

Table SI.F3: Extension of literature review for pattern recognition models on neuromorphic back-ends, including results which do not detail certain measurements.

platform	type	coding	network size/structure	energy per classification	classifications per second	test accuracy	reference
SpiNNaker	digital	rate	764–600–500–10	3.3 mJ	91	95.0 %	Stromatias et al. (2015)
True North[1]	digital	rate	CNN	0.27 µJ	1000	92.7 %	Esser et al. (2015)
True North[1]	digital	rate	CNN	108 µJ	1000	99.4 %	Esser et al. (2015)
FPGA (nLIF neurons)[2]	digital	temporal	784–600–10	—	—	96.8 %	Mostafa et al. (2017)
Loihi	digital	bin. rate	400–400–10	2.5 µJ	5917	96.2 %	Renner et al. (2021)
Loihi[3]	digital	temporal	1920–10	—	—	96.4 %	Lin et al. (2018)
unnamed (Intel)[4]	digital	temporal	236–20	1.0 µJ	6250	88.0 %	Chen et al. (2018)
unnamed (Intel)[5]	digital	temporal	784–1024–512.-10	12.4 µJ	—	98.2 %	Chen et al. (2018)
unnamed (Intel)[5]	digital	temporal	784–1024–512–10	1.7 µJ	—	97.9 %	Chen et al. (2018)
SPOON[6]	digital	temporal	CNN	0.3 µJ	8547	97.5 %	Frenkel et al. (2020)
BrainScaleS-2	mixed	temporal	256–246–10	8.4 µJ	20 800	96.9 %	this work

[1] In Esser et al. (2015) it is stated that "The instrumentation available measures active power for the network in operation and leakage power for the entire chip, which consists of 4096 cores. We report energy numbers as active power plus the fraction of leakage power for the cores in use.". For the first result 5 cores were used, while the second result requires 1920 cores.

[2] No energy or speed measurements reported.

[3] No energy or speed measurements reported. Images were preprocessed with an algorithm described as "using scan-line encoders".

[4] Images preprocessed with 4 5×5 Gabor filters and 3×3 pooling.

[5] No speed measurements reported.

[6] Reported energy values are pre-silicon simulations.

Chapter 6

Result III: Towards dendritic microcircuits on neuromorphic hardware

6.1 Introduction

In this chapter we employ an opposite approach to the one in Chapter 5: There, we took the leaky integrate-and-fire (LIF) neuron which is commonly utilized on neuromorphic platforms as starting point and derived an error backpropagation method for its spike times. Here, we start from an already existing form of error backpropagation and make it more amenable to a neuromorphic implementation. As the design of neuromorphic platforms is inspired by the brain, they share some of the constraints that biology imposes. One example for this is the requirement for plasticity to be local in space and time. Thus, biologically plausible models, in addition to forming hypotheses about learning in the brain, provide a better starting point for neuromorphic implementations, as they inherently take some neuromorphic constraints into account. However, these models (a collection of them can be found in Section 2.4.3) are typically not designed with an immediate neuromorphic implementation in mind and are therefore not directly portable to currently existing neuromorphic platforms[1].
Out of the many available models we choose the dendritic microcircuits of Sacramento et al. (2018) because it is able to approximate the error backpropagation algorithm (see Appendix A.2) while also addressing all of the aspects of biological implausibility detailed in Section 2.4.3:

- Through the usage of feedback alignment (FA) the dendritic microcircuits do not suffer from the weight transport problem.

[1]The exception here is Tang et al. (2021), however even in this case the algorithm is only approximately portable to the Loihi platform and suffers a severe loss in achieved accuracy on the task.

- There are no separate forward- and backward propagation phases. Through the recurrency of the dendritic microcircuit network information flows forward and backward simultaneously. There is also no phasing in plasticity, so the network constantly learns.

- All synaptic weight updates are local in space and time. The synaptic update rules only rely on information available within the pre- and postsynaptic neurons at the time of the update.

- Errors are not sent through the network via synaptic connections but instead computed locally from different firing rates arriving at the neuron. This also avoids the issue of having to communicate negative values via firing rates.

- The microcircuits consist of multi-compartment neurons with leaky dynamics which makes them more realistic than the artificial neurons in an artificial neural network (ANN).

Nevertheless, the dendritic microcircuits are not directly portable to neuromorphic platforms. In the following we will evaluate the dendritic microcircuit in detail (Section 6.2), highlight the challenges for the implementability on neuromorphic systems (Section 6.2.3) and address the drawbacks step-by-step to make it more practically feasible (Sections 6.3 to 6.5).

6.2 Biologically plausible error backpropagation in dendritic microcircuits

The article *Dendritic cortical microcircuits approximate the backpropagation algorithm* was published by Sacramento et al. in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*[2]. Since the rest of Chapter 6 builds on and extends this model, we here lay the groundwork with a detailed description of the model in the state of the original publication (Sacramento et al., 2018). This includes not only the dynamics and functional principles of the model as stated in the publication but also our assumptions on implementation details where information was incomplete. We also reiterate how the model approximates error backpropagation in a from that is more easily transferable to the content of the following sections than the approach chosen in Sacramento et al. (2018). Finally, we highlight key points which need to be addressed to improve the model's practical feasibility and neuromorphic implementability. Note that for consistency with the publications in Section 6.3 and Section 6.4, we here use a notation that differs from the original publication.

[2] Sacramento et al. (2018)

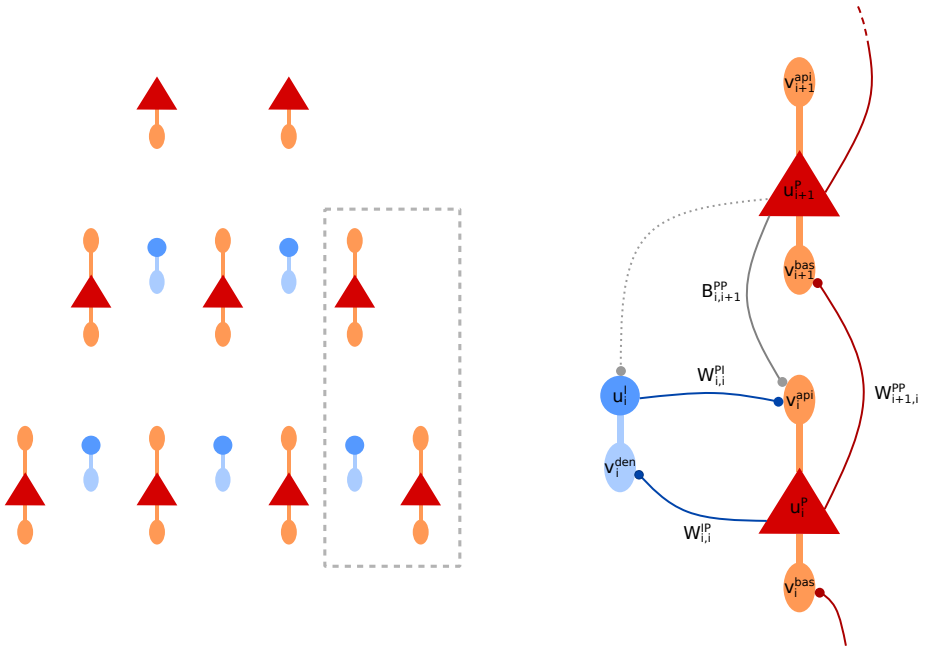

Figure 6.1: Schematic of a dendritic microcircuit network. Left: Layered network of dendritic microcircuits. Information travels from bottom to top. The pyramidal neurons are drawn in red (soma) and orange (dendrites) while the interneurons are colored in blue. In contrast to the hidden layer pyramidal neurons, the ones in the top layer only have a basal dendrite and no apical dendrite. The gray box marks the area drawn in detail on the right. Right: Single microcircuit consisting of a pyramidal neuron (red) in layer i and $i+1$ and an interneuron (blue). The lines between neurons represent synaptic connections, where the lateral connections are drawn in blue, the backward connections in gray and the forward connections in red. For clarity, the schematic of the connections is simplified here: In general, solid lines represent all-to-all connections between neurons, i.e. $\boldsymbol{W}^{\text{PP}}_{i+1,i}$, $\boldsymbol{W}^{\text{IP}}_{i,i}$, $\boldsymbol{W}^{\text{PI}}_{i,i}$ and $\boldsymbol{B}^{\text{PP}}_{i,i+1}$ are matrices. For example $\boldsymbol{W}^{\text{PP}}_{i+1,i}$ connects all pyramidal neurons in layer i to all pyramidal $i+1$, but here we only draw the one synapse connecting the two pyramidal neurons in the magnified drawing. The dotted connection represents the one-to-one nudging from a higher layer pyramidal neuron to the interneuron which is supposed to mimic it.

6.2.1 Summary of network-, neuron- and plasticity model

Network structure and dynamics

Networks of dendritic microcircuits as illustrated in Fig. 6.1 contain three types of multi-compartment neurons: the interneuron, the pyramidal neurons in the hidden layers of the network and the pyramidal neurons in the output layer of the network which differ slightly from their counterparts in the lower layers. A pyramidal neuron in a hidden layer has two dendritic compartments: the basal compartment which receives bottom-up input and the apical compartment which receives top-down and lateral input from the interneurons

in the same layer. An interneuron has only one dendritic compartment but receives an additional input which we call a nudging current and which will be detailed later. The top-layer pyramidal neurons, in contrast to their lower-layer counterpart, do not have an apical compartment since there are no interneurons in the top-layer nor another layer above from which an apical compartment could receive input (Fig. 6.1). Instead, these pyramidal neurons also receive a nudging current which is used to provide a learning target to the network. The differential equations describing the somatic membrane dynamics of the three neuron types are similar

$$C_\mathrm{m}\dot{\boldsymbol{u}}_\ell^\mathrm{P} = g_\mathrm{l}\left(E_\mathrm{l} - \boldsymbol{u}_\ell^\mathrm{P}\right) + g^\mathrm{bas}\left(\boldsymbol{v}_\ell^\mathrm{bas} - \boldsymbol{u}_\ell^\mathrm{P}\right) + g^\mathrm{api}\left(\boldsymbol{v}_\ell^\mathrm{api} - \boldsymbol{u}_\ell^\mathrm{P}\right) \ , \tag{6.1}$$

$$C_\mathrm{m}\dot{\boldsymbol{u}}_\ell^\mathrm{I} = g_\mathrm{l}\left(E_\mathrm{l} - \boldsymbol{u}_\ell^\mathrm{I}\right) + g^\mathrm{den}\left(\boldsymbol{v}_\ell^\mathrm{den} - \boldsymbol{u}_\ell^\mathrm{I}\right) + i^{\text{nudge, I}} \ , \tag{6.2}$$

$$C_\mathrm{m}\dot{\boldsymbol{u}}_N^\mathrm{P} = g_\mathrm{l}\left(E_\mathrm{l} - \boldsymbol{u}_N^\mathrm{P}\right) + g^\mathrm{bas}\left(\boldsymbol{v}_N^\mathrm{bas} - \boldsymbol{u}_N^\mathrm{P}\right) + i^{\text{nudge, tgt}} \tag{6.3}$$

where the respective somatic voltage u is driven by a leak g_l, conductive coupling with $g^{\text{bas/den/api}}$ to the dendritic voltages $v^{\text{bas/den/api}}$ and for the interneuron and the top-layer pyramidal a nudging current i. The index ℓ or N denotes the layer of the neurons. Note that the bold notation indicates a vector that contains the voltages of all neurons in a layer. The dendritic voltages v are instantaneous functions of their input

$$\boldsymbol{v}_\ell^\mathrm{bas} = \boldsymbol{W}_{\ell,\ell-1}^\mathrm{PP}\varphi\left(\boldsymbol{u}_{\ell-1}^\mathrm{P}\right) \ , \tag{6.4}$$

$$\boldsymbol{v}_\ell^\mathrm{api} = \boldsymbol{B}_{\ell,\ell+1}^\mathrm{PP}\varphi\left(\boldsymbol{u}_{\ell+1}^\mathrm{P}\right) + \boldsymbol{W}_{\ell,\ell}^\mathrm{PI}\varphi\left(\boldsymbol{u}_\ell^\mathrm{I}\right) \ , \tag{6.5}$$

$$\boldsymbol{v}_\ell^\mathrm{den} = \boldsymbol{W}_{\ell,\ell}^\mathrm{IP}\varphi\left(\boldsymbol{u}_\ell^\mathrm{P}\right) \tag{6.6}$$

where $\boldsymbol{W}_{\ell,\ell-1}^\mathrm{PP}$ is the synaptic weight matrix from the pyramidal neurons in layer $\ell-1$ to the pyramidal neurons in layer ℓ, $\boldsymbol{B}_{\ell,\ell+1}^\mathrm{PP}$ is the matrix of backward pyramidal to pyramidal connections from layer $\ell+1$ to layer ℓ and $\boldsymbol{W}_{\ell,\ell}^\mathrm{PI}$ and $\boldsymbol{W}_{\ell,\ell}^\mathrm{IP}$ are lateral connections from the interneurons to the pyramidal neurons and the other way around within the same layer. The weight matrices are multiplied by the firing rates of the presynaptic neurons which are given as a non-linear function of the presynaptic somatic voltage $\varphi(u)$. To simplify the notation we will in the following assume without loss of generality a membrane capacitance of $C_\mathrm{m} = 1$ and a leak potential of $E_\mathrm{l} = 0$.

Plasticity

Plasticity in the dendritic microcircuit model is implemented via a rate-based variant of the Urbanczik-Senn plasticity rule (Urbanczik and Senn, 2014). In this rule the bottom-up synapses w which connect to a dendritic compartment are learned with

$$\dot{w} = \eta\ \left[\varphi(u) - \varphi(v^*)\right]\ r_\mathrm{in} \tag{6.7}$$

where u is the somatic and v dendritic voltage and r_{in} the presynaptic firing rate. v^* is a scaled version of the dendritic voltage. The scaling ensures that, in the case where the soma receives no other input than the one from its dendritic compartment, v^* matches the somatic voltage u. In that case we can see that Eqn. (6.7) would result in zero weight change. If an additional input is now applied to the soma that is proportional to an error signal $u^{\text{tgt}} - u$, the somatic voltage u is closer to its target than if the additional input was not present (which is represented by v^*). This process is called nudging the somatic voltage towards its target. Now the difference calculated in the plasticity rule is no longer zero and results in a weight change. The changing weight converges to a value that allows the dendritic input into the soma alone to produce a somatic voltage which matches the target.

The plasticity in the network of dendritic microcircuits is divided into two stages: The first stage starts from a completely random weight initialization and learns what is called the self-predicting state. In that state the lateral connectivity in the network is set up such that, in the second stage, error signals are correctly propagated in the network and the feedforward weights can learn to solve a task. Note that this first stage of network setup is only used once before the actual training starts. Once the self-predicting state is reached and the learning of a task has begun, the network automatically retains the self-predicting state.

To learn the self-predicting state from a random initialization, the network receives random inputs but is not given any target or error signal. The self-predicting state is characterized by two requirements: Firstly, each interneuron mimics (i.e. produces the same output) as its corresponding pyramidal neuron in the layer above (see Fig. 6.1). The nudging input received by the interneurons in layer ℓ is

$$i^{\text{nudge, I}} = g^{\text{nudge, I}} \left(\boldsymbol{u}_{\ell+1}^{\text{P}} - \boldsymbol{u}_{\ell}^{\text{I}} \right) \ . \tag{6.8}$$

The input weights to the interneurons $\boldsymbol{W}^{\text{IP}}$ can then be learned analogously to Eqn. (6.7)

$$\dot{\boldsymbol{W}}_{\ell,\ell}^{\text{IP}} = \eta_{\ell}^{\text{IP}} \left[\varphi \left(\boldsymbol{u}_{\ell}^{\text{I}} \right) - \varphi \left(\frac{g^{\text{den}}}{g_{\text{l}} + g^{\text{den}}} \boldsymbol{v}_{\ell}^{\text{den}} \right) \right] \varphi \left(\boldsymbol{u}_{\ell}^{\text{P}} \right) \tag{6.9}$$

with $\frac{g^{\text{den}}}{g_{\text{l}}+g^{\text{den}}} \boldsymbol{v}_{\ell}^{\text{den}} = \boldsymbol{v}_{\ell}^{\text{den},*}$.

The second requirement for the self-predicting state is that all apical voltages v^{api} in the network must be zero. Since they are used to store local error signals during the second stage of learning, they need to be zero if no error/target signal is provided to the network. This goal is reached by adjusting the lateral weights from interneurons to the apical compartments of the pyramidal neurons $\boldsymbol{W}^{\text{PI}}$

$$\dot{\boldsymbol{W}}_{\ell,\ell}^{\text{PI}} = \eta_{\ell}^{\text{PI}} \left[-\boldsymbol{v}_{\ell}^{\text{api}} \right] \varphi \left(\boldsymbol{u}_{\ell}^{\text{I}} \right) \ . \tag{6.10}$$

This learning rule is a slight variant of Eqn. (6.7), as it is no longer based on the difference of rates $\varphi(u) - \varphi(v^*)$ but instead on the difference of the potentials $E_\mathrm{l} - v^\mathrm{api}$ with $E_\mathrm{l} = 0$. When both lateral weights are learned correctly, each interneuron receives the same input as the pyramidal neuron it is supposed to mimic. Therefore, both interneuron and above-layer pyramidal neuron will produce the same output. Additionally, the weights connecting to the apical compartments have been set up in a way so the signals from the interneurons and from the above-layer pyramidal neurons cancel each other out and the apical voltage is at rest. An example of a small network learning the self-predicting state can be found in Fig. 6.3 b, c. Note that for this figure the network dynamics have been modified but the principle of learning the self-predicting state remains the same[3].
It is possible to initialize the network directly in a self-predicting state by setting

$$\boldsymbol{W}^\mathrm{IP}_{\ell,\ell} = \frac{g^\mathrm{bas}\left(g_\mathrm{l} + g^\mathrm{den}\right)}{g^\mathrm{den}\left(g_\mathrm{l} + g^\mathrm{bas} + g^\mathrm{api}\right)} \boldsymbol{W}^\mathrm{PP}_{\ell+1,\ell} \tag{6.11}$$

$$\boldsymbol{W}^\mathrm{PI}_{\ell,\ell} = -\boldsymbol{B}^\mathrm{PP}_{\ell,\ell+1} \tag{6.12}$$

in the lower layers and

$$\boldsymbol{W}^\mathrm{IP}_{N-1,N-1} = \frac{g^\mathrm{bas}\left(g_\mathrm{l} + g^\mathrm{den}\right)}{g^\mathrm{den}\left(g_\mathrm{l} + g^\mathrm{bas}\right)} \boldsymbol{W}^\mathrm{PP}_{N,N-1} \tag{6.13}$$

in the last layer which contains interneurons.
Once the network is in the self-predicting state, the learning of the task can start. To this end pairs of input- and target signals are provided to the network. The pyramidal neurons in the top-layer receive a nudging signal

$$i^\mathrm{nudge,\,tgt} = g^\mathrm{nudge,\,tgt}\left(\boldsymbol{u}^\mathrm{tgt} - \boldsymbol{u}^\mathrm{P}_N\right) \ . \tag{6.14}$$

With this the weight from the last hidden layer to the top-layer pyramidal neurons can be learnt analogously to Eqn. (6.7)

$$\dot{\boldsymbol{W}}^\mathrm{PP}_{N,N-1} = \eta^\mathrm{PP}_N \left[\varphi\left(\boldsymbol{u}^\mathrm{P}_N\right) - \varphi\left(\frac{g^\mathrm{bas}}{g_\mathrm{l} + g^\mathrm{bas}} \boldsymbol{v}^\mathrm{bas}_N\right)\right] \varphi\left(\boldsymbol{u}^\mathrm{P}_{N-1}\right) \tag{6.15}$$

where $\frac{g^\mathrm{bas}}{g_\mathrm{l}+g^\mathrm{bas}} \boldsymbol{v}^\mathrm{bas}_N = \boldsymbol{v}^{\mathrm{bas},*}_N$.
In the hidden layers the nudging of the somatic voltage is not done via a nudging current but via the apical dendrite. If the top-layer pyramidal neurons receive external nudging to drive them closer to the target, the interneurons in the layer below can no longer exactly mimic the pyramidal neurons as they only receive their bottom-up input. Therefore, the

[3] This figure was produced for Haider et al. (2021) to illustrate the learning mechanisms of dendritic microcircuits with neuron dynamics that contain the Latent Equilibrium mechanism.

two signals arriving at the apical dendrites in the last hidden layer no longer perfectly cancel out and what remains serves as local error to nudge these hidden pyramidal neurons.
This local error which nudges the pyramidal neurons in the highest hidden layer is in turn not available to interneurons in the layer below and therefore the signals arriving at the apical dendrites in the layer below also do not cancel and provide a local error to those pyramidal neurons too. And with that, all pyramidal neurons in all layers receive a back-propagated error signal which is stored in the apical dendrite and serves as nudging for learning:

$$\dot{\boldsymbol{W}}^{\text{PP}}_{\ell,\ell-1} = \eta^{\text{PP}}_{\ell}\left[\varphi\left(\boldsymbol{u}^{\text{P}}_{\ell}\right) - \varphi\left(\frac{g^{\text{bas}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}}\boldsymbol{v}^{\text{bas}}_{\ell}\right)\right]\varphi\left(\boldsymbol{u}^{\text{P}}_{\ell-1}\right) \tag{6.16}$$

Note that in the hidden layers $v^{\text{bas},*}_{\ell} = \frac{g^{\text{bas}}}{g_{\text{l}}+g^{\text{bas}}+g^{\text{api}}}\boldsymbol{v}^{\text{bas}}_{\ell}$ includes the apical conductance which is not present in the top-layer.
During learning the self-predicting state has to be maintained in order to keep the error propagation mechanism functional. Therefore, whenever the feedforward weights $\boldsymbol{W}^{\text{PP}}$ change, the lateral weights $\boldsymbol{W}^{\text{IP}}$ need to be adapted accordingly. As there are no phases in the learning process, this has to be done simultaneously to learning the feedforward weights. The update rule is the same as shown for the self-predicting state in Eqn. (6.9). In Section 6.2.2 and Appendix A.2 we show how the learning mechanisms in dendritic microcircuits under certain circumstances approximate the error backpropagation algorithm.

Implementation

We have found that in practice numerical stability of the simulations is improved if we rewrite the differential equations for the neurons to

$$C_{\text{m}}\dot{\boldsymbol{u}} = \frac{1}{\tau_{\text{eff}}}\left(\boldsymbol{u}^{\text{eff}} - \boldsymbol{u}\right)\ , \tag{6.17}$$

$$\tau_{\text{eff}} = \frac{C_{\text{m}}}{g_{\text{l}} + g^{\text{bas/den}} + g^{\text{api/nudge}}}\ . \tag{6.18}$$

$\boldsymbol{u}^{\text{eff}}$ represents the voltage at which the somatic voltage will settle once the steady state is reached. The steady-state voltage is calculated differently for the different neuron types:

$$\boldsymbol{u}^{\text{eff,P}}_{\ell} = \frac{g_{\text{l}}E_{\text{l}} + g^{\text{bas}}\boldsymbol{v}^{\text{bas}}_{\ell} + g^{\text{api}}\boldsymbol{v}^{\text{api}}_{\ell}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}}\ , \tag{6.19}$$

$$\boldsymbol{u}^{\text{eff,I}}_{\ell} = \frac{g_{\text{l}}E_{\text{l}} + g^{\text{den}}\boldsymbol{v}^{\text{den}}_{\ell} + g^{\text{nudge, I}}\boldsymbol{u}^{\text{P}}_{\ell+1}}{g_{\text{l}} + g^{\text{den}} + g^{\text{nudge, I}}}\ , \tag{6.20}$$

$$\boldsymbol{u}^{\text{eff,P}}_{N} = \frac{g_{\text{l}}E_{\text{l}} + g^{\text{bas}}\boldsymbol{v}^{\text{bas}}_{N} + g^{\text{nudge, tgt}}\boldsymbol{u}^{\text{tgt}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{nudge, tgt}}}\ . \tag{6.21}$$

If no target is presented to the top-layer pyramidal neurons their steady state changes to

$$\boldsymbol{u}_N^{\text{eff,P}} = \frac{g_\text{l} E_\text{l} + g^{\text{bas}} \boldsymbol{v}_N^{\text{bas}}}{g_\text{l} + g^{\text{bas}}} \,. \tag{6.22}$$

We have also found in practice that for larger networks and more complicated tasks it is advisable to initialize the network directly in the self-predicting state as shown in Eqn. (6.11) to Eqn. (6.13). We believe that the reason for that is that for larger networks this weight configuration is not the only one which (at least approximately, for the limited number of inputs shown) fulfills the requirements of the self-predicting state. Large networks trying to learn the self-predicting state will typically converge to an approximate weight configuration which is closest to their initialization. This can be problematic for learning, since these alternative solutions are highly dependent on the feedforward weights and on the input samples shown. If the feedforward weights change slightly during the learning of the task, a self-predicting state which was initialized with Eqn. (6.11) can be reestablished with only a slight change in the lateral weights, while the alternative solutions typically require much larger weight changes. This slows down the learning of the task as it requires the learning rates for the feedforward weights to be lowered drastically compared to when Eqn. (6.11) to Eqn. (6.13) is used for network initialization.

6.2.2 Approximation of the error backpropagation algorithm

Under the following assumptions it can be shown that the dendritic microcircuit architecture approximates the error backpropagation algorithm:

- No FA (i.e. random and fixed $\boldsymbol{B}_{\ell-1,\ell}^{\text{PP}}$) but $\boldsymbol{B}_{\ell-1,\ell}^{\text{PP}} = \boldsymbol{W}_{\ell,\ell-1}^{\text{PP, T}}$ instead
- Perfect self-predicting state at all times (see Eqns. (6.11) to (6.13))
- Interneurons match their corresponding above-layer pyramidal neurons perfectly. This requires the self-predicting state and the nudging on the interneurons ($i^{\text{nudge,I}}$ in Eqn. (6.8)) is negligible such that $\boldsymbol{u}_\ell^{\text{I}} = \boldsymbol{v}_{\ell+1}^{\text{bas,*}}$
- Only the steady-state solutions of the somatic membrane voltages (i.e. the value to which the somatic membrane voltage settles for constant input) are regarded.

We will see in Section 6.3 that among these assumptions the steady-state approximation is the most problematic one.
It is important to note that in contrast to ANNs, dendritic microcircuit networks are recursively connected networks. Due to the recursive connectivity the information flowing from top-to-bottom has an impact on the signals travelling along the feedforward pathway in the network. This is different to an ANN where the flow of forward and backward information

are separated into different phases and do not interfere with each other. Therefore, the dendritic microcircuits can only approximate error backpropagation under the condition that the backward information flow has only a weak influence on the feedforward pathway. Under these conditions it can be shown (see Appendix A.2) that there is a layer-wise recursive formulation for the apical voltages

$$\boldsymbol{v}_\ell^{\text{api}} = \lambda \boldsymbol{W}_{\ell+1,\ell}^{\text{PP, T}} \left(\boldsymbol{v}_{\ell+1}^{\text{api}} \odot \varphi' \left(\boldsymbol{v}_{\ell+1}^{\text{bas},*} \right) \right) \tag{6.23}$$

and that the weight update rule for the feedforward weights can be rewritten to

$$\dot{\boldsymbol{W}}_{\ell,\ell-1}^{\text{PP}} = \eta \, \lambda \, \boldsymbol{v}_\ell^{\text{api}} \odot \varphi' \left(\boldsymbol{v}_\ell^{\text{bas},*} \right) \, \varphi \left(\boldsymbol{u}_{\ell-1}^{\text{P}} \right)^{\text{T}} . \tag{6.24}$$

By identifying the error signal as

$$\boldsymbol{e}_\ell = \boldsymbol{v}_\ell^{\text{api}} \odot \varphi' \left(\boldsymbol{v}_\ell^{\text{bas},*} \right) \tag{6.25}$$

we achieve a correspondence to the error backpropagation equations Eqn. (2.26) and Eqn. (2.28). For a more detailed discussion of this correspondence and the treatment of the top-layer see Appendix A.2.2. While the mechanism for error backpropagation in microcircuits is derived for an arbitrary number of hidden layers, in practice it is very difficult to train a larger network with more than one hidden layer. In Appendix A.2.3 we give an explanation on why this is the case.

6.2.3 Neuromorphic implementability and known drawbacks

While the dendritic microcircuit addresses many of the points of biological implausibility in the error backpropagation algorithm, we have identified three main drawbacks of the model that stand in the way of practical implementations in physical systems. These three points will be addressed in the following sections.

- Section 6.3: The choice of leaky-integrator dynamics for the neurons makes them more biologically plausible than their ANN counterparts. However, this comes with the disadvantage of slow information propagation through the network which has a severely negative impact on learning performance and speed (see Section 6.3.2).
- Section 6.4: Dendritic microcircuits employ the FA mechanism to avoid the weight-transport problem. However, the effectiveness of FA is highly task-dependent and is suboptimal both for too small or too deep networks.
- Section 6.5: The model is fundamentally rate-based, which is not amenable to currently available, typically spiking, neuromorphic hardware.

6.3 Latent equilibrium: A unified learning theory for arbitrarily fast computation with arbitrarily slow neurons

6.3.1 Article and author contribution

The article *Latent equilibrium: A unified learning theory for arbitrarily fast computation with arbitrarily slow neurons* was published by Haider et al. in *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*[4]. LK produced the experimental setup and results for Figure 3 and Figure 5 (shown as Fig. 6.4 and Fig. 6.3 here) and was involved in the writing of the corresponding Sections 5 and 14.

6.3.2 Summary

In the following we summarize the main contents of Haider et al. (2021) while focussing in particular on its impact on dendritic microcircuits.

The relaxation problem

The dendritic microcircuit model (Sacramento et al., 2018) as well as a whole host of other models for biologically plausible error backpropagation (Whittington and Bogacz, 2017; Scellier and Bengio, 2017; Guerguiev et al., 2017; Song et al., 2020; Millidge et al., 2020a,b) suffer from what we call the relaxation problem. The relaxation problem is caused by the output of a neuron reacting slowly to a change in the neuron's input. One common example for this are neurons with leaky integrator dynamics:

$$C_\mathrm{m}\dot{u} = -g_\mathrm{l}\,u + I_\mathrm{input} \tag{6.26}$$

where the membrane voltage u is a low-pass filtered version of the input with the membrane time constant $\tau_\mathrm{m} = \frac{C_\mathrm{m}}{g_\mathrm{l}}$. Here, a sudden jump from one value of the input current to a new one will trigger the membrane voltage to decay exponentially to its new value with the time constant τ_m. If we now imagine a hierarchical setup where the lowest layer of leaky integrators receives an input from outside the system and the output of the network (for example a classification result) is observed in the highest layer, we see that a network with n layers will take on the order of $n\,\tau_\mathrm{m}$ to settle to the output value which corresponds to the new input.
This is problematic for two reasons: Firstly, in an inference setting the response time of the network is increased with network depth which might make it unsuitable in situations where fast reaction times to inputs are required. Secondly and more importantly, the delayed network response disrupts the learning process: In a setting where the network is trained with pairs of inputs and corresponding target- or teaching-signals, the difference

[4] Haider et al. (2021)

between the output of the network and the target is driving the learning. But what should ideally drive the learning instead is the difference between the *converged* output and the target. Using the non-converged network output leads to wrong error- or learning signals being passed into the network during the time when the network has not yet reached its settled state.

This effect is illustrated in a simple example with a chain of two neurons in Fig. 6.2. We see that in a system where the output, once settled, matches the target, learning does take place and changes the weights. This is due to the mismatch of output and target before the output has reached its settled state. This mismatch is transported as an error-signal into the network and drives plasticity. The plasticity "over corrects" and pushes the output above the target. Even though, towards the end of the experiment, the output reaches the target again, the synaptic weights have permanently changed, so the previous synaptic weights, which already were able to produce a correct output, were unlearned. This can be problematic, if we imagine that the first set of weights was able to provide a good solution to a given task for all possible inputs. The new weights that the network settled on provide a correct solution for the one specific input that was shown, but can be much for the other inputs, since there was only this one specific input present during their learning.

Previously there have been two common ways of mitigating or weakening this effect: On the one hand phasing of the plasticity can be introduced: Plasticity is switched off until the network has reached its settled state, so all "wrong" error signals from the pre-settling phase have no impact on the weights. This method requires an external and global scheduling mechanism, which, while easily implemented in a software simulation, might be hard to realize in an asynchronous neuromorphic or biological system. The other option is to decrease the learning rate drastically and present each stimulus for a long time. Then the "wrong" learning in the beginning has a low impact and there is enough time once the settled state is reached to undo it and learn with the correct error signals. This has the obvious drawback that each input has to be presented for a long time and therefore the training process of the network is very slow.

Latent equilibrium

Here we introduce a new and superior solution to the relaxation problem. For this we first define a new network state $\breve{\boldsymbol{u}}_\mathrm{m}$ which depends on the network's membrane voltages $\boldsymbol{u}$ and their derivatives $\dot{\boldsymbol{u}}$

$$\breve{\boldsymbol{u}}_\mathrm{m} := \boldsymbol{u} + \tau_\mathrm{m}\dot{\boldsymbol{u}} \ . \tag{6.27}$$

Intuitively, this $\breve{\boldsymbol{u}}_\mathrm{m}$ is a prediction of the future membrane voltages in the network, given the current membrane voltages $\boldsymbol{u}$ and their derivatives. To reflect this we call it the "prospective voltages" of the network.

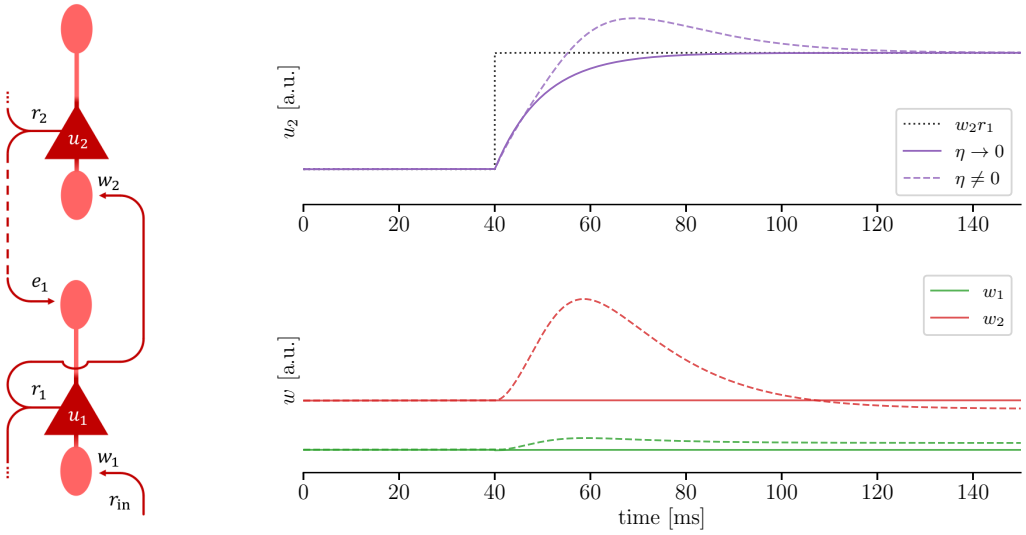

Figure 6.2: Relaxation problem in learning with slow neurons. Figure and caption adapted from (Haider et al., 2021, Fig. 1a,c). Left: A simple, functionally feedforward network of two neurons. Note the recurrence induced by the propagation of both bottom-up signals and top-down errors. Right: Continuous learning during relaxation without prospective coding. Dotted black: target membrane potential. Solid lines: trajectories for vanishing learning rates. Dashed lines: trajectories for nonzero learning rates. Purple: membrane potentials. Note the overshoot that occurs when learning is continuously active. Green/red: presynaptic weights w_1, w_2 of 1st/2nd neuron.

We then define an energy for the network state of prospective voltages

$$E(\breve{\boldsymbol{u}}_\mathrm{m}) := \frac{1}{2} \left\| \breve{\boldsymbol{u}}_\mathrm{m} - \boldsymbol{W}\boldsymbol{\varphi}(\breve{\boldsymbol{u}}_\mathrm{m}) - \boldsymbol{b} \right\|^2 + \beta \mathcal{L}(\breve{\boldsymbol{u}}_\mathrm{m}) \tag{6.28}$$

with the weight matrix $\boldsymbol{W}$, neuronal biases $\boldsymbol{b}$, activation function φ and a loss term $\mathcal{L}$. The first part of the energy is a mismatch energy between "what neurons guess that they will be doing in the future" ($\breve{\boldsymbol{u}}_\mathrm{m}$) and "what their biases and presynaptic afferents believe they should be doing" ($\boldsymbol{b} + \boldsymbol{W}\boldsymbol{\varphi}(\breve{\boldsymbol{u}}_\mathrm{m})$). The second part is a task-dependent loss which provides a measure on how well the neurons perform at the chosen task.

By requiring the energy function to be extremal we can now derive the dynamics of the neurons

$$\nabla_{\breve{\boldsymbol{u}}_\mathrm{m}} E = 0 \quad \Longrightarrow \quad \tau_\mathrm{m} \dot{\boldsymbol{u}} = -\boldsymbol{u} + \boldsymbol{W}\boldsymbol{\varphi}(\breve{\boldsymbol{u}}_\mathrm{m}) + \boldsymbol{b} + \boldsymbol{e} \,, \tag{6.29}$$

with $\boldsymbol{W}\boldsymbol{\varphi}(\breve{\boldsymbol{u}}_\mathrm{m})$ as the bottom-up input and top-down error signals

$$\boldsymbol{e} = \boldsymbol{\varphi}'(\breve{\boldsymbol{u}}_\mathrm{m}) \boldsymbol{W}^T \left[\breve{\boldsymbol{u}}_\mathrm{m} - \boldsymbol{W}\boldsymbol{\varphi}(\breve{\boldsymbol{u}}_\mathrm{m}) - \boldsymbol{b} \right] \,. \tag{6.30}$$

For top-layer neurons which receive direct teaching input the error-signal is $\boldsymbol{e} = -\beta \nabla_{\breve{\boldsymbol{u}}_\mathrm{m}} \mathcal{L}$. In a layered network the error signals can be rewritten to a layer-wise recursive form $\boldsymbol{e}_\ell = \boldsymbol{\varphi}'(\breve{\boldsymbol{u}}_\mathrm{m}) \boldsymbol{W}_{\ell+1}^T \boldsymbol{e}_{\ell+1}$, therefore performing a variant of error backpropagation. For the detailed derivation of these results see (Haider et al., 2021, Supplementary Information, Section 11). The stationarity requirement on the energy function enforces our networks to evolve, for a constant input, on a constant-energy manifold, we therefore call our approach Latent Equilibrium (LE).

From the differential equation for the membrane voltage in Eqn. (6.29) we see that the neurons in our model are leaky integrators. With this and the definition for $\breve{\boldsymbol{u}}_\mathrm{m}$ we can show that the breve operator exactly "un-does" the slow low-pass filtering of the leaky integrator, see (Haider et al., 2021, Supplementary Information, Section 10). Then, the neurons' membrane voltages $\boldsymbol{u}$ react slowly to a jump in the input, while the corresponding prospective voltages $\breve{\boldsymbol{u}}_\mathrm{m}$ immediately jump to the value that $\boldsymbol{u}$ will reach once settled. We now define the output firing rate of the neuron as $\boldsymbol{r} = \varphi\left(\breve{\boldsymbol{u}}_\mathrm{m}\right)$, in contrast to the previous definition $\boldsymbol{r} = \varphi\left(\boldsymbol{u}\right)$. This allows information to travel instantaneously through the network, even though the actual membrane voltages of the neurons have not reached their settled state.

Synaptic plasticity is derived as gradient descent on the energy function:

$$\dot{\boldsymbol{W}} \propto -\nabla_{\boldsymbol{W}} E \quad \Longrightarrow \quad \dot{\boldsymbol{W}} = \eta_W \left[\breve{\boldsymbol{u}}_\mathrm{m} - \boldsymbol{W}\boldsymbol{r} - \boldsymbol{b}\right] \boldsymbol{r}^T \tag{6.31}$$

The resulting weight updates evolve continuously in time and rely only on information and error signals that are locally available. Plasticity in the output layer depends on the chosen loss function $\mathcal{L}$. For the least squares loss on the output rates and a target rate the weight updates are

$$\dot{\boldsymbol{W}}_N = \eta_W \beta \left[\boldsymbol{r}_N^* - \boldsymbol{r}_N\right] \boldsymbol{r}_{N-1}^T \,.$$

The ability to train networks of slow leaky integrator neuron with time-continuous and fast-changing inputs is demonstrated on the MNIST (LeCun et al., 1998) and the more challenging CIFAR10 (Krizhevsky, 2009) datasets. The LE networks achieve test errors of (1.1 ± 0.1) % (MNIST) and (38.0 ± 1.3) % (CIFAR10). For reference a classical ANN with the same structure was trained and achieves (1.08 ± 0.07) % (MNIST) and (39.4 ± 5.6) % (CIFAR10).

Application to dendritic microcircuits

The dendritic microcircuit model (Sacramento et al., 2018) suffers from the relaxation problem and in practice requires such long presentation times for each input sample to make it practically unfeasible[5]. However, the ideas of LE can be applied to the model to alleviate

[5]This can be seen in the original publication, where only the small scale examples are simulated with full dynamics. For the larger simulations on the MNIST dataset a highly simplified steady-state approximation of the model has to be used, to shorten the simulation times.

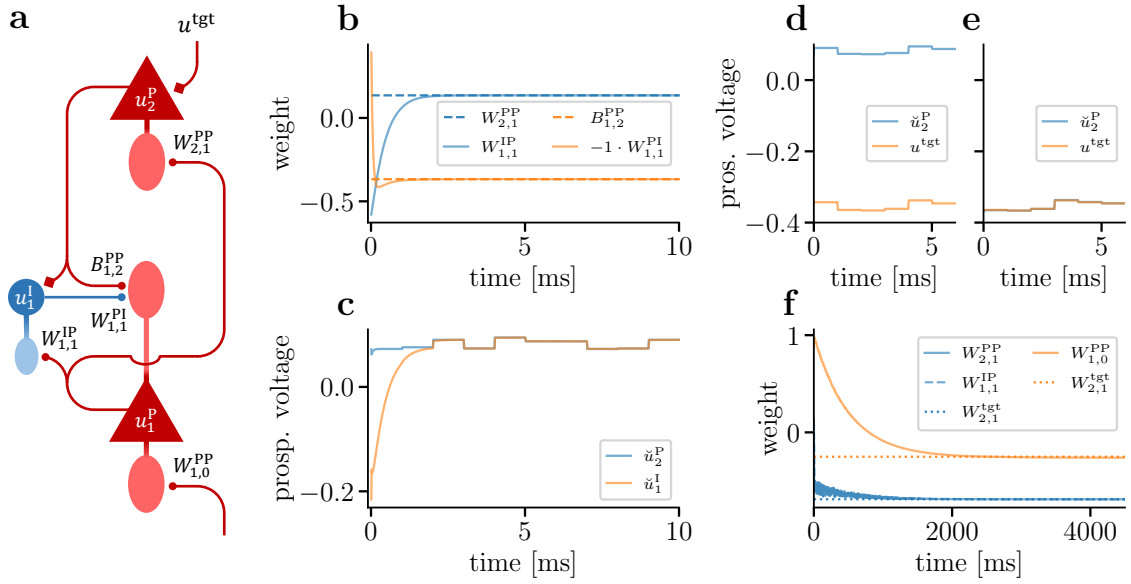

Figure 6.3: Learning to mimic a teacher microcircuit with LE. Figure and caption taken from (Haider et al., 2021, Fig. 5). **(a)** Microcircuit architecture following Sacramento et al. (2018). **(b)** Learning of the lateral weights $W^{\mathrm{IP}}_{1,1}$ and $W^{\mathrm{PI}}_{1,1}$ to implement the self-predicting state. **(c)** Prospective membrane voltages during learning of the self-predicting state where (in absence of a target) the top-down activity is matched by the activity of the interneuron. **(d, e)** Comparison between the prospective membrane voltage $\breve{u}^{\mathrm{P}}_2$ of the output pyramidal neuron and the target voltage u^{tgt} before (d) and after (e) training. **(f)** Weight evolution during learning.

this issue with only two changes: Firstly, the firing rates of all neurons are no longer calculated as $\varphi(\boldsymbol{u})$ but as $\varphi(\breve{\boldsymbol{u}}_{\mathrm{eff}})$. Note that here we use $\breve{u}_{\mathrm{eff}}$ instead of $\breve{u}_{\mathrm{m}}$. This is due to the fact that for the multi-compartment neurons the time constant of the low-pass filtering is not only the time constant of the leak $\tau_{\mathrm{m}} = \frac{C_{\mathrm{m}}}{g_{\mathrm{l}}}$ but the effective time constant $\tau_{\mathrm{eff}} = \frac{C_{\mathrm{m}}}{g_{\mathrm{l}}+g_{\mathrm{compartments}}}$ which combines the influence of leak and compartmental coupling. For notational simplicity we will in the following omit the index indicating the time constant and write $\breve{u}$ instead of $\breve{u}_{\mathrm{eff}}$.

The replacement of $\varphi(\boldsymbol{u})$ with $\varphi(\breve{\boldsymbol{u}})$ occurs at all points where the activation function is applied to a somatic voltage: the calculation of the dendritic membrane potentials (Eqn. (6.4)), the calculation of neuronal output firing rates as well as the weight updates (Eqn. (6.60)). And finally, the interneurons must also be nudged with the prospective voltage of the pyramidal neuron in the layer above

$$\boldsymbol{u}^{\mathrm{eff,I}}_{\ell} = \frac{g_{\mathrm{l}}E_{\mathrm{l}} + g^{\mathrm{den}}\boldsymbol{v}^{\mathrm{den}}_{\ell} + g^{\mathrm{nudge,\,I}}\breve{\boldsymbol{u}}^{\mathrm{P}}_{\ell+1}}{g_{\mathrm{l}} + g^{\mathrm{den}} + g^{\mathrm{nudge,\,I}}} \,. \tag{6.32}$$

Fig. 6.3 illustrates on a simple teacher-student-mimic task that the addition of the LE mechanism preserves the basic microcircuit functionality, while at the same time adding near

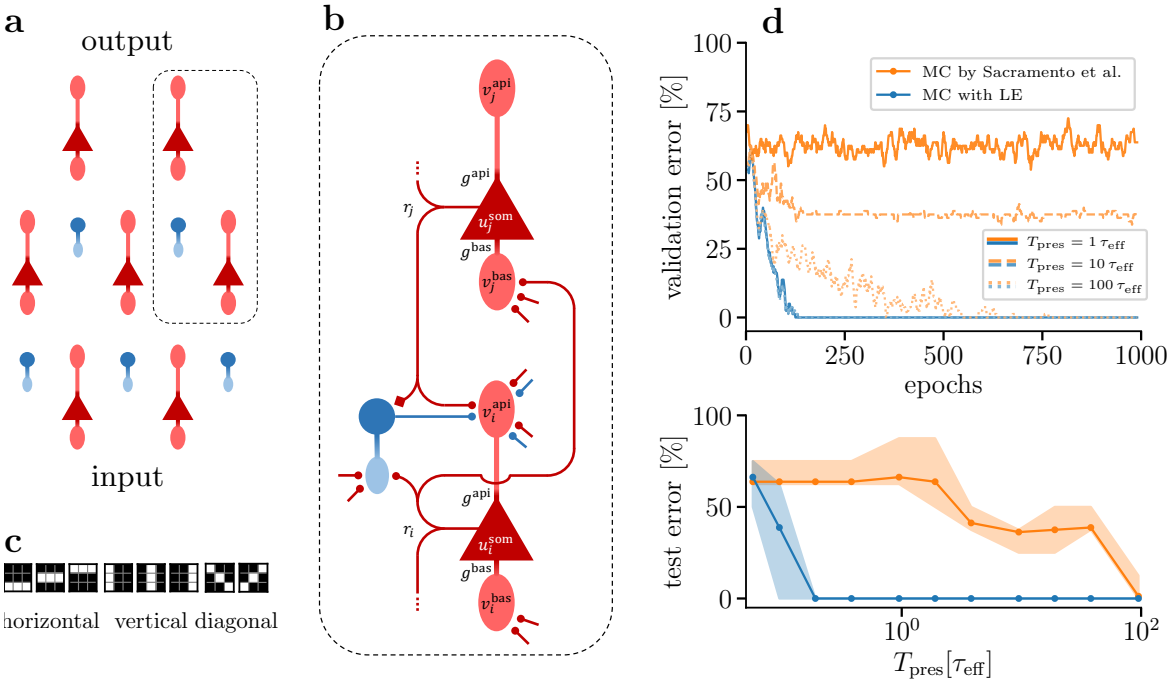

Figure 6.4: Dendritic microcircuits with LE mechanism. Figure and caption taken from (Haider et al., 2021, Fig. 3). **(a,b)** Model architecture following Sacramento et al. (2018). Red: pyramidal cells, blue: interneurons; somatic compartments have darker colors. Each soma sends out a single axon that transmits either $\phi(u)$ (Sacramento et al.) or $\phi(\breve{u}_{\mathrm{m}})$ (LE). Except for top-down synapses, all synapses are plastic. **(c)** Synthetic dataset with 8 images grouped in 3 classes used to train a 3-layer network with 9-30-3 pyramidal cells. **(d)** Model performance during (top) and after (bottom) learning with (blue) and without (orange) LE. Top: Note the similarity in performance gains at the beginning of training, before the disruptive effects of relaxation begin to dominate. For better visualization, fluctuations are smoothed with a sliding window over 10 epochs. Bottom: Model performance (min, max and mean over 10 seeds) after 1000 epochs for different input presentation times.

instantaneous information propagation in the network. First, as a preparation for the learning of the actual task, the self-predicting state is learned by the lateral weights from a fully random weight initialization (Fig. 6.3 b, c). At this stage random input but no targets are provided to the network. Once the self-predicting state is reached, the network is taught to reproduce an input-output relationship which is produced by a teacher network of the same size but with different synaptic weights (Fig. 6.3 d – f)[6]. Note how, even though neurons in this microcircuit are simulated with an effective time constant $\tau_{\mathrm{eff}} = 5\,\mathrm{ms}$, the inputs into the networks change every 1 ms and the prospective voltages of the top-layer and interneuron instantaneously react to those changes (Fig. 6.3 c – e). The slow change in the

[6]We use a teacher network to produce the input-output pairs in order to guarantee that the input-output relationship is actually realizable by a network of the chosen architecture.

interneuron voltage $\breve{u}_1^I$ in Fig. 6.3 c is due to its slowly adapting $\boldsymbol{W}^{\text{IP}}$ and not due to the neuron dynamics.

To illustrate the effect of the inclusion of LE in the microcircuit model on its learning capabilities we compare the adapted model and the original as described in Sacramento et al. (2018) on a small classification task (Fig. 6.4). We see that the microcircuit model with LE is able to fully learn the task even for presentation times of each input sample that are far below its effective time constant τ_{eff}. In contrast, the original model is unable to learn the task if each sample is not at least presented for $100\,\tau_{\text{eff}}$. This shows that the inclusion of the LE mechanism into the dendritic microcircuit model is absolutely crucial to bring the required presentation times (and by that also the effective training durations) into a biologically plausible and practically feasible range.

6.4 Learning efficient backprojections across cortical hierarchies in real time

This chapter contains the article *Learning efficient backprojections across cortical hierarchies in real time*. It is available as a preprint on *arXiv*[7]. The format was adapted to the format of this thesis and the references have been included in the main bibliography.

Author contributions

KM derived, with contributions by LK and MAP, the phaseless alignment learning (PAL) algorithm. KM and LK adapted the dendritic microcircuit model to include PAL for learning the feedback weights. GG and TN developed a dendritic microcircuit module for the GeNN simulator. LK added the latent equilibrium and PAL mechanisms to the module. KM and LK performed the simulation experiments (KM: Figs. 6.5 to 6.7, 6.9 and 6.10; LK: Fig. 6.8). The manuscript was mainly written by KM, aided by LK and MAP.

[7] Max et al. (2022)

Learning efficient backprojections across cortical hierarchies in real time

K. Max[1], L. Kriener[1], G. Garcia[2], T. Nowotny[2], W. Senn[1], M. A. Petrovici[1]

[1] Department of Physiology, University of Bern, 3012 Bern, Switzerland.
[2] School of Engineering and Informatics, University of Sussex, BN1 9RH Brighton, United Kingdom.

Abstract

Models of sensory processing and learning in the cortex need to efficiently assign credit to synapses in all areas. In deep learning, a known solution is error backpropagation, which however requires biologically implausible weight transport from feed-forward to feedback paths.
We introduce Phaseless Alignment Learning (PAL), a bio-plausible method to learn efficient feedback weights in layered cortical hierarchies. This is achieved by exploiting the noise naturally found in biophysical systems as an additional carrier of information. In our dynamical system, all weights are learned simultaneously with always-on plasticity and using only information locally available to the synapses. Our method is completely phase-free (no forward and backward passes or phased learning) and allows for efficient error propagation across multi-layer cortical hierarchies, while maintaining biologically plausible signal transport and learning.
Our method is applicable to a wide class of models and improves on previously known biologically plausible ways of credit assignment: compared to random synaptic feedback, it can solve complex tasks with less neurons and learn more useful latent representations. We demonstrate this on various classification tasks using a cortical microcircuit model with prospective coding.

6.4.1 Introduction

Deep learning has originally been inspired by neuroscience, being influenced by the description of the visual cortex in particular. Nonetheless, these two fields remain at vastly different levels in the description of their respective subjects. While deep learning has made great leaps in terms of applicability and real-world usage in the past decade, the study of biological neural systems has revealed a plethora of different brain areas, connection types, cell types, and neuron as well as system states. Currently, no clear organization scheme of computations and information transfer in the brain is known, and the question of how ANNs are related to models of the cortex remains an active field of research (Yamins and DiCarlo, 2016; Richards et al., 2019; Lillicrap et al., 2020).
However, progress is being made in bridging the gap between these two fields (Roelfsema and Ooyen; Costa et al., 2017; Scellier and Bengio, 2017; Whittington and Bogacz, 2017; Sacramento et al., 2018; Haider et al., 2021; Lillicrap et al., 2016; Yamins and DiCarlo, 2016;

Payeur et al., 2021). For recent review articles, see Marblestone et al. (2016) and Richards et al. (2019). In particular, important similarities between cortical and artificial information processing have been highlighted: as in the cortex (Haak and Beckmann, 2018), most ANN architectures process information hierarchically. Additionally, external stimuli generate activity in functional units (neurons), which utilize bottom-up and top-down information (Zmarz and Keller, 2016; Jordan and Keller, 2020; Cerliani et al., 2022). Neural activity is modulated through learning, i.e. long-term adaptation of synaptic weights. However, it is currently unclear how weights are adapted across the cortex in order to competently solve a task. This is commonly referred to as the credit assignment problem, where neuroscience may learn from deep learning (Friedrich et al., 2011; Richards et al., 2019).

In the case of ANNs, an efficient solution to this problem is known: currently, error backpropagation (BP) (Rumelhart et al., 1986; LeCun et al., 1988) is the gold standard for learning in artificial networks. However, BP has several biologically implausible requirements. ANNs trained with BP operate in distinct forward and backward phases, where inference and learning alternate. Between phases, network activities need to be buffered — i.e., information is processed non-locally in time. Furthermore, error propagation occurs through weights which need to be mirrored at synapses in different layers (weight transport problem).

In order to explain credit assignment for analog, physical computing (in the cortex or on neuromorphic hardware), physically plausible architectures and algorithms are therefore needed. We assume such dynamical systems to operate in continuous time. They may minimize the difference between network output and target ("cost") by performing (approximate) gradient descent. Ideally, such physical systems are able to learn from useful instructive signals at all times, using only information which is locally available in space and time. Crucially, all physical systems have inherent sources of noise — in the form of stochastic activity, noisy parameters or intrinsic fluctuations of electrical and chemical signals. The theory we propose makes use of neuronal noise as an additional carrier of information, instead of treating it as a nuisance parameter.

For efficient credit assignment as in ANNs, errors need to be propagated from higher to lower areas in the cortical hierarchy. With vanilla BP being excluded due to biological implausibility of weight transport, the question of how such error propagation occurs remains open. Several methods have been proposed where feedback connections are assumed to be fixed or are learned. Broadly, these can be categorized into methods with fixed feedback connections (feedback alignment, FA Lillicrap et al. (2016); Nøkland (2016)), bio-plausible approximations to BP (Kolen and Pollack, 1994; Akrout et al., 2019; Lansdell et al., 2019; Ernoult et al., 2022), or alternative cost minimization schemes (Bengio, 2014; Lee et al., 2015; Meulemans et al., 2020, 2021). In this work, we introduce a method in the second category, related to the top-down weight alignment method of Ernoult et al. (2022), which itself is based on previous insights on cost minimization in difference target propagation (Meule-

mans et al., 2020). The aim of our algorithm is to propagate BP-like error, and to perform gradient descent on a cost function.

The novelty of this work is that we propose a fully dynamical system with efficient always-on plasticity: the neuronal and weight dynamics model properties of physical substrates, while learning is completely phase-less, and plasticity is enabled for all synapses and at all times. In agreement with biological plausibility, our method allows for efficient learning without requiring wake-sleep phases or other forms of phased plasticity implemented in many other models of learning in the cortex (O'Reilly, 1996; Ackley et al., 1987; Bengio and Fischer, 2015; Sacramento et al., 2018; Guerguiev et al., 2017; Scellier and Bengio, 2017; Mesnard et al., 2019; Xie and Seung, 2003; Song et al., 2022).

Our method is based on modeling of biologically plausible signal transport in the form of rate-coding. The learning mechanism incorporates bio- and hardware-plausible components and computations. All dynamics and plasticity rules are fully local in time and space. Our model also makes full use of a recently proposed prospective coding mechanism (Haider et al., 2021). This ensures fast propagation of information through layered networks, leading to quick convergence of useful top-down projections when using our method.

6.4.2 Results

Learning of efficient backprojections

We describe our theory in a rate-based coding scheme. Following the convention defined for artificial neural networks, different cortical areas are represented by layers. The somatic potentials of all neurons follow the dynamics of leaky integrators: given an input current $\boldsymbol{I}[t]$, the membrane potential $\boldsymbol{u}$ obeys $C_m \dot{\boldsymbol{u}} = -g_\mathrm{l}\,\boldsymbol{u} + \boldsymbol{I}[t]$, where C_m denotes the capacitance, and g_l the leak conductance of the cell membrane. These dynamics imply a delayed response of the somatic potential with membrane time constant $\tau_\mathrm{eff} \coloneqq C_m/g_\mathrm{l}$.

Our model integrates the prospecitve coding mechanism of Latent Equilibrium (Haider et al., 2021), which solves the relaxation problem of slow physical substrates, which disrupts inference as well as learning (see Methods). This is achieved by calculating the neural output from the prospective voltage $\breve{\boldsymbol{u}} \coloneqq \boldsymbol{u} + \tau_\mathrm{eff}\,\frac{d\boldsymbol{u}}{dt}$ where τ_eff is the effective membrane time constant. As a result, the rates calculated from $\breve{\boldsymbol{u}}$ follow the input current $\boldsymbol{I}[t]$ instantaneously.

Our theory describes neural dynamics where the current $\boldsymbol{I}[t]$ contains a local error signal. For concreteness, but without loss of generality, we consider the leaky-integrator model for a layered architecture with $\ell = 1 \ldots N$,

$$\tau_\mathrm{eff}\dot{\boldsymbol{u}}_\ell = -\boldsymbol{u}_\ell + \boldsymbol{b}_\ell + \boldsymbol{W}_{\ell,\ell-1}\boldsymbol{r}_{\ell-1} + \boldsymbol{e}_\ell + \boldsymbol{\xi}_\ell\,. \tag{6.33}$$

In this description, the somatic potential $\boldsymbol{u}_\ell$ integrates neuron bias $\boldsymbol{b}_\ell$, the bottom-up input rate $\boldsymbol{r}_{\ell-1}$ weighted with $\boldsymbol{W}_{\ell,\ell-1}$, and the local error $\boldsymbol{e}_\ell$. We also model a noise component $\boldsymbol{\xi}_\ell$. The somatic potential of each layer generates a rate, which we denote as $\boldsymbol{r}_\ell \coloneqq \varphi(\breve{\boldsymbol{u}}_\ell)$.
The central question of cortical credit assignment is how the error $\boldsymbol{e}_\ell$ is calculated, given an error signal in a higher area, $\boldsymbol{e}_{\ell+1}$, and how this error signal is used to adjust bottom-up weights. Plenty of solutions to this question have been proposed (Marblestone et al., 2016). Here, we focus on theories which can be formulated such that forward weights are updated as $\dot{\boldsymbol{W}}_{\ell,\ell-1} \propto \boldsymbol{e}_\ell \boldsymbol{r}_{\ell-1}^T$. Under this scheme, several theories for bio-plausible error transport exist (Xie and Seung, 2003; Scellier and Bengio, 2017; Haider et al., 2021; Lee et al., 2015; Sacramento et al., 2018; Meulemans et al., 2020), among them contrastive Hebbian learning or difference target propagation. They have in common that errors in a higher layer $\boldsymbol{e}_{\ell+1}$ are propagated down through feedback (top-down) weights $\boldsymbol{B}_{\ell,\ell+1}$ to form errors in a given layer $\boldsymbol{e}_\ell$.
Typically, a symmetric overall weight matrix is assumed, such that $\boldsymbol{B}_{\ell,\ell+1} = [\boldsymbol{W}_{\ell+1,\ell}]^T$ (Xie and Seung, 2003; Scellier and Bengio, 2017; Sacramento et al., 2018; Podlaski and Machens, 2020).[8] This assumption relates the above schemes to classical error backpropagation based on gradient descent on a loss function, where $\boldsymbol{e}_\ell = \varphi' \cdot [\boldsymbol{W}_{\ell+1,\ell}]^T \boldsymbol{e}_{\ell+1}$. However, this assignment of weights implies that top-down synapses in layer ℓ must adapt to the (potentially distant) bottom-up synapses in layer $\ell+1$. This issue of how two spatially distant synapses (e.g. across cortical areas) can keep up a similar weight when one of them is learning is known as the weight transport problem.
A proposed solution is the replacement of $\boldsymbol{B}_{\ell,\ell+1}$ with a random, fixed weight matrix (known as feedback alignment, FA (Lillicrap et al., 2016)). However, FA has been shown to solve credit assignment inefficiently, and does not scale well to complex problems (Nøkland, 2016; Moskovitz et al., 2018; Bartunov et al., 2018; Lansdell et al., 2019). We are therefore motivated to learn top-down weights such that credit assignment is improved compared to random feedback with layer-wise connections. Furthermore, we would like to learn all weights $\boldsymbol{B}_{\ell,\ell+1}$ simultaneously, and in a way which does not interrupt feed-forward inference or learning of bottom-up weights.
The general method we propose is explained in the following (see Fig. 6.5). An input signal $\boldsymbol{r}_0(t)$ ("data") is presented to the neurons in the lowest layer for the duration of a presentation time T_{pres}. This signal is propagated forward from layer 1 to N, where each neuron follows the dynamics of Eqn. (6.33). To reflect the inherent stochasticity of biological neurons subject to synaptic noise, thermal activity and probabilistic firing, high-frequency noise is modeled at every neuron. This noise is accumulated across layers, and propagated on top of the data signal. The top-down projections carry this mixed signal back to the lower layers, and we exploit the auto-correlation between noise signals to learn the corresponding feedback synapses based on a local alignment loss.

[8] Some learning schemes apply $\dot{\boldsymbol{B}}_{\ell,\ell+1} = [\dot{\boldsymbol{W}}_{\ell+1,\ell}]^T$ (Roelfsema and Ooyen; Pozzi et al., 2018), leading to a symmetric overall weight matrix. It is unclear however how weight updates are communicated.

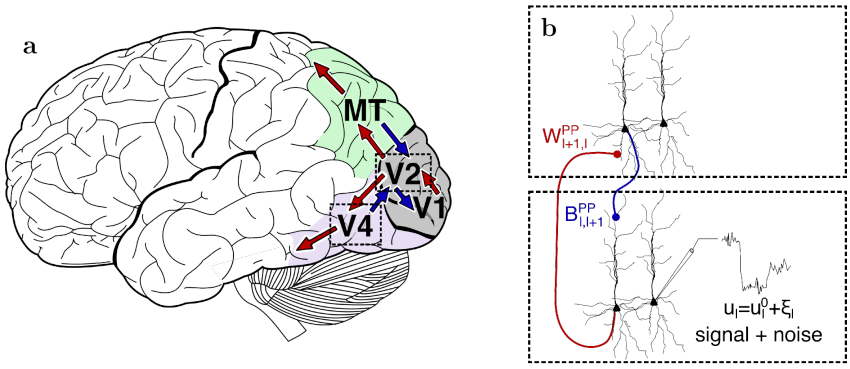

Figure 6.5: Sensory processing over cortical hierarchies. a: Brain areas in the visual pathway beyond the primary visual cortex (V1). Information is propagated to higher areas (red arrows) such as V2, V4, the medial temporal (MT) area, and beyond. In order to assign credit, feedback information from higher level areas needs to be propagated top-down (blue arrows). Adapted from Gray (1918). **b:** Pyramidal cells as functional units of sensory processing and credit assignment. Top-down and bottom-up projections preferentially target different dendrites. Due to stochastic dynamics of individual neurons, noise is added to the signal.

Concretely, for each hidden layer we sample Ornstein-Uhlenbeck noise $\boldsymbol{\xi}_\ell$ with zero mean and a small amplitude compared to the somatic potential $\boldsymbol{u}_\ell$ (see Methods, Eqn. (6.40)). This generic noise term is added as a current to the soma of each hidden layer neuron, where it adds to the data signal to form a noisy firing rate. therefore, the sampled noise changes faster than the data signal (i.e. stimulus). This condition, $T_{\text{pres}} \gg \tau_\xi$, ensures that data and noise are separable in frequencies.

To learn the backwards weights, simple and only local computations need to be performed by the backprojections. At every top-down synapse $\boldsymbol{B}_{\ell,\ell+1}$, a high-pass filtered rate $\widehat{\boldsymbol{r}}_{\ell+1}$ is computed, which extracts the noise signal; this can be implemented in a physical substrate as the difference of the top-down rate with a low-pass filtered version. This filter separates the noise from the data portion of the top-down rate. We learn the feedback synapses through minimization of a layer-wise alignment loss defined as

$$\mathcal{L}^{\text{PAL}}_\ell = -\boldsymbol{\xi}^T_\ell(t)[\boldsymbol{B}_{\ell,\ell+1}\widehat{\boldsymbol{r}}_{\ell+1}(t)] + \frac{\alpha}{2}\left\|\boldsymbol{B}_{\ell,\ell+1}\right\|^2 , \tag{6.34}$$

where α is a constant defining the size of the regularizer. Minimization of $\mathcal{L}^{\text{PAL}}_\ell$ leads to approximate alignment of $\boldsymbol{B}_{\ell,\ell+1}$ with $\boldsymbol{W}_{\ell+1,\ell}$, as detailed below. Performing gradient descent on the alignment loss defines the top-down weight updates,

$$\dot{\boldsymbol{B}}_{\ell,\ell+1} := -\eta^{\text{bw}}_\ell \nabla_{\boldsymbol{B}_{\ell,\ell+1}} \mathcal{L}^{\text{PAL}}_\ell = \eta^{\text{bw}}_\ell \left[\boldsymbol{\xi}_\ell \left(\widehat{\boldsymbol{r}}_{\ell+1}\right)^T - \alpha\, \boldsymbol{B}_{\ell,\ell+1}\right] , \tag{6.35}$$

which can be performed *simultaneously* for all layers to learn all backprojections $\boldsymbol{B}_{\ell,\ell+1}$, while allowing the learning of forward weights $\boldsymbol{W}_{\ell+1,\ell}$ at the same time. Due to this crucial

property, we name the above method *phaseless alignment learning (PAL)*. Note also that the learning rule is constructed solely from information which is available pre- and post-synaptically for each neuron at each point in time. This is in line with our requirement of physical information processing, as well as phenomenological models of plasticity (Clopath et al., 2010; Bono and Clopath, 2017).

Useful alignment of $\boldsymbol{B}_{\ell,\ell+1}$ through minimization of $\mathcal{L}_\ell^{\mathrm{PAL}}$ occurs in the following way (see also Methods). At a given top-down synapse, the rate $\boldsymbol{r}_{\ell+1}$ arrives from the layer above. Note that this rate is made from data as well as noise accumulated from all layers; among this is also the noise originating in layer ℓ. The top-down synapse now calculates the high-pass filtered rate $\widehat{\boldsymbol{r}}_{\ell+1}$, discarding the data portion of the incoming signal. As has been pointed out in Meulemans et al. (2021), we then can exploit that the autocovariance of Ornstein-Uhlenbeck noise decays exponentially in time (Särkkä and Solin, 2019). Therefore, the only non-zero correlation between all noise signals contained in $\widehat{\boldsymbol{r}}_{\ell+1}$ and the current, local noise sample $\boldsymbol{\xi}_\ell$ is proportional to the expectation value of the local noise auto-covariance $\boldsymbol{\xi}_\ell(t+\Delta t)\,\boldsymbol{\xi}_\ell(t)^T$. Here Δt denotes the time it takes for a noise sample to travel in a loop containing the layer above.

Minimization of $\mathcal{L}_\ell^{\mathrm{PAL}}$ for a given input sample and fixed bottom-up weights aligns the backwards weights as

$$\boldsymbol{B}_{\ell,\ell+1} \propto \varphi'(\breve{\boldsymbol{u}}_\ell^0)\left[\boldsymbol{W}_{\ell+1,\ell}\right]^T \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0)\,, \tag{6.36}$$

where we refer to Methods for the derivation. More generally, in a fully dynamical system with changing input, $\boldsymbol{B}_{\ell,\ell+1}$ will converge to a weight which also aligns approximately with $[\boldsymbol{W}_{\ell+1,\ell}]^T$, but is a mean over input data, i.e. $\boldsymbol{B}_{\ell,\ell+1} \propto \mathbb{E}[\varphi'\,[\boldsymbol{W}_{\ell+1,\ell}]^T\,\varphi']_{\boldsymbol{r}_0}$.

Note that our mechanism is able to take full advantage of the property of arbitrarily fast propagation due to Latent Equilibrium. Noisy rates are calculated from the prospective voltage, and therefore the time delay between the top-down noise signal and the post-synaptic noise sample can become arbitrarily small. This means that the correlation time scale of the Ornstein-Uhlenbeck noise τ_ξ can also be small,[9] leading to fast convergence of backprojections; in comparison, methods without prospective coding require $\tau_\xi \gg \tau_{\mathrm{eff}}$, such that top-down weights converge slowly (e.g. Meulemans et al. (2021)).

Learning backward weights in our framework is not disturbed by simultaneous learning of forward weights due to the frequency separation of data and noise: as we require $\tau_\xi \ll T_{\mathrm{pres}}$, the error signal for forward weights can be recovered from backprojections by a low-pass filter with time constant larger than τ_ξ. Furthermore, phaseless alignment learning (PAL) is also able to learn useful backprojections in absence of a teaching signal, facilitating efficient learning once an instructive signal is (re-)introduced. In particular, top-down weights do not decay to zero if forward weights are kept fixed, even though the weight decay term $\dot{\boldsymbol{B}}_{\ell,\ell+1} \propto -\alpha\,\boldsymbol{B}_{\ell,\ell+1}$ might suggest so. The reason for this is that the expectation value of

[9] The exact requirement on the time scales is $\tau_\xi \gtrsim \Delta t$.

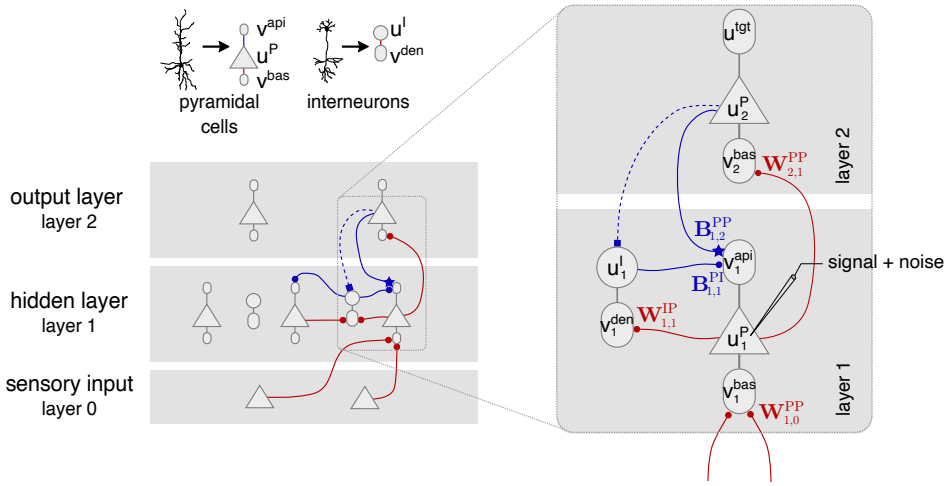

Figure 6.6: Cortical microcircuit setup with one hidden layer. Left: Full network with pyramidal cells and interneurons. Triangles represent somata of pyramidal neurons, with attached basal and apical compartments. Interneuron somata (circle) receive input from a single dendritic compartment and a nudging signal from a matching pyramidal cell in the layer above. **Right:** Single microcircuit. Somatic voltages contain bottom-up data signal, top-down error, and noise. The top-down synapses adapted with PAL are marked with a star.

top-down weights, Eqn. (6.36), are formed from a balance between noise and the contribution due to the regularizer; see simulation results, Fig. 6.8, and Methods for details.

Cortical microcircuit implementation

We now consider a particular implementation of PAL in the framework of dendritic cortical microcircuits (Sacramento et al., 2018). This model has been introduced with biological plausible (error) signal transport in mind. Each microcircuit is defined by populations of two types of neurons, pyramidal cells and interneurons. These are organized in layers corresponding to cortical areas with a biologically plausible connection scheme, see Fig. 6.6.

In absence of a teaching signal, pyramidal cells take the role of representation units, reflecting feed-forward activation. They receive bottom-up information onto their basal dendrites and top-down activity in the distal apical dendrite, integrating both signals in the soma in accordance with observations of layer 2/3 pyramidal cells (Jordan and Keller, 2020). Pyramidal cells are modeled with a simplified three-compartment model with distinct basal, apical and somatic voltages (Körding and König, 2001; Spruston, 2008).

The interneurons in this model are present in the hidden layers, and aim to represent a copy of the activation of pyramidal cells in the layer above. Across layers, populations of pyramidal neurons and interneurons are arranged such that the number of interneurons in the hidden layers matches that of pyramidal cells in the layer above. Interneurons are modeled

with two compartments, representing dendritic tree and soma. They receive lateral input from pyramidal cells in the same layer, and project back laterally to the same neurons.

The dynamics of the somatic membrane potentials of pyramidal cells $\boldsymbol{u}_\ell^{\mathrm{P}}$ with $\ell = 1, \ldots, N-1$ are an instance of the general leaky-integrator equation Eqn. (6.33),

$$C_{\mathrm{m}}\dot{\boldsymbol{u}}_\ell^{\mathrm{P}} = g_{\mathrm{l}}\left(\boldsymbol{E}_{\mathrm{l}} - \boldsymbol{u}_\ell^{\mathrm{P}}\right) + g^{\mathrm{bas}}\left(\boldsymbol{v}_\ell^{\mathrm{bas}} - \boldsymbol{u}_\ell^{\mathrm{P}}\right) + g^{\mathrm{api}}\left(\boldsymbol{v}_\ell^{\mathrm{api}} + \boldsymbol{\xi}_\ell(t) - \boldsymbol{u}_\ell^{\mathrm{P}}\right) , \tag{6.37}$$

where Ornstein-Uhlenbeck noise $\boldsymbol{\xi}_\ell$ is modeled at all hidden layers, and $\boldsymbol{E}_{\mathrm{l}}$ denotes the leak potential. A target signal can be introduced by clamping the apical compartment of the top layer pyramidal neurons to the target voltage $\boldsymbol{u}^{\mathrm{tgt}}$. Somatic voltages are determined by leaky integration of input basal and apical currents. Dendritic compartment voltages are calculated instantaneously from their rate input, through $\boldsymbol{v}_\ell^{\mathrm{bas}} = \boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}\boldsymbol{r}_{\ell-1}^{\mathrm{P}}$ for basal (bottom-up) input, $\boldsymbol{v}_1^{\mathrm{den}} = \boldsymbol{W}_{1,1}^{\mathrm{IP}}\,\boldsymbol{r}_1^{\mathrm{P}}$ for the lateral input from pyramidal cells to interneurons, and the apical compartment potential determined from the sum of top-down and lateral activity, $\boldsymbol{v}_1^{\mathrm{api}} = \boldsymbol{B}_{1,2}^{\mathrm{PP}}\boldsymbol{r}_2^{\mathrm{P}} + \boldsymbol{B}_{1,1}^{\mathrm{PI}}\,\boldsymbol{r}_1^{\mathrm{I}}$. We refer to Methods for details.

As shown in Sacramento et al. (2018), the weight updates in this model approximate those of error backpropagation in the limit of weak nudging (small top-down conductances):

$$\Delta\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}} \propto \lambda^{N-\ell+1}\,\varphi'(\hat{\boldsymbol{v}}_\ell^{\mathrm{bas}})\left[\prod_{k=\ell}^{N-1}\boldsymbol{B}_{k,k+1}^{\mathrm{PP}}\,\varphi'(\hat{\boldsymbol{v}}_{k+1}^{\mathrm{bas}})\right]\boldsymbol{e}_N\left(\boldsymbol{r}_{\ell-1}^{\mathrm{P}}\right)^T , \tag{6.38}$$

where $\hat{\boldsymbol{v}}_\ell^{\mathrm{bas}} := \frac{g^{\mathrm{bas}}}{g_{\mathrm{l}}+g^{\mathrm{bas}}+g^{\mathrm{api}}}\boldsymbol{v}_\ell^{\mathrm{bas}}$ denotes the conductance-weighted feed-forward input to each pyramidal cell, $\boldsymbol{e}_N := \boldsymbol{u}^{\mathrm{tgt}} - \hat{\boldsymbol{v}}_N^{\mathrm{bas}}$ the output layer error, and a small parameter λ, which regulates the amount of top-down nudging.

In contrast to the model defined by Sacramento et al. (2018), which employs fixed feedback connections, we learn the backward connections using PAL with the scheme defined in Section 6.4.2. We consider noise in the hidden layers with a small amplitude compared to the corresponding somatic potentials. Synaptic plasticity of thus-far fixed weights $\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}$ is enabled through the learning rule Eqn. (6.35). Finally, in order to preserve learning of feed-forward weights, we endow the update rule of $\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}$ with a low-pass filter with time constant τ_{lo}. Note that all computations required for PAL can be performed locally by the corresponding synapse.

Additionally, we implement Latent Equilibrium into the microcircuit model as in Haider et al. (2021) by replacing all rates calculated from somatic potentials with rates obtained from the prospective voltage, $\boldsymbol{r}_\ell^{\mathrm{P/I}} = \varphi(\boldsymbol{u}_\ell^{\mathrm{P/I}}) \mapsto \varphi(\breve{\boldsymbol{u}}_\ell^{\mathrm{P/I}})$. This affects all compartment potentials as well as synaptic plasticity rules.

Experiments

We perform several experiments in order to evaluate PAL. The base algorithm given by Eqn. (6.35) is applicable to rate-based neuron models. Here, we focus our experiments on a microcircuit implementation as defined in the previous section, in order to demonstrate its merits as a bio-plausible method for learning. We show that PAL is able to align top-down weights to useful backprojections, and compare weight updates to those of an ANN trained with BP. A simple toy task (teacher-student) illustrates where PAL improves on using fixed random backprojections. Using computer vision benchmark tests, we demonstrate that PAL is able to scale to bigger networks and more complex tasks. Finally, we show that PAL facilitates credit assignment in deep networks, where multiple hidden layers are required for successful learning.
We stress that all simulations are performed with fully recurrent dynamics described by Eqns. (6.33) and (6.37), differentiating our work from similar studies where the dynamics are replaced by steady-state approximations and the recurrency is implicitly removed by calculating separate forward and backward passes (Sacramento et al., 2018; Greedy et al., 2022). All simulation parameters are given in Supplementary Information.

Phaseless backwards weight alignment We first demonstrate that PAL aligns top-down weights in cortical microcircuits with the theoretical result given by Eqn. (6.36). We simulate the dendritic microcircuit model with three hidden layers, keeping the forward weights $\boldsymbol{W}^{\text{PP}}_{\ell,\ell-1}$ fixed while modeling noise in all hidden layers and learning all $\boldsymbol{B}^{\text{PP}}_{\ell,\ell+1}$ simultaneously. In this experiment, we present no target to the output layer. Top-down weights as well as lateral weights from interneurons to pyramidal cells are adapted fully dynamically during training.
Results are shown in Fig. 6.7, where the upper and lower row correspond to the two cases where neurons are active in their linear/non-linear regime. It is of interest to evaluate both of these regimes, as complex tasks cannot be solved with a fully linear network. Nevertheless, it is also not the case that all neurons are in the non-linear regime for all inputs; typically, there is a mixture of both states present in the network.
The first column shows the angle between top-down weights and Eqn. (6.36), demonstrating good agreement with the theoretical expectation over all hidden layers.[10] These backward weight configurations are useful, as they approximately align with the transpose of the forward weights; we show the corresponding alignment angle in the second column. Alignment of top-down weights with the transpose is much better in the linear regime, as $\varphi' = \mathbf{1}$.
In the third column, we show that errors propagated in a microcircuit with PAL approximately align with backpropagation. After each epoch of training the backprojections, we evaluate the model in its current state by introducing a teaching signal. This generates an

[10] We hypothesize that the larger misalignment angle in the non-linear case is due to the data-specific learning of backwards weights (see Discussion).

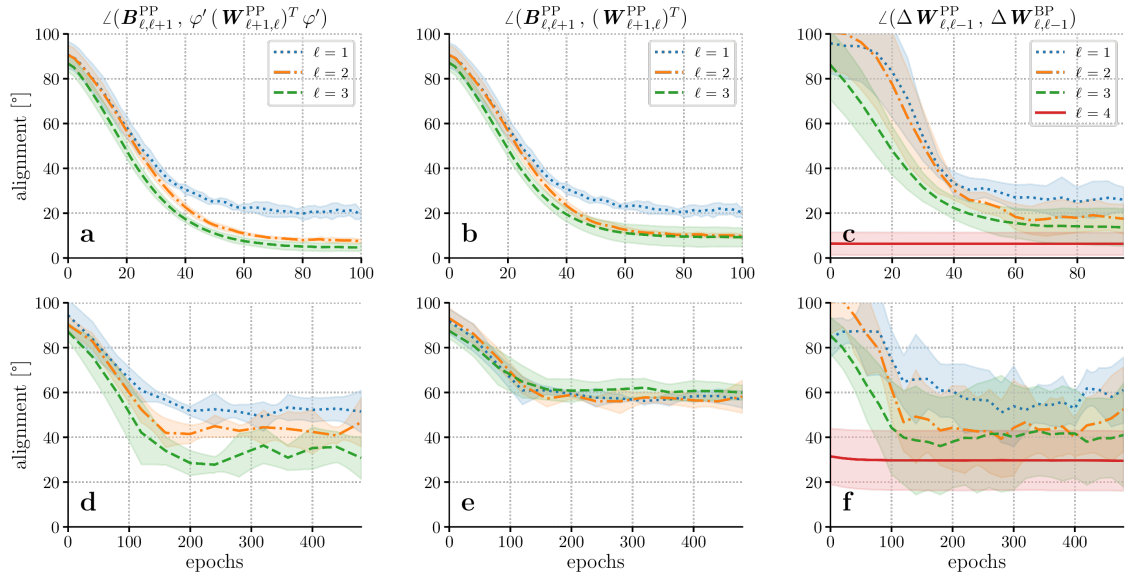

Figure 6.7: PAL aligns weight updates with backpropagation in deep networks. Top: We train the backward projections in a deep microcircuit network with layer sizes [5-20-10-20-5] and sigmoid activation with no target present. All backward weights $\boldsymbol{B}^{\mathrm{PP}}_{\ell,\ell+1}$ are learned simultaneously, while forward weights are fixed. Lines and shading show mean and standard deviation over 5 seeds. Weights are initialized as $\boldsymbol{W}^{\mathrm{PP}} \sim \mathcal{U}[-1, 1]$, such that neurons are activated in their linear regime. The right column compares the *potential* forward weight updates generated from backpropagation using $\boldsymbol{B}^{\mathrm{PP}}_{\ell,\ell+1}$ in the microcircuit model to those in an ANN with BP (see main text and Methods). **Bottom:** Same as above, but with weights initialized in non-linear regime, $\boldsymbol{W}^{\mathrm{PP}} \sim \mathcal{U}[-5, 5]$. Weight updates (f) are biased towards misalignment due to the dendritic microcircuit model, see Methods.

error which propagates to all layers, and from which a forward weight update $\Delta \boldsymbol{W}^{\mathrm{PP}}_{\ell,\ell-1}$ is constructed; see Methods for details. We stress that here, no weight update is applied, as in this experiment, we only demonstrate learning of top-down weights. We compare $\Delta \boldsymbol{W}^{\mathrm{PP}}_{\ell,\ell-1}$ of this microcircuit model to the weight updates $\Delta \boldsymbol{W}^{\mathrm{BP}}_{\ell,\ell-1}$ in an ANN with backpropagation and equivalent feed-forward weights. The results in the third column demonstrate that PAL is able to propagate useful error signals through alignment of backward weights.

Teacher-student setup To further demonstrate that PAL enables propagation of useful error signals, we turn to a simple teacher-student task. A microcircuit model consisting of a chain of two neurons is trained with PAL and, for comparison, random fixed backwards weights. Plasticity is enabled in all synapses, i.e. forward weights are adapted, too. A teaching signal is obtained from a similar two-neuron chain connected with fixed and positive weights. The teacher chain produces a non-linear input-output mapping determined by the

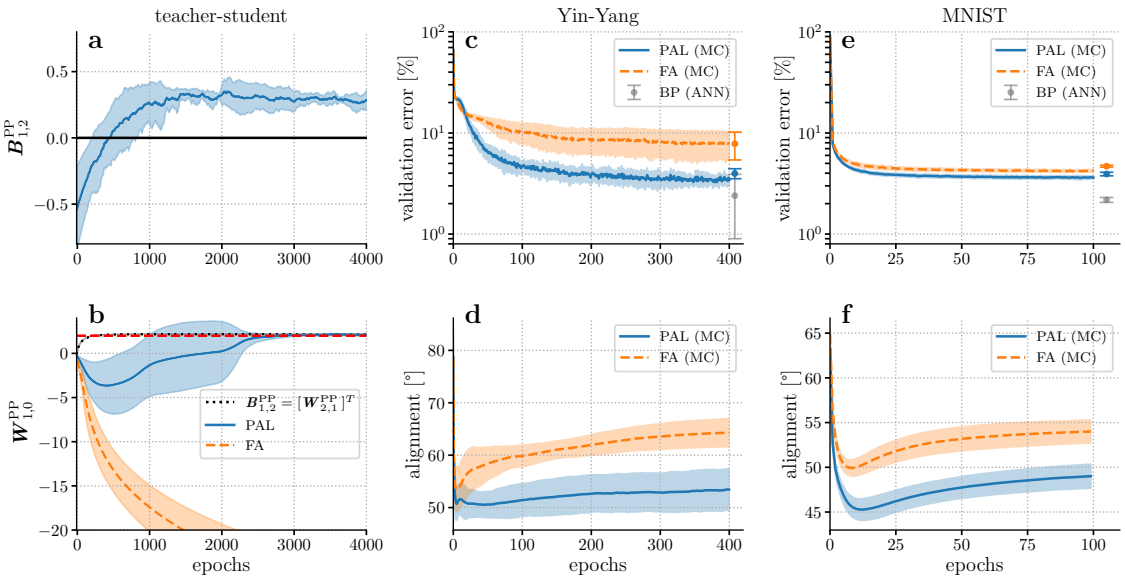

Figure 6.8: PAL improves learning on teacher-student and classification tasks compared to fixed random synaptic feedback. Left column: A chain of two neurons learns to mimic a teaching signal. The student neurons are initialized with negative weights and need to flip the sign. In particular, in order to achieve correct weights to the hidden neuron $\boldsymbol{W}_{1,0}^{\text{PP}}$, positive feedback weights are required. The teacher (red) has a positive weight. Results for student neuron chains are shown for PAL (blue) and FA (orange). The shading indicates mean and standard deviation over 5 seeds. Due to the wrong sign of the transported error, random synaptic feedback fails to solve the task, and weights diverge. The models trained with PAL start in a similar fashion; however, after sign flip (at about 500 epochs, see **a**), the error signal becomes useful, and $\boldsymbol{W}_{1,0}^{\text{PP}}$ converges to the weight of the teacher. As a control, we also show the ideal solution with weight transport, $\boldsymbol{B}_{1,2}^{\text{PP}} = (\boldsymbol{W}_{2,1}^{\text{PP}})^T$. **Center column:** Validation error during training and test errors for the Yin-Yang task (**c**) of the microcircuit model (MC) with network size [4-30-3]. For reference, we also show the test error in an ANN trained with BP with equal network size. The shading indicates mean and standard deviation over 10 seeds. **d:** Alignment angle between backwards weights $\boldsymbol{B}_{1,2}^{\text{PP}}$ and $(\boldsymbol{W}_{2,1}^{\text{PP}})^T$. While FA relies on alignment of forward weights, PAL improves on this by aligning the backward weights with the transpose of forward weights. **Right column:** Same as center column, but for the MNIST data set with network size [784-100-10].

choice of its synaptic weights. The task of the student is to adapt its weights to reproduce this input-output relationship.

In order to highlight an important shortcoming of fixed feedback weights, we initialize the student models with negative forward and backward weights. As shown in Fig. 6.8, a model trained with fixed random feedback weights is not able to reproduce the teacher output and even has diverging weights. This is caused by the wrong sign of the top-down weights: a positive error on the output layer is projected backwards through the negative synapse $\boldsymbol{B}_{1,2}^{\text{PP}}$. Thus, the weight $\boldsymbol{W}_{1,0}^{\text{PP}}$ to the hidden layer grows negatively, further increasing the disparity between teacher and student weights. PAL resolves this issue by approximately

aligning $\boldsymbol{B}_{1,2}^{\text{PP}}$ with $\boldsymbol{W}_{2,1}^{\text{PP}}$, thereby learning backwards weights with correct sign (left column in Fig. 6.8). Note that as as long as $\boldsymbol{B}_{1,2}^{\text{PP}}$ has not yet aligned, $\boldsymbol{W}_{1,0}^{\text{PP}}$ moves in the wrong direction, but as soon as $\boldsymbol{B}_{1,2}^{\text{PP}}$ switches sign (at epoch ~ 500), the forward weight is able to learn correctly[11].
For comparison, we also show the ideal however bio-implausible case corresponding to BP, where top-down weights are set to the same value as the forward weights to the output neuron.

Classification experiments We now turn to more complex tasks and evaluate PAL on classification benchmarks, while still working with the biologically plausible microcircuits as the base model. Due to the complexity of simulating microcircuit models with full dynamics, we focus this evaluation on the computationally effective Yin-Yang task, and perform experiments on MNIST digit classification as a sanity check.
The Yin-Yang classification problem (Kriener et al., 2022) is designed to be a computationally inexpensive task which nonetheless requires useful error signals to reach the lower layer of a network, i.e. is able to differentiate the error propagation quality between FA and BP or variants of it. The task consists in learning to map 2d input coordinates correctly to three distinct categories. ANNs trained with backprop can solve this task with as few as 30 hidden neurons (test error (2.4 ± 1.5) %) (Kriener et al., 2022). This requires the formation of a useful hidden layer representation, which is more likely if backwards weights are adapted instead of random and fixed.
The microcircuit models with PAL achieve a test error of (4.0 ± 0.4) %, performing considerably better than microcircuits with fixed random feedback weights at (7.8 ± 2.4) % (Fig. 6.8, center column). This is reflected in the increased alignment between the transpose of forward and backward weights.
We also perform the MNIST digit classification task with a similar setup (right panel in Fig. 6.8). In a similar vein to the Yin-Yang experiments, this single and small hidden layer is chosen as to highlight whether a good latent representation is formed. We achieve a final test error (3.9 ± 0.2) % using PAL and (4.7 ± 0.1) % with microcircuits with FA.
We highlight that our results were obtained by simulating a fully dynamical, recurrent and bio-plausible system with weight and voltage updates applied at every time step. This is in contrast to previous simulations using bio-plausible networks with recurrency, where simplified network dynamics were assumed for computational feasibility (Sacramento et al., 2018; Greedy et al., 2022). Such approximations do not accurately reproduce the dynamics of recurrent physical networks.[12]

[11] This also requires the weights $\boldsymbol{W}_{2,1}^{\text{PP}}$ to be positive, which is achieved independently of the method (PAL/FA) through learning of the top-layer (not shown).

[12] Simulations on the MNIST task in Sacramento et al. (2018) use the steady state approximation of voltage dynamics, and weight updates are calculated in two distinct steps. In effect, this simplifies the recurrent dynamics defined by Eqn. (6.37) to those of an ANN with separate forward and backward phases and voltage buffering.

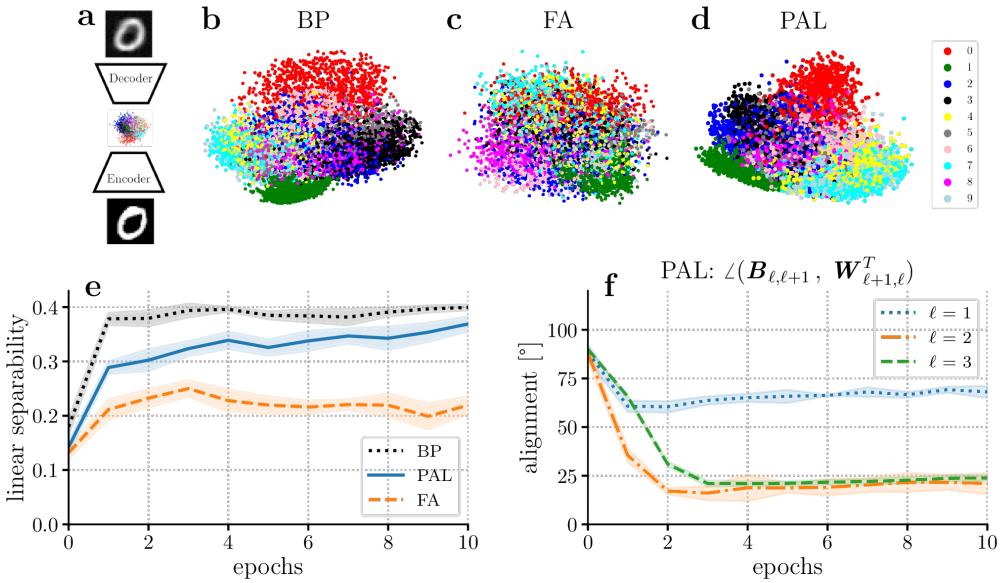

Figure 6.9: PAL learns useful latent representations, where feedback alignment fails to do so. **a:** Encoder/decoder setup network with size [784-200-2-200-784]. MNIST digit dataset is fed into an encoder network with two output neurons. A stacked decoder network aims to reproduce the original input. **b-d:** Latent space activations after training. We show the activations after training in the two-neuron layer of one seed for all samples in the test set; colors encode the corresponding label. Backpropagation and PAL show improved feature separation compared to feedback alignment. **e:** Linear separability of latent activation. **f:** Alignment angle of top-down weights to all layers for the PAL setup (mean and standard deviation over 5 seeds). PAL is able adapt top-down weights while forward weights are also learned. Note that the layer $\ell = 1$ maps 200 neurons onto two in the forward direction, leading to many possible solutions in the backwards direction, and hence a larger alignment angle is to be expected.

Efficient credit assignment in deep networks The previous analyses have shown that PAL can learn useful backprojections in dynamical systems. The simulations performed with microcircuit models stress the bio-plausibility of PAL. However, the microcircuit model (both with and without PAL) carries the issue that error signals decay with increasing hidden layer number (see Methods). PAL is designed to learn useful backprojections in deep hierarchies; in order to demonstrate the full capability of our method, we now relax our requirement for bio-plausible error transport and shift away from the dendritic microcircuit model.

We revisit the general leaky integrator model defined by Eqn. (6.33). The difference between this simpler model and the microcircuit model is the exclusion of interneurons. Instead, errors are transported directly via $\boldsymbol{e}_\ell = \varphi'(\breve{\boldsymbol{u}}_\ell) \cdot \boldsymbol{B}_{\ell,\ell+1}\, \boldsymbol{e}_{\ell+1}$.

We demonstrate the capability of PAL for credit assignment using the MNIST-autoencoder task (Fig. 6.9) (Lansdell et al., 2019). Autoencoders can be used to to greatly compress an input image to a latent representation, in this case reducing down to a vector of dimension two. In order to decode such a representation, a successfully trained autoencoder network should show a separation of input of different classes in the latent space. To learn a well separated latent representation, suitable error signals need to travel through the whole network to train the encoder weights.
We evaluate the latent space separability by training a linear classifier. Results shown in Fig. 6.9 (e) demonstrate that networks trained with PAL achieve linear separability close to BP ((37 ± 1) % vs. (40 ± 1) %, respectively), while training with FA leads to significantly poorer linear separability at a test accuracy of (22 ± 2) %. Figure 6.9 (f) shows that PAL is able to learn the transpose of forward weights across hidden layers. Our results imply that with PAL, the network is able to transport useful error signals and learn suitable weights throughout all hidden layers, whereas FA leads to poor feature separation.

6.4.3 Discussion

We have introduced phaseless alignment learning (PAL), a general method of learning backprojections in hierarchical, dynamical networks. Our theoretical results and simulations show that PAL provides online learning of forward and backward weights in a phase-less manner. As a general method, it is applicable to models where time-continuous activity is propagated, and approximately aligns feedback weights with those of backpropagation. PAL fulfills the requirements of learning and signal transport in physical systems: all necessary information is available locally in time and space.
In our evaluation, we have emphasized the biological plausibility of PAL as a model of sensory processing. PAL could be implemented in biological components; in particular, it explicitly exploits noise found in physical systems and makes use of simple filtering techniques for disentangling signal and noise where needed. We argue that a cortical realization of PAL (or a variant) would be evolutionarily more advantageous than fixed feedback weights, as it implements a significantly more efficient solution to the weight transport problem.
Our simulation results show that PAL is able to outperform FA in terms of credit assignment in biologically plausible, recurrently connected networks. The requirements for PAL are quite general, and we stress that the dendritic microcircuit model with PAL is only one possible implementation, which however is notable for its bio-plausible error transport. In principle, it can be argued that a test error comparable to PAL could have been achieved with fixed random backprojections by scaling up the hidden layer size (not shown). Nevertheless, it is advantageous for a method to perform well while requiring less neurons/synapses, for biological plausibility but also energetic efficiency. It has been suggested that the inefficiency of FA can mitigated through direct random backprojections (direct feedback alignment, DFA), where the feedback signal from the output layer is sent directly to all hidden

layers, passing only through a single random feedback matrix (Nøkland, 2016; Crafton et al., 2019). However, in the context of cortical hierarchies, this presupposes skip connections from one higher cortical area to all lower areas instead of layer-wise connections. While such connections have been observed in the cortex, the visual stream is largely organized in hierarchical manner, with significantly weaker correlation between non-neighboring areas (Haak and Beckmann, 2018).

However, like any algorithm based on learning from data samples, weights trained with PAL are data-specific. That is, top-down weights are aligned with an average over many input samples $\boldsymbol{r}_0$ as $\boldsymbol{B}_{\ell,\ell+1} \propto \mathbb{E}[\varphi' \, [\boldsymbol{W}_{\ell+1,\ell}]^T \, \varphi']_{\boldsymbol{r}_0}$, leading to imperfect alignment of transported errors with those of backpropagation (see Fig. 6.7). Lower learning rates η^{bw} lead to an inclusion of more samples into the expectation value; by sampling over the whole training set, data-dependency can be minimized — however, as forward weights need to evolve slower than backward weights, this leads to slow overall learning. In contrast to this, Ernoult et al. (2022) circumvents this issue by separate phases of forward and backward learning for each data sample, but no fully on-line solution is currently known. As we have shown, the error transported by this data-specific weights can still be efficient in learning to solve complex tasks.

Note that PAL is able to make full use of prospective coding — i.e., that all information propagation occurs through prospective rates $\varphi(\breve{\boldsymbol{u}}_\ell)$, which converge to their steady state quasi-instantaneously. In particular, this ensures that the weight update rules are constructed from useful learning signals at all times, not only after the neuronal dynamics have settled into a steady state. As a consequence, we were able to simulate our dynamical system with fully continuous voltage dynamics and learning of all synapses enabled at all times. This is an important difference compared to previously known bio-plausible mechanisms of learning in dynamical systems, which have mitigated the issue of slow relaxation through slow or phased learning (see e.g. O'Reilly (1996); Bengio and Fischer (2015); Sacramento et al. (2018); Guerguiev et al. (2017); Scellier and Bengio (2017); Mesnard et al. (2019); Xie and Seung (2003); Song et al. (2022)), and/or by re-initializing the somatic potentials to their bottom-up input state for every data sample (Scellier and Bengio, 2017; Sacramento et al., 2018; Meulemans et al., 2021).

As our theory is based on a rate-based abstractions of neural dynamics, there remain several open questions of bio-plausibility and realism. Extensions to our theory may implement Dale's law of either inhibitory or excitatory activity, a constraint which could be realized by separate populations of neurons (Cornford et al., 2021). Equally, the dendritic microcircuit model could model biology more closely by implementing spiking neuronal output, and arguing that credit is assigned through a probabilistic interpretation.

Our results clearly point towards future work on theories of on-line learning in the cortex and on neuro-inspired hardware. Building on similar ideas in the literature (Faisal et al., 2008; McDonnell and Ward, 2011; Maass, 2014; Rusakov et al., 2020), we hypothesize a general principle of using noise in physical systems for learning, instead of considering it

an undesirable side effect when modeling substrates. PAL can serve as a blueprint for the greater philosophy of viewing noise as a resource rather than a nuisance.

6.4.4 Methods

As in the main text, bold lowercase (uppercase) variables $\boldsymbol{x}$ ($\boldsymbol{X}$) denote vectors (matrices). The partial derivative of the activation given by $\boldsymbol{r}_\ell = \varphi(\breve{\boldsymbol{u}})$ is denoted by $\varphi'(\breve{\boldsymbol{u}})$, which is a diagonal matrix with μ-th entry $\frac{\partial r_\mu}{\partial \breve{u}_\mu}$.

Prospective Coding

As neurons in our theory are modelled by leaky integrators, the somatic voltage follows the low-pass filtered sum of input currents, and therefore exhibits a slow response to its input. This effect multiplies with increasing layer number, resulting in the requirement to present an input for many membrane time constants to allow both input signals from the bottom and learning signals from the top to fully propagate through the whole network. Additionally, slow neuron dynamics do not only slow down the flow of information, but also introduce incorrect error signals, as demonstrated in Haider et al. (2021).
Several schemes have been proposed to solve this issue. A common fix is scheduled plasticity, where synapses are only learned once the system has settled into the equilibrium state (O'Reilly, 1996; Bengio and Fischer, 2015; Sacramento et al., 2018; Guerguiev et al., 2017; Scellier and Bengio, 2017; Mesnard et al., 2019; Xie and Seung, 2003; Song et al., 2022). This leads to slow learning, and the need to explain the phased plasticity through a biologically plausible mechanism. In Haider et al. (2021) and our model however, the issues caused by response lag are overcome by calculating the firing rate of each neuron based on the prospective future voltage: all neural outputs and weight updates are calculated from the prospective voltage $\breve{\boldsymbol{u}} \coloneqq \boldsymbol{u} + \tau_{\text{eff}} \frac{d\boldsymbol{u}}{dt}$. The implementation of prospective coding with PAL is essential for fast transfer of information, ensuring quick convergence of weights.

Alignment of feedback weights

We show how the weight transport problem can be solved through alignment of top-down weights. We keep our description general by discussing the basic leaky integrator model with noise,

$$\tau_{\text{eff}} \dot{\boldsymbol{u}}_\ell = -\boldsymbol{u}_\ell + \boldsymbol{b}_\ell + \boldsymbol{W}_{\ell,\ell-1}\boldsymbol{r}_{\ell-1} + \boldsymbol{e}_\ell + \boldsymbol{\xi}_\ell \tag{6.39}$$

with $\boldsymbol{e}_\ell$ propagated downwards from the upper layer error through feedback connections $\boldsymbol{B}_{\ell,\ell+1}$. As detailed in Section 6.4.2, we do not discuss error propagation itself, but focus instead on the learning of feedback weights from the rates generated within each layer. The procedure to learn the backwards weights relies on two recent theoretical advancements: the local difference reconstruction loss defined in Ernoult et al. (2022) trains back-

wards weights to approximate backpropagation, whereas the proposal of Meulemans et al. (2021) to consider Ornstein-Uhlenbeck noise enables us to learn all backwards weights simultaneously.

To begin, we model Ornstein-Uhlenbeck noise $\boldsymbol{\xi}_k$ in each hidden layer. The noise signal is generated from low-pass filtered white noise,

$$\dot{\boldsymbol{\xi}}_\ell(t) = -\frac{1}{\tau_\xi}\left[\boldsymbol{\xi}_\ell(t) - \boldsymbol{\mu}_\ell(t)\right], \tag{6.40}$$

with $\boldsymbol{\mu}_\ell(t) \sim \mathcal{N}(0, \sigma^2)$, and low-pass filtering constant τ_ξ smaller than the usual presentation time T_{pres} of input signals $\boldsymbol{r}_0$. Therefore, $\boldsymbol{\xi}_k$ represents an Ornstein-Uhlenbeck process with high frequency compared to the inference and error signals. We choose the scale of noise σ such that it is small compared to the somatic potential in all hidden layers. If we denote as $\varphi(\breve{\boldsymbol{u}}_k^0)$ the neuron output in absence of noise, we can expand to first order in small noise, $\boldsymbol{r}_k \approx \varphi(\breve{\boldsymbol{u}}_k^0) + \varphi'(\breve{\boldsymbol{u}}_k^0)\,\boldsymbol{\xi}_k$. This signal is sent to the corresponding higher layer $k+1$ through the weight $\boldsymbol{W}_{k+1,k}$, where the upper layer noise $\boldsymbol{\xi}_{k+1}$ is added on top. This continues through all layers up to the output layer. Hence, for a given layer $\ell+1$, the output rate is modified by noise to be[13]

$$\boldsymbol{r}_{\ell+1} \approx \varphi(\breve{\boldsymbol{u}}_{\ell+1}^0) + \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0)\,\boldsymbol{\xi}_{\ell+1} + \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0) \sum_{m=1}^{\ell}\Big[\prod_{n=m}^{\ell} \boldsymbol{W}_{n+1,n}\varphi'(\breve{\boldsymbol{u}}_n^0)\Big]\,\boldsymbol{\xi}_m\,. \tag{6.41}$$

This noise-inclusive rate is also propagated top-down. In the case of layer-wise feedback connections, $\boldsymbol{r}_{\ell+1}$ is sent to layer ℓ through the synapse $\boldsymbol{B}_{\ell,\ell+1}$. Therefore, the information locally available to learn useful top-down weights is restricted to the pre-synaptic rate $\boldsymbol{r}_{\ell+1}$ and the post-synaptic potential including noise.

We aim now to learn the top-down synapses by exploiting the auto-correlation of noise. The slow portion $\varphi(\breve{\boldsymbol{u}}_{\ell+1}^0)$ of the pre-synaptic signal is not useful for learning of backwards weights, as it contains correlations across layers which cannot be canceled. Therefore, we extract the noise-induced portion of the signal with a high-pass filtered version of the top-down rate, $\widehat{\boldsymbol{r}}_{\ell+1}$ (Meulemans et al., 2021). The dynamics of $\boldsymbol{B}_{\ell,\ell+1}$ are derived from gradient descent on the local alignment loss $\mathcal{L}_\ell^{\text{PAL}}$, cf. Eqn. (6.34). This yields the learning rule

$$\dot{\boldsymbol{B}}_{\ell,\ell+1} = \eta_\ell^{\text{bw}}\left[\boldsymbol{\xi}_\ell\,\left(\widehat{\boldsymbol{r}}_{\ell+1}\right)^T - \alpha\,\boldsymbol{B}_{\ell,\ell+1}\right] \tag{6.42}$$

[13] In this description, we omit noise originating in downstream areas ($\xi_{\ell+2}, \ldots, \xi_N$) as it quickly averages to zero (see below).

We determine the fixed point of the feedback weights by taking the expectation value over many noise samples for fixed inputs and weights,

$$0 \stackrel{!}{=} \mathbb{E}\big[\dot{\boldsymbol{B}}_{\ell,\ell+1}\big]_{\boldsymbol{\xi}} \tag{6.43}$$

$$= \mathbb{E}\Big[\eta_\ell^{\text{bw}}\big\{\boldsymbol{\xi}_\ell \left(\widehat{\boldsymbol{r}}_{\ell+1}\right)^T - \alpha\, \boldsymbol{B}_{\ell,\ell+1}\big\}\Big]_{\boldsymbol{\xi}} \tag{6.44}$$

$$\Rightarrow \quad \mathbb{E}\big[\boldsymbol{B}_{\ell,\ell+1}\big]_{\boldsymbol{\xi}} = \frac{1}{\alpha}\,\mathbb{E}\big[\boldsymbol{\xi}_\ell \left(\widehat{\boldsymbol{r}}_{\ell+1}\right)^T\big]_{\boldsymbol{\xi}} \tag{6.45}$$

Using Eqn. (6.41), the right hand side can be expanded,

$$\mathbb{E}\big[\boldsymbol{B}_{\ell,\ell+1}\big]_{\boldsymbol{\xi}} \approx \frac{1}{\alpha}\mathbb{E}\Big[\boldsymbol{\xi}_\ell\Big\{\varphi'(\breve{\boldsymbol{u}}^0_{\ell+1})\,\boldsymbol{\xi}_{\ell+1} + \varphi'(\breve{\boldsymbol{u}}^0_{\ell+1})\sum_{m=1}^{\ell}\Big[\prod_{n=m}^{\ell}\boldsymbol{W}_{n+1,n}\varphi'(\breve{\boldsymbol{u}}^0_n)\Big]\,\boldsymbol{\xi}_m\Big\}^T\Big]_{\boldsymbol{\xi}} \tag{6.46}$$

We now make use of the fact that the auto-covariance of Ornstein-Uhlenbeck noise decays exponentially in time,

$$\mathbb{E}\big[\boldsymbol{\xi}_\ell(t+\Delta t)\;\boldsymbol{\xi}_k(t)^T\big]_{\boldsymbol{\xi}} = \mathbf{1}\,\delta_{k,\ell}\,\frac{\sigma^2}{2}\,e^{-|\Delta t|/\tau_\xi}\;, \tag{6.47}$$

where $\delta_{k,\ell}$ is the Kronecker delta, σ^2 the variance, and Δt corresponds to the time needed for a noise sample to travel to the layer above and back. As noise originating in different layers is uncorrelated, it quickly averages to zero, whereas the correlation of noise samples at different times generated at the same layer is non-zero. We thus have as our final result

$$\mathbb{E}\big[\boldsymbol{B}_{\ell,\ell+1}\big]_{\boldsymbol{\xi}} \approx \frac{1}{\alpha}\,\frac{\sigma^2}{2}\,e^{-|\Delta t|/\tau_\xi}\big\{\varphi'(\breve{\boldsymbol{u}}^0_\ell)\,\big[\boldsymbol{W}_{\ell+1,\ell}\big]^T\varphi'(\breve{\boldsymbol{u}}^0_{\ell+1})\big\}\;. \tag{6.48}$$

The curly brackets show the alignment between feedback and feedforward weights.

We now incorporate this result with the learning rule for bottom-up weights $\boldsymbol{W}_{\ell,\ell-1}$. As discussed in Section 6.4.2, error propagation mechanisms generally produce a layer-wise error $\boldsymbol{e}_\ell$ as a function of top-down synapses $\boldsymbol{B}_{\ell,\ell+1}$.

Models closely related to backpropagation (Xie and Seung, 2003; Scellier and Bengio, 2017; Haider et al., 2021) employ forward weight updates of the form

$$\begin{aligned}\dot{\boldsymbol{W}}_{\ell,\ell-1} &= \eta_\ell^{\text{fw}}\,\boldsymbol{e}_\ell\,\boldsymbol{r}_{\ell-1}^T \\ &= \eta_\ell^{\text{fw}}\,\big\{\varphi'(\breve{\boldsymbol{u}}^0_\ell)\,\boldsymbol{B}_{\ell,\ell+1}\,\boldsymbol{e}_{\ell+1}\big\}\,\boldsymbol{r}_{\ell-1}^T\;,\end{aligned} \tag{6.49}$$

whereas models where rates are propagated top-down (Lee et al., 2015; Sacramento et al., 2018; Meulemans et al., 2020) can be generally described by

$$\dot{\boldsymbol{W}}_{\ell,\ell-1} = \eta_\ell^{\text{fw}} \left\{ \boldsymbol{B}_{\ell,\ell+1}\, \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0)\, \boldsymbol{e}_{\ell+1} \right\} \boldsymbol{r}_{\ell-1}^T \,. \tag{6.50}$$

We plug our result for learned top-down weights, Eqn. (6.48), into these update rules. Because we have based our derivation on a general leaky-integrator model, our result in the form of $\mathbb{E}\left[\boldsymbol{B}_{\ell,\ell+1}\right] \propto \varphi'(\breve{\boldsymbol{u}}_\ell^0) \left[\boldsymbol{W}_{\ell+1,\ell}\right]^T \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0)\ \forall\ \ell$ can be employed in any of these theories. Plugging into either Eqn. (6.49) or Eqn. (6.50), we see that the weight updates $\Delta \boldsymbol{W}_{\ell,\ell-1}$ align with those of a feed-forward network trained with backpropagation up to additional factors of derivatives. Therefore, our algorithm can provide useful error signals which approximately align with backpropagation, improving on random feedback weights.

Dendritic cortical microcircuits

We now describe learning through minimization of the dendritic error as proposed in Sacramento et al. (2018). In this model, weights are adapted using local dendritic plasticity rules in the form $\dot{\boldsymbol{W}} = \eta \left[\varphi(\boldsymbol{u}) - \varphi(\boldsymbol{v})\right] \boldsymbol{r}^T$, where $\boldsymbol{W}$ represents lateral or feed-forward weights, η is a learning rate, $\boldsymbol{u}$ and $\boldsymbol{v}$ denote different compartmental voltages and $\boldsymbol{r}$ the pre-synaptic rate (Urbanczik and Senn, 2014; Gerstner et al., 2018). In order to adhere to the principle of bio-plausibility, the microcircuit model uses rules such that $\boldsymbol{u}$ corresponds to the soma of a given neuron and $\boldsymbol{v}$ to the corresponding compartment the synapse connects to. The concrete form of all learning rules can be found in Supplementary Information. They are designed such that the system settles in a specific state, where activity of the apical dendrite of hidden layer pyramidal cells represents an error useful for learning.

We first describe learning using fixed random feedback connections. Before supervised training, the system is run in absence of a teaching signal with random input $\boldsymbol{r}_0(t)$. As demonstrated in Sacramento et al. (2018), the plasticity of $\boldsymbol{W}_{\ell,\ell}^{\text{IP}}$ and $\boldsymbol{B}_{\ell,\ell}^{\text{PI}}$ allow the system to settle in a *self-predicting state*. This is a system state described by matching voltages between interneurons and pyramidal cells, $\boldsymbol{u}_\ell^{\text{I}} = \boldsymbol{u}_{\ell+1}^{\text{P}}$, together with zero apical voltages $\boldsymbol{v}_\ell^{\text{api}}$. This second condition is achieved in the hidden layers by learning lateral weights such that top-down and lateral activity cancel, i.e. $\boldsymbol{B}_{\ell,\ell}^{\text{PI}}\, \boldsymbol{r}_\ell^{\text{I}} = -\boldsymbol{B}_{\ell,\ell+1}^{\text{PP}}\, \boldsymbol{r}_{\ell+1}^{\text{P}}$. We now turn on an instructive signal by clamping the apical compartment of top layer pyramidal cells to the target voltage $\boldsymbol{u}^{\text{tgt}}$. The somata of these neurons integrate the target signal with the bottom-up input and propagate it top-down to the hidden layers. In the limit of small conductances (weak nudging), the pyramidal neurons are now slightly nudged towards the target, while the interneurons in the hidden layers do not observe the teaching signal; therefore, they represent the activity which pyramidal cells in the layer above would have if there were no target. As the apical dendrite calculates the difference between top-down and lateral activity, it now encodes the error signal passed down to the hidden layers. This is how cortical microcircuits with dendritic error encoding assign credit to hidden layers neurons.

In fact, the microcircuit model implements difference target propagation (Lee et al., 2015) in a dynamical system with recurrency, with $\boldsymbol{v}_\ell^{\text{api}} = \boldsymbol{B}_{\ell,\ell+1}^{\text{PP}}(\boldsymbol{r}_{\ell+1}^{\text{P}} - \boldsymbol{r}_\ell^{\text{I}})$ representing the backprojected difference target $g_\ell(\hat{\boldsymbol{h}}_{\ell+1}) - g_\ell(\boldsymbol{h}_{\ell+1})$.[14]
PAL is implemented naturally by the inclusion of noise and our learning rule Eqn. (6.35). Contrary to simulations with fixed top-down weights (Sacramento et al., 2018; Haider et al., 2021), the inclusion of PAL also requires dynamical lateral weights from interneurons to pyramidal cells. For efficient learning, a tight balance between learning rates needs to be kept: lateral weights $\boldsymbol{W}_{\ell,\ell}^{\text{IP}}$ need to adapt quickly to any changes of feed-forward weights $\boldsymbol{W}_{\ell+1,\ell}^{\text{PP}}$, while top-down weights $\boldsymbol{B}_{\ell,\ell+1}^{\text{PP}}$ need to adapt to changing forward weights quickly; on the other hand, lateral weights $\boldsymbol{B}_{\ell,\ell}^{\text{PI}}$ from interneurons to pyramidal cells need to adapt quickly to changing top-down weights, such that no spurious error occurs. The precise order of weights updates is $|\Delta \boldsymbol{W}_{\ell+1,\ell}^{\text{PP}}| < |\Delta \boldsymbol{B}_{\ell,\ell+1}^{\text{PP}}| < |\Delta \boldsymbol{B}_{\ell,\ell}^{\text{PI}}| \lesssim |\Delta \boldsymbol{W}_{\ell,\ell}^{\text{IP}}|$.
We comment on the ability of cortical microcircuits to assign credit over hierarchies with multiple hidden layers. While Eqn. (6.38) implies that in theory, tasks can be learned successfully using many hidden layers, the derivation assumes perfect cancellation of the top-down signal with the interneuron activity, such that only the error signal is encoded in the apical dendrite. This requires a perfect self-predicting state, which is unattainable in practice unless learning is phased (learning phases interleaved with phases where no target is present and the self-predicting state is reestablished). As the error signal scales with the small nudging strength λ, early layers in the network receive instructive signals which are weaker by orders of magnitude, further complicating realistic evaluations of the model. For this reason, we restrict our simulations showing credit assignment in cortical microcircuits to a single hidden layer (see Section 6.4.2).

Simulation details

In all experiments, dynamics are simulated in discrete time steps of length dt using the Euler-Maruyama method (Särkkä and Solin, 2019). Input and targets are passed as data streams in the form of vectors presented for $T_{\text{pres}} = 100\, dt$ without any kind of filtering or pre-processing. In all simulations, voltage and weight updates (where non-zero) are applied at all time steps. All layers are fully connected throughout all experiments.
Microcircuits are simulated by defining effective voltages $\boldsymbol{u}_{\text{eff}}$ as described in Haider et al. (2021): we rewrite all dynamical equations in the form $C_m \dot{\boldsymbol{u}} = \frac{1}{\tau_{\text{eff}}}(\boldsymbol{u}_{\text{eff}} - \boldsymbol{u})$, where $\boldsymbol{u}$ denotes the pyramidal or interneuron somatic potential. Models are initialized in the self-predicting state defined (see Supplement, and Sacramento et al. (2018)). Before training, we allow the voltages to equilibrate during a brief settling phase (several dt). Activation functions are the same throughout all layers, including the output layer. Targets are provided as a target voltage $\boldsymbol{u}^{\text{tgt}}$ at the output layer.

[14] Independent of the error propagation scheme, the framework of DTP also includes the learning of top-down weights such that the system's cost is minimized according to Gauss-Newton optimization with batch size 1 (Meulemans et al., 2020).

For the PAL implementations, we calculate Ornstein-Uhlenbeck noise by sampling white noise $\boldsymbol{w} \sim \mathcal{N}(0, 1)$ and low-pass filtering with time constant τ_ξ, that is, $\boldsymbol{\xi}_\ell[t + dt] = \boldsymbol{\xi}_\ell[t] + \frac{1}{\tau_\xi}(\sqrt{\tau_\xi\, dt}\, \sigma_\ell\, \boldsymbol{w} - dt\, \boldsymbol{\xi}_\ell[t])$. High-pass filtered rates $\widehat{\boldsymbol{r}}_\ell$ are calculated with respect to the time constant τ_{hp} through $\frac{d\widehat{\boldsymbol{r}}_\ell^{\text{P}}}{dt} = \frac{d\boldsymbol{r}_\ell^{\text{P}}}{dt} - \frac{\widehat{\boldsymbol{r}}_\ell^{\text{P}}}{\tau_{\text{hp}}}$. Forward weight updates low-pass filtered with τ_{lo} before application.
For pseudocode, all parameters and architecture details, see Supplementary Information.

Phaseless backwards weight alignment We simulate a microcircuit network of size [5-20-10-20-5] with sigmoid activation. Linear and non-linear regimes are simulated by choosing bottom-up weights as $\boldsymbol{W}^{\text{PP}} \sim \mathcal{U}[-1, 1]$ and $\sim \mathcal{U}[-5, 5]$, respectively. Forward weights are fixed, while top-down and lateral (inter- to pyramidal neuron) weights are learned.
During evaluation against BP (right column in Fig. 6.7), we set the lateral weights during to the exact self-predicting state, in order to observe an error signal in earlier hidden layers. Note that the weight updates in the microcircuit model do not exactly represent those of an ANN. This is due to an additional factor $\varphi'(\breve{\boldsymbol{u}}_N^{\text{P}})$ as well as the fact that top-down nudging influences the somatic activity (see Supplementary Information for details). These factors introduce a misalignment unrelated to the performance of PAL, as can be seen through the comparison of updates in the non-linear case (Fig. 6.7 (f)); in particular, already in the output layer ($\ell = 4$), a misalignment of $\sim 20°$ can be observed, and hence even perfectly learned backprojections are likely to observe increased misalignment. Therefore, it is to be expected that alignment is improved further in theories of error propagation which relate more closely to backpropagation.

Classification experiments The Yin-yang and MNIST tasks were solved using microcircuit networks of size [4-30-3] and [784-100-10], respectively, with sigmoid activation. All weights (including lateral) were trained with fully recurrent dynamics. For the experiments shown in this section we use the GPU enhanced Neuronal Network simulation environment (GeNN) (Yavuz et al., 2016; Knight et al., 2021). Natively, GeNN supports the simulation of spiking neural networks, but the possibility to add custom neuron and synapse models to the already provided ones makes the implementation of rate-based models such as the dendritic microcircuit possible. The simulation of the dendritic microcircuits benefited greatly from the graphical processing unit (GPU) support provided by GeNN which allowed us to perform the experiments shown in this section within practically feasible simulation times.

Efficient credit assignment in deep networks For this experiment, we simulated the general leaky integrator model, Eqn. (6.33). Network size of the autoencoder is [784-200-2-200-784], with activations [tanh, linear, tanh, linear]. We define as latent space activity the output of the two neurons in the central hidden layer. PAL is implemented by adding Ornstein-Uhlenbeck noise $\boldsymbol{\xi}_\ell$ to each hidden layer neuron, calculating the high-pass filtered rate $\widehat{\boldsymbol{r}}_{\ell+1}$ and updating top-down weights with Eqn. (6.35). The output layer error is defined

as $\boldsymbol{e}_N = \boldsymbol{u}^{\text{tgt}} - \breve{\boldsymbol{u}}_N$. We also include a bias term for each neuron. Forward weight updates are low-pass filtered with time constant τ_{lo}.
As previously, this model is simulated in discrete time steps, with images presented for $T_{\text{pres}} = 100$ dt. Voltages are updated continuously, and weight updates are applied at all steps. After every epoch of training the LI model, we trained a linear classifier on the MNIST train set and show the accuracy of the linear classifier on the test set.

Acknowledgment

We wish to thank Jakob Jordan, Alexander Meulemans and João Sacramento for valuable discussions. We gratefully acknowledge funding from the European Union under grant agreements 604102, 720270, 785907, 945539 (HBP) and the Manfred Stärk Foundation. Additionally, our work has greatly benefited from access to the Fenix Infrastructure resources, which are partially funded from the European Union's Horizon 2020 research and innovation programme through the ICEI project under the grant agreement No. 800858.

6.4.5 Supplementary Information A: Additional information on PAL

In this Supplement, we give more detail on the derivation and application of PAL and the microcircuit implementation used to perform the simulations.
As in the main text, bold lowercase (uppercase) variables $\boldsymbol{x}$ ($\boldsymbol{X}$) denote vectors (matrices). The partial derivative of the activation given by $\boldsymbol{r}_\ell = \varphi(\breve{\boldsymbol{u}})$ is denoted by $\varphi'(\breve{\boldsymbol{u}})$, which is a diagonal matrix with μ-th entry $\frac{\partial r_\mu}{\partial \breve{u}_\mu}$.

Derivation of PAL

We point out how and why our alignment loss $\mathcal{L}_\ell^{\text{PAL}}$, defined in Eqn. (6.34), differs from the reconstruction loss $\widehat{\mathcal{L}}$ introduced in Ernoult et al. (2022). Using the notation of this manuscript, this can be expressed as

$$\widehat{\mathcal{L}}^{\ell}_{\boldsymbol{B}_{\ell,\ell+1}} \mathrel{\hat{=}} -\boldsymbol{\xi}_\ell^T\, \boldsymbol{B}_{\ell,\ell+1}\big\{\boldsymbol{r}_{\ell+1}^{\boldsymbol{\xi}_\ell} - \varphi(\breve{\boldsymbol{u}}^0_{\ell+1})\big\} + \Big\|\boldsymbol{B}_{\ell,\ell+1}\big\{\boldsymbol{r}_{\ell+1}^{\boldsymbol{\xi}_{\ell+1}} - \varphi(\breve{\boldsymbol{u}}^0_{\ell+1})\big\}\Big\|^2\,, \tag{6.51}$$

where $\boldsymbol{r}_{\ell+1}^{\boldsymbol{\xi}_\ell}$ is the rate which comprising data signal and noise from layer ℓ only, while the second term generated from $\breve{\boldsymbol{u}}^0_\ell$ contains no noise signal. The regularizer requires a separate phase, where noise is injected only into layer $\ell+1$ and backpropagated to layer ℓ.
Training feedback weights by gradient descent on this alignment loss represents a case closely related to PAL – for fixed input and forward weights, $\boldsymbol{B}_{\ell,\ell+1}$ converges such that the Jacobians matrices align, $\boldsymbol{B}_{\ell,\ell+1}\,\varphi'(\breve{\boldsymbol{u}}^0_{\ell+1}) \parallel [\varphi'(\breve{\boldsymbol{u}}^0_\ell)\,\boldsymbol{W}_{\ell+1,\ell}]^T$. Inserting this result into the difference target propagation rule of Eqn. (6.50), we see that the update reproduces exact backpropagation with linear activation function on the output layer.

An analogous implementation of gradient descent on $\widehat{\mathcal{L}}$ in our setup requires making use of the noise in the output layer, and using a different kind of regularizer, i.e. $-\alpha \left\| \boldsymbol{B}_{\ell,\ell+1} \widehat{\boldsymbol{r}}_{\ell+1} \right\|^2$ instead of weight decay. Unfortunately, this regularizer contains non-zero correlations of all noise signals up to layer $\ell+1$, and not only auto-correlations of $\boldsymbol{\xi}_\ell$. Therefore, gradient descent on a difference reconstruction loss with this regularizer does not lead to useful top-down weights, as a particular weight $\boldsymbol{B}_{\ell,\ell+1}$ receives contributions proportional to all weights $\boldsymbol{W}_{k+1,k}$ for $k = 1 \dots \ell$. The central reason causing this issue is that our system learns to adapt all feedback weights simultaneously, which requires considering noise in all layers at all times. Contrary to this, in Ernoult et al. (2022), feedback weights are trained sequentially with two separate phases of noise injections in different layers. We have therefore designed Eqn. (6.42) as a heuristic approximation to the optimal update rule, while achieving full always-on plasticity in our system.

Alternatively, the problem of superfluous derivatives φ' can be addressed if the derivative of the activation w.r.t. to the potential is available at the synapse. Given this information, the weight updates can be defined as

$$\dot{\boldsymbol{B}}_{\ell,\ell+1} = \eta_\ell^{\text{bw}} \left[\boldsymbol{\xi}_\ell \left(\widehat{\boldsymbol{r}}_{\ell+1} \right)^T - \alpha \, \varphi'(\breve{\boldsymbol{u}}_\ell) \boldsymbol{B}_{\ell,\ell+1} \varphi'(\breve{\boldsymbol{u}}_{\ell+1}) \right] . \tag{6.52}$$

Note that the derivatives $\varphi'(\breve{\boldsymbol{u}}_\ell)$ are a function of the full somatic potential, comprising data as well as noise. Using the fact that correlations between $\boldsymbol{\xi}_\ell$ and $\boldsymbol{\xi}_{\ell+1}$ cancel to zero, we obtain that the new expectation value of top-down weights to first order,

$$\mathbb{E}\left[\varphi'(\breve{\boldsymbol{u}}_\ell^0) \boldsymbol{B}_{\ell,\ell+1} \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0) \right]_{\boldsymbol{\xi}} \propto \varphi'(\breve{\boldsymbol{u}}_\ell^0) [\boldsymbol{W}_{\ell+1,\ell}]^T \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^0) \, . \tag{6.53}$$

We have tested this alternative regularizer in the relevant regime of in non-linear activation. As shown in Fig. 6.10, it is able to improve alignment of $\boldsymbol{B}_{\ell,\ell+1}^{\text{PP}}$ with $[\boldsymbol{W}_{\ell+1,\ell}^{\text{PP}}]^T$ by about 10° in this example (note that we have used the same parameters as in the PAL setup. With appropriate hyperparameter search, convergence time and final alignment may be improved). However, whether a bio-plausible synapse can calculate the derivative $\varphi' = \frac{\partial \varphi}{\partial \breve{u}}$ is not clear. Given our requirement that all computations can be implemented with simple physical components, we have opted for the weight decay regularizer as defined in Eqn. (6.34).

Error propagation in microcircuits

In this section, we explain in detail how the cortical microcircuit is able to propagate meaningful targets, and align its feedback weights using PAL in order to efficiently minimize the difference between its output and a teaching signal.

We briefly review the general microcircuit setup defined by Sacramento et al. (2018). In this model, the different neuron populations are each selected to play a distinct role. Each hidden layer is composed of a population of pyramidal neurons and interneurons, where

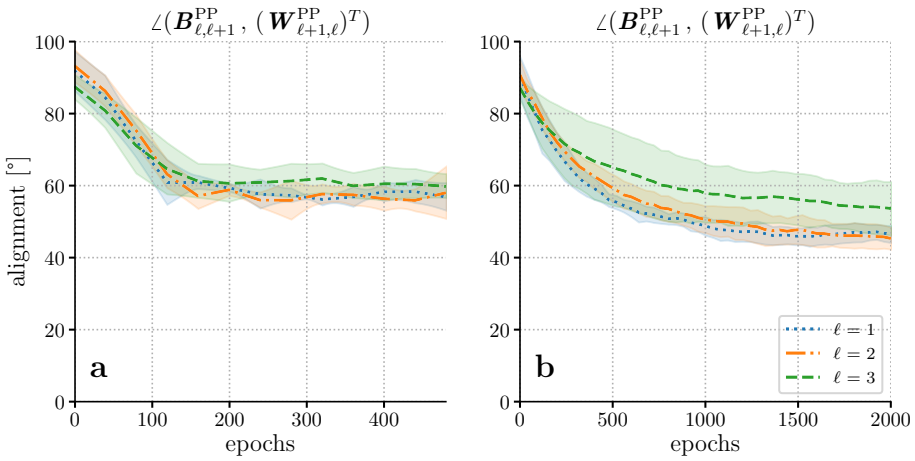

Figure 6.10: Alternative regularizer with derivative shows further improvement in alignment. We reproduce the experiment in Fig. 6.7 (e) using the same parameters: microcircuits learning to adapt backwards weights with PAL using **(a)** the standard weight decay regularizer and **(b)** the derivative-dependent regularizer of Eqn. (6.52).

the number of interneurons in a given layer matches the number of pyramidal cells in the layer above. The neurons form a network defined by connections as shown in Fig. 6.6.

As in Sacramento et al. (2018), we define the following coupled differential equations to govern the voltage dynamics of pyramidal cells ($\boldsymbol{u}^{\mathrm{P}}_\ell$) and interneurons ($\boldsymbol{u}^{\mathrm{I}}_\ell$) in a network with layers $\ell = 1 \ldots N$:

$$C_{\mathrm{m}}\dot{\boldsymbol{u}}^{\mathrm{P}}_\ell = g_{\mathrm{l}}\left(\boldsymbol{E}_{\mathrm{l}} - \boldsymbol{u}^{\mathrm{P}}_\ell\right) + g^{\mathrm{bas}}\left(\boldsymbol{v}^{\mathrm{bas}}_\ell - \boldsymbol{u}^{\mathrm{P}}_\ell\right) + g^{\mathrm{api}}\left(\boldsymbol{v}^{\mathrm{api}}_\ell + \boldsymbol{\xi}_\ell(t) - \boldsymbol{u}^{\mathrm{P}}_\ell\right) \qquad \forall \ell \neq N, \tag{6.54}$$

$$C_{\mathrm{m}}\dot{\boldsymbol{u}}^{\mathrm{P}}_N = g_{\mathrm{l}}\left(\boldsymbol{E}_{\mathrm{l}} - \boldsymbol{u}^{\mathrm{P}}_N\right) + g^{\mathrm{bas}}\left(\boldsymbol{v}^{\mathrm{bas}}_N - \boldsymbol{u}^{\mathrm{P}}_N\right) + \boldsymbol{i}^{\mathrm{nudge,tgt}}\,, \tag{6.55}$$

$$C_{\mathrm{m}}\dot{\boldsymbol{u}}^{\mathrm{I}}_\ell = g_{\mathrm{l}}\left(\boldsymbol{E}_{\mathrm{l}} - \boldsymbol{u}^{\mathrm{I}}_\ell\right) + g^{\mathrm{den}}\left(\boldsymbol{v}^{\mathrm{den}} - \boldsymbol{u}^{\mathrm{I}}_\ell\right) + \boldsymbol{i}^{\mathrm{nudge,I}}\,. \tag{6.56}$$

Here, $\boldsymbol{\xi}_\ell$ denotes the noise modeled in all hidden layers. Compartment voltages are induced instantaneously by the respective input rates and synaptic weight,

$$\boldsymbol{v}^{\mathrm{bas}}_\ell = \boldsymbol{W}^{\mathrm{PP}}_{\ell,\ell-1}\varphi\left(\breve{\boldsymbol{u}}^{\mathrm{P}}_{\ell-1}\right)\,, \tag{6.57}$$

$$\boldsymbol{v}^{\mathrm{api}}_\ell = \boldsymbol{B}^{\mathrm{PP}}_{\ell,\ell+1}\varphi\left(\breve{\boldsymbol{u}}^{\mathrm{P}}_{\ell+1}\right) + \boldsymbol{B}^{\mathrm{PI}}_{\ell,\ell}\,\varphi\left(\breve{\boldsymbol{u}}^{\mathrm{I}}_\ell\right)\,, \tag{6.58}$$

$$\boldsymbol{v}^{\mathrm{den}}_\ell = \boldsymbol{W}^{\mathrm{IP}}_{\ell,\ell}\varphi\left(\breve{\boldsymbol{u}}^{\mathrm{P}}_\ell\right)\,. \tag{6.59}$$

The nudging currents for the interneurons are $\boldsymbol{i}^{\mathrm{nudge,I}} = g^{\mathrm{nudge,I}}(\breve{\boldsymbol{u}}^{\mathrm{P}}_{\ell+1} - \breve{\boldsymbol{u}}^{\mathrm{I}}_\ell)$, and the output layer pyramidal neurons receive a weak instructive signal via $\boldsymbol{i}^{\mathrm{nudge,tgt}} = g^{\mathrm{nudge,tgt}}(\boldsymbol{u}^{\mathrm{tgt}} - \breve{\boldsymbol{u}}^{\mathrm{P}}_N)$.

In the base microcircuit model of Sacramento et al. (2018) augmented with prospective coding (Haider et al., 2021), synaptic plasticity of forward and lateral weights is defined as

$$\dot{\boldsymbol{W}}^{\text{PP}}_{\ell,\ell-1} = \eta^{\text{fw}}_{\ell}\left[\varphi\left(\breve{\boldsymbol{u}}^{\text{P}}_{\ell}\right) - \varphi\left(\frac{g^{\text{bas}}}{g_{\text{l}}+g^{\text{bas}}+g^{\text{api}}}\boldsymbol{v}^{\text{bas}}_{\ell}\right)\right]\varphi\left(\breve{\boldsymbol{u}}^{\text{P}}_{\ell-1}\right)^{T} \quad \forall \ell \neq N\ , \tag{6.60}$$

$$\dot{\boldsymbol{W}}^{\text{PP}}_{N,N-1} = \eta^{\text{fw}}_{N}\left[\varphi\left(\breve{\boldsymbol{u}}^{\text{P}}_{N}\right) - \varphi\left(\frac{g^{\text{bas}}}{g_{\text{l}}+g^{\text{bas}}}\boldsymbol{v}^{\text{bas}}_{N}\right)\right]\varphi\left(\breve{\boldsymbol{u}}^{\text{P}}_{N-1}\right)^{T}\ , \tag{6.61}$$

$$\dot{\boldsymbol{W}}^{\text{IP}}_{\ell,\ell} = \eta^{\text{IP}}\left[\varphi\left(\breve{\boldsymbol{u}}^{\text{I}}_{\ell}\right) - \varphi\left(\frac{g^{\text{den}}}{g_{\text{l}}+g^{\text{den}}}\boldsymbol{v}^{\text{den}}_{\ell}\right)\right]\varphi\left(\breve{\boldsymbol{u}}^{\text{P}}_{\ell}\right)^{T}\ , \tag{6.62}$$

$$\dot{\boldsymbol{B}}^{\text{PI}}_{\ell,\ell} = \eta^{\text{PI}}_{\ell}\left[-\boldsymbol{v}^{\text{api}}_{\ell}\right]\varphi\left(\breve{\boldsymbol{u}}^{\text{I}}_{\ell}\right)^{T}\ , \tag{6.63}$$

while top-down weights $\boldsymbol{B}^{\text{PP}}_{\ell,\ell+1}$ are fixed.

Before deriving the relevant analytical expressions, we briefly explain how the design of the circuitry leads to well-defined error propagation. In absence of a teaching signal, and if the microcircuit has settled into its self-predicting state, the interneuron activity in each layer represents an exact copy of the pyramidal neurons in the layer above. Interneurons project laterally onto the apical dendrites of pyramidal cells in the same layer; these apical dendrites also receive input from pyramidal cells in the layer above. In the self-predicting state, these activities are subtracted from each other; as they are exactly the same, the inputs cancel, and the apical compartment voltage is zero.

We now introduce a weak nudging signal towards the correct voltage at the output layer. To first order in expansion parameters, the interneurons still represent the pyramidal neurons in the layer above *in absence of a teaching signal*. The activity of the pyramidal cell in the layer above now however additionally contains the error signal. Therefore, the difference in activity calculated at the apical dendrite in a given layer also represents an error. Starting from the penultimate layer, this argument extends successively to the apical compartment voltages in all hidden layers. Consequently, the apical compartments represent errors useful for learning, and these errors are backpropagated.

In order to prove the above statements, we reconsider the dynamics defined by Eqns. (6.54) to (6.56) without noise, and learning rules Eqns. (6.60) to (6.63). Before performing supervised training, the system must settle in a self-predicting state. This is achieved by presenting input sequences while clamping the target voltage to the prospective voltage, $\boldsymbol{u}^{\text{tgt}} = \breve{\boldsymbol{u}}^{\text{P}}_{N}$, and evolving the system while keeping the bottom-up weights $\boldsymbol{W}^{\text{PP}}_{\ell+1,\ell}$ and top-down weights $\boldsymbol{B}^{\text{PP}}_{\ell,\ell+1}$ fixed. The dynamics of the lateral weights, $\dot{\boldsymbol{W}}^{\text{IP}}_{\ell,\ell}$ and $\dot{\boldsymbol{B}}^{\text{PI}}_{\ell,\ell}$, are designed to drive the respective weights to the self-predicting state and are required to work in conjunction.

The lateral connections from interneurons to pyramidal cells $\boldsymbol{B}^{\text{PI}}_{\ell,\ell}$ are driven by gradient descent on the mismatch energy $\|\boldsymbol{v}^{\text{api}}_{\ell}\|^2$ defined by the apical compartment potential. For

fixed top-down synapses, the dynamics of Eqn. (6.63) settle such that $\boldsymbol{v}_\ell^{\text{api}}$ is (approximately) zero for all inputs.

Through the dynamics of Eqn. (6.62), the weights $\boldsymbol{W}_{\ell,\ell}^{\text{IP}}$ are adapted to minimize the difference between the dendritic potential of the interneurons and the voltage in the basal compartment of pyramidal cells in the layer above. This can be seen by expanding the learning rule in $g^{\text{nudge,I}} \ll g_\text{l} + g^{\text{den}}$,

$$\dot{\boldsymbol{W}}_{\ell,\ell}^{\text{IP}} \propto \varphi\left(\breve{\boldsymbol{u}}_\ell^{\text{I}}\right) - \varphi\Big(\frac{g^{\text{den}}}{g_\text{l}+g^{\text{den}}}\boldsymbol{v}_\ell^{\text{den}}\Big) \tag{6.64}$$

$$= \varphi\Big(\frac{g^{\text{den}}\boldsymbol{v}_\ell^{\text{den}} + g^{\text{nudge,I}}\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}}{g_\text{l}+g^{\text{den}}+g^{\text{nudge,I}}}\Big) - \varphi\Big(\frac{g^{\text{den}}}{g_\text{l}+g^{\text{den}}}\boldsymbol{v}_\ell^{\text{den}}\Big) \tag{6.65}$$

$$\approx \varphi'\Big(\frac{g^{\text{den}}}{g_\text{l}+g^{\text{den}}}\boldsymbol{v}_\ell^{\text{den}}\Big)\ \frac{g^{\text{nudge,I}}}{g_\text{l}+g^{\text{den}}}\left[\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}} - \frac{g^{\text{den}}}{g_\text{l}+g^{\text{den}}}\boldsymbol{v}_\ell^{\text{den}}\right], \tag{6.66}$$

where in the first step, we have replaced the prospective interneuron voltage with the potentials which induce it, $\breve{\boldsymbol{u}}_\ell^{\text{I}} = \frac{g^{\text{den}}\boldsymbol{v}_\ell^{\text{den}}+g^{\text{nudge,I}}\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}}{g_\text{l}+g^{\text{den}}+g^{\text{nudge,I}}}$ given by Eqn. (6.56). In the second step, we expand in weak nudging of the interneuron.

In conjunction with the minimization of the apical potential in all layers through $\dot{\boldsymbol{B}}_{\ell,\ell}^{\text{PI}}$, the prospective potential $\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}$ is fully determined by its basal input, $\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}} = \frac{g^{\text{bas}}}{g_\text{l}+g^{\text{bas}}+g^{\text{api}}}\boldsymbol{v}_{\ell+1}^{\text{bas}}$ for $1 \leq \ell < N-1$ and $\breve{\boldsymbol{u}}_N^{\text{P}} = \frac{g^{\text{bas}}}{g_\text{l}+g^{\text{bas}}}\boldsymbol{v}_N^{\text{bas}}$ for the output layer. Therefore, the synapses settle into a state which minimizes the difference between the basal voltage $\boldsymbol{v}_{\ell+1}^{\text{bas}}$ and the interneuron compartment $\boldsymbol{v}_\ell^{\text{den}}$ (up to a factor defined by the conductances) for all input samples.

After the lateral weights have converged, the interneuron potentials are an exact copy of the pyramidal cells in the layer above, $\breve{\boldsymbol{u}}_\ell^{\text{I}} = \breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}$; this can be seen by plugging the steady state solution $g^{\text{den}}\boldsymbol{v}_\ell^{\text{den}} = (g_\text{l}+g^{\text{den}})\,\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}$ into the expression of $\breve{\boldsymbol{u}}_\ell^{\text{I}}$ in terms of its compartmental voltages.

A particularly well-suited self-predicting state is defined by

$$\begin{aligned}\boldsymbol{B}_{\ell,\ell}^{\text{PI}} &= -\boldsymbol{B}_{\ell,\ell+1}^{\text{PP}} \\ \boldsymbol{W}_{\ell,\ell}^{\text{IP}} &= \frac{g^{\text{bas}}}{g^{\text{den}}}\frac{g_\text{l}+g^{\text{den}}}{g_\text{l}+g^{\text{bas}}+g^{\text{api}}}\boldsymbol{W}_{\ell+1,\ell}^{\text{PP}}\end{aligned} \tag{6.67}$$

for hidden layers, and $\boldsymbol{W}_{N-1,N-1}^{\text{IP}} = \frac{g^{\text{bas}}}{g^{\text{den}}}\frac{g_\text{l}+g^{\text{den}}}{g_\text{l}+g^{\text{bas}}}\boldsymbol{W}_{N,N-1}^{\text{PP}}$ for the lateral weights projecting to the interneurons in the final hidden layer. This state has the advantage that the lateral weights form a self-predicting state independent of the input data (general solutions of $\dot{\boldsymbol{W}}_{\ell,\ell}^{\text{IP}} = 0 = \dot{\boldsymbol{B}}_{\ell,\ell}^{\text{PI}}$ do not perform as well in practice, as stimulus switching often requires re-learning of lateral weights before apical compartments represent useful error signal). In the simulations presented in this work, the networks are initialized in this specific self-predicting state.

We now turn on a teaching signal $\boldsymbol{u}^{\text{tgt}}$. The new, nudged prospective state of the output neurons is $\breve{\boldsymbol{u}}_N^{\text{P}} = \frac{g^{\text{bas}}\boldsymbol{v}_N^{\text{bas}} + g^{\text{nudge,tgt}}\boldsymbol{u}^{\text{tgt}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{nudge,tgt}}}$. Inserting this state into Eqn. (6.61) and expanding the somatic potential about the weighted basal input $\hat{\boldsymbol{v}}_N^{\text{bas}}$, we obtain

$$\Delta \boldsymbol{W}_{N,N-1}^{\text{PP}} \approx \varphi'(\hat{\boldsymbol{v}}_N^{\text{bas}}) \cdot \frac{g^{\text{nudge,tgt}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{nudge,tgt}}} \left[\boldsymbol{u}^{\text{tgt}} - \hat{\boldsymbol{v}}_N^{\text{bas}}\right] \left(\boldsymbol{r}_{N-1}^{\text{P}}\right)^T . \tag{6.68}$$

Here, we have defined $\hat{\boldsymbol{v}}_N^{\text{bas}} \coloneqq \frac{g^{\text{bas}}}{g_{\text{l}} + g^{\text{bas}}} \boldsymbol{v}_N^{\text{bas}}$, and we have rewritten the bottom-up input from pyramidal neurons in the penultimate layer as a rate $\boldsymbol{r}_{N-1}^{\text{P}}$. We can now identify the difference between target and bottom-up input as the output layer error, $\boldsymbol{e}_N \coloneqq \boldsymbol{u}^{\text{tgt}} - \hat{\boldsymbol{v}}_N^{\text{bas}}$, and we may thus write

$$\Delta \boldsymbol{W}_{N,N-1}^{\text{PP}} \propto \varphi'(\hat{\boldsymbol{v}}_N^{\text{bas}})\, \boldsymbol{e}_N \left(\boldsymbol{r}_{N-1}^{\text{P}}\right)^T . \tag{6.69}$$

Written in this form, we have demonstrated that the update rule in Eqn. (6.61) implements error minimization on the output layer in the limit of weak nudging. One can regard this as equivalent to training the output layer of a feed-forward network, evaluated at the weighted input $\hat{\boldsymbol{v}}_N^{\text{bas}}$.

We now aim to show that the error $\boldsymbol{e}_N$ is propagated backwards through the network, where the apical compartment voltages represent the local error within each layer. Starting from the self-predicting state, the apical voltages are

$$\begin{aligned}
\boldsymbol{v}_{N-1}^{\text{api}} &= \boldsymbol{B}_{N-1,N}^{\text{PP}} \left[\varphi\left(\breve{\boldsymbol{u}}_N^{\text{P}}\right) - \varphi\left(\breve{\boldsymbol{u}}_{N-1}^{\text{I}}\right)\right] && (6.70)\\
&= \boldsymbol{B}_{N-1,N}^{\text{PP}} \left[\varphi\left(\frac{g^{\text{bas}}\,\boldsymbol{v}_N^{\text{bas}} + g^{\text{nudge,tgt}}\,\boldsymbol{u}^{\text{tgt}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{nudge,tgt}}}\right) - \varphi\left((1-\lambda^{\text{I}})\,\hat{\boldsymbol{v}}_N^{\text{bas}} + \lambda^{\text{I}}\,\frac{g^{\text{bas}}\,\boldsymbol{v}_N^{\text{bas}} + g^{\text{nudge,tgt}}\,\boldsymbol{u}^{\text{tgt}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{nudge,tgt}}}\right)\right]\\
\boldsymbol{v}_{\ell}^{\text{api}} &= \boldsymbol{B}_{\ell,\ell+1}^{\text{PP}} \left[\varphi\left(\breve{\boldsymbol{u}}_{\ell+1}^{\text{P}}\right) - \varphi\left(\breve{\boldsymbol{u}}_{\ell}^{\text{I}}\right)\right] && (6.71)\\
&= \boldsymbol{B}_{\ell,\ell+1}^{\text{PP}} \left[\varphi\left(\hat{\boldsymbol{v}}_{\ell+1}^{\text{bas}} + \lambda^{\text{P}}\,\boldsymbol{v}_{\ell+1}^{\text{api}}\right) - \varphi\left(\hat{\boldsymbol{v}}_{\ell+1}^{\text{bas}} + \lambda^{\text{I}}\,\lambda^{\text{P}}\,\boldsymbol{v}_{\ell+1}^{\text{api}}\right)\right]
\end{aligned}$$

with $\lambda^{\text{I}} \coloneqq \frac{g^{\text{nudge,I}}}{g_{\text{l}} + g^{\text{den}} + g^{\text{nudge,I}}}$, $\lambda^{\text{P}} \coloneqq \frac{g^{\text{api}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}}$, and $\hat{\boldsymbol{v}}_{\ell+1}^{\text{bas}} \coloneqq \frac{g^{\text{bas}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}} \boldsymbol{v}_{\ell+1}^{\text{bas}}$. Let us first focus on the apical voltage in the penultimate layer. We again make use of the assumption of weak nudging, and additionally require that the interneuron is only weakly nudged by the top-down input it receives, $\lambda^{\text{I}} \ll 1$. The apical voltage takes the form

$$\boldsymbol{v}_{N-1}^{\text{api}} \approx \boldsymbol{B}_{N-1,N}^{\text{PP}}\, \varphi'\left(\hat{\boldsymbol{v}}_N^{\text{bas}}\right) \frac{g^{\text{nudge,tgt}}}{g_{\text{l}} + g^{\text{bas}}} \boldsymbol{e}_N . \tag{6.72}$$

In the same fashion, we expand the apical potentials in the layers below. The first order in $\lambda^{\mathrm{P}}\, \boldsymbol{v}_{\ell+1}^{\mathrm{api}}$ and zeroth order in λ^{I} yields

$$\boldsymbol{v}_{\ell}^{\mathrm{api}} \approx \boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}\, \varphi'\left(\hat{\boldsymbol{v}}_{\ell+1}^{\mathrm{bas}}\right) \lambda^{\mathrm{P}}\, \boldsymbol{v}_{\ell+1}^{\mathrm{api}} \,. \tag{6.73}$$

Taken together, these two results show that the apical potentials represent errors which are successively propagated backwards through the network.

Finally, the last missing ingredient is how apical errors are used to update the forward weights. Performing the same expansion in small apical voltages on Eqn. (6.60), we obtain

$$\Delta \boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}} \propto \lambda^{\mathrm{P}} \varphi'(\hat{\boldsymbol{v}}_{\ell}^{\mathrm{bas}})\, \boldsymbol{v}_{\ell}^{\mathrm{api}} \left(\boldsymbol{r}_{\ell-1}^{\mathrm{P}}\right)^{T} . \tag{6.74}$$

We now collect and summarize our findings. Bottom-up weights $\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}$ are updated using the local error, represented by the apical voltage as $\boldsymbol{v}_{\ell}^{\mathrm{api}}$, multiplied with the bottom-up signal $\boldsymbol{r}_{\ell-1}^{\mathrm{P}}$. Eqns. (6.72) and (6.73) show that these errors are backpropagated by multiplying with the derivative $\varphi'(\hat{\boldsymbol{v}}_{\ell+1}^{\mathrm{bas}})$ and feedback weights $\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}$. This learning scheme resembles that of feed-forward networks trained with feedback alignment (for fixed $\boldsymbol{B}^{\mathrm{PP}}$), or backpropagation (if $\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}} = (\boldsymbol{W}_{\ell+1,\ell}^{\mathrm{PP}})^{T}$). One marked difference to a feed-forward network is the emergence of the derivative $\varphi'(\hat{\boldsymbol{v}}_{N}^{\mathrm{bas}})$ in the update rule to all bottom-up weights, see Eqns. (6.69) and (6.72). As we have defined the error $\boldsymbol{e}_N$ on the voltage level, one may expect there to be no such derivative, as one finds in the corresponding case of a feed-forward networks trained with backpropagation with linear activation functions on the output layer. This additional factor signals a fundamental difference in architecture between backpropagation and difference target propagation, of which dendritic cortical microcircuits are an implementation. In difference target propagation, targets are constructed locally from backpropagated rates – i.e. a target potential $\boldsymbol{u}^{\mathrm{tgt}}$ is converted into a rate before it can be passed to a lower layer. In contrast, in backpropagation, top-down signals are given by errors, bypassing the activation function in the upper layer and instead directly transporting potential differences to hidden layers.

We now incorporate this result with PAL. As shown in Eqn. (6.48), using PAL, the top-down weights converge to $\mathbb{E}\left[\boldsymbol{B}_{\ell,\ell+1}\right] \propto \varphi'(\breve{\boldsymbol{u}}_{\ell}^{0}) \left[\boldsymbol{W}_{\ell+1,\ell}\right]^{T} \varphi'(\breve{\boldsymbol{u}}_{\ell+1}^{0}) \;\forall\; \ell$. In the microcircuit model, forward weights are learned as derived in Eqn. (6.74); in summary, we have found that

$$\Delta \boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}} \propto \varphi'(\hat{\boldsymbol{v}}_{\ell}^{\mathrm{bas}}) \Big[\prod_{k=\ell}^{N-1} \boldsymbol{B}_{k,k+1}^{\mathrm{PP}}\, \varphi'(\hat{\boldsymbol{v}}_{k+1}^{\mathrm{bas}}) \Big]\, \boldsymbol{e}_N \left(\boldsymbol{r}_{\ell-1}^{\mathrm{P}}\right)^{T} . \tag{6.75}$$

In the limit of weak nudging and feedback, the noise-free potentials $\breve{\boldsymbol{u}}_{\ell}^{\mathrm{P},0}$ are well approximated by the bottom-up input $\hat{\boldsymbol{v}}_{\ell}^{\mathrm{bas}}$, and our result for top-down weights takes the form $\mathbb{E}\left[\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}\right] \propto \varphi'(\hat{\boldsymbol{v}}_{\ell}^{\mathrm{bas}}) \left[\boldsymbol{W}_{n+1,n}^{\mathrm{PP}}\right]^{T} \varphi'(\hat{\boldsymbol{v}}_{\ell+1}^{\mathrm{bas}}) \;\forall\; \ell$. Plugging this into Eqn. (6.75), we see that the weight updates $\Delta \boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}$ align with those of a feed-forward network trained with back-

propagation up to additional factors of derivatives. Therefore, our algorithm can provide useful error signals which approximately align with backpropagation, improving on random feedback weights.

Local alignment is compatible with approximate Gauss Newton-target propagation

The framework of PAL can easily be extended to propagate Gauss-Newton targets. As shown in Meulemans et al. (2020), this requires the training of feedback connections such that they invert the signal passed from given layer ℓ to the output layer N; i.e. training weights such that the backward mapping Jacobian matrix $\boldsymbol{J}_{g_{\ell,N}}$ is a pseudoinverse of the forward mapping, $\boldsymbol{J}_{g_{\ell,N}} = \left[\boldsymbol{J}_{f_{N,\ell}}\right]^+$. These feedback mappings could be realized by skip connections from the output layer to each hidden layer, while maintaining a layer-wise feed-forward architecture. In contrast to the layer-wise feedback setup defined in the main text (cf. Fig. 6.6), only one interneuron population matching the output layer pyramidal cells would be required here, as the same error signal $\boldsymbol{e}_N$ is backpropagated to all hidden layers.

In analogy to DTP-DRL (Meulemans et al., 2020), we can define a difference-based reconstruction loss,

$$\mathcal{L}_\ell^{\text{diff}} = \|\boldsymbol{B}_{\ell,N}\,\widehat{\boldsymbol{r}}_N - \boldsymbol{\xi}_\ell\|^2 + \alpha\,\|\boldsymbol{B}_\ell\|^2\ . \tag{6.76}$$

Gradient descent on this reconstruction loss yields the update rule

$$\dot{\boldsymbol{B}}_{\ell,N} = -\eta_i^{\text{bw}}\left[\left(\boldsymbol{B}_{\ell,N}\,\widehat{\boldsymbol{r}}_N - \boldsymbol{\xi}_\ell\right)\widehat{\boldsymbol{r}}_N + \alpha\,\boldsymbol{B}_{\ell,N}\right]\ . \tag{6.77}$$

If noise injection and plasticity of top-down weights is phased, i.e. by a schedule that sequentially injects noise only into a given layer ℓ while enabling plasticity of $\boldsymbol{B}_{\ell,N}$, this learning rule minimizes the reconstruction loss of $\boldsymbol{\xi}_\ell$ as it passes to the output layer and back. This can be seen by plugging in the equivalent of Eqn. (6.41) for phased noise,

$$\widehat{\boldsymbol{r}}_N \approx \varphi'(\breve{\boldsymbol{u}}_N^0)\left[\prod_{n=\ell}^{N-1}\boldsymbol{W}_{n+1,n}\varphi'(\breve{\boldsymbol{u}}_n^0)\right]\boldsymbol{\xi}_\ell\ . \tag{6.78}$$

The difference $\boldsymbol{B}_{\ell,N}\,\widehat{\boldsymbol{r}}_N - \boldsymbol{\xi}_\ell$ is minimal if $\boldsymbol{B}_{\ell,N}\,\varphi'(\breve{\boldsymbol{u}}_N^0)$ is equal to $\lambda^{\text{P}}\left[\prod_{n=\ell}^{N-1}\boldsymbol{W}_{n+1,n}^{\text{PP}}\varphi'(\breve{\boldsymbol{u}}_n^0)\right]^+$, thereby aligning the backwards Jacobian with the Moore-Penrose inverse of the forward Jacobian matrix.

As shown in Meulemans et al. (2020), learning backwards weight to minimize such a reconstruction loss produces forward updates closely related to Gauss-Newton optimization,

$$\Delta\boldsymbol{W}_{\ell,\ell-1}^{\text{PP}} \propto \left[\boldsymbol{J}_{f_{N,\ell}}\right]^+\boldsymbol{e}_N\,(\boldsymbol{r}_{\ell-1}^{\text{P}})^T\ , \tag{6.79}$$

where $\boldsymbol{J}_{f_{N,\ell}}$ denotes the Jacobian matrix mapping potentials in layer ℓ to the output layer N.
While error propagation using the Moore-Penrose inverse has been shown to perform well on simple classification tasks (Lee et al., 2015; Bartunov et al., 2018; Meulemans et al., 2020; Podlaski and Machens, 2020), it is currently not known how such a difference reconstruction loss could be implemented while training all top-down weights simultaneously.

6.4.6 Supplementary Information B: Simulation of PAL

Microcircuit models

Below we address several implementation details:
For all simulations, we set the resting potential to $\boldsymbol{E}_1 = 0$.
Figure 6.7 (Phaseless backwards weight alignment): In order to compare the backprojections with those in an ANN trained with BP (right column), we perform several steps. After each epoch of top-down weight training, we instantiate a teacher model with the architecture as the respective 'student' microcircuit model. To this model, we pass the input sequence and record the teacher output as a target signal. We now provide the newly acquired input/-target pairs to the student model. As in all other simulations, these pairs are presented for $T_{\text{pres}} = 100\,dt$, and we disable noise during this evaluation. It is also necessary to set the lateral weights to the self-predicting state defined by Eqn. (6.67) in order to obtain a measurable error signal in layers below the last hidden layer. We record the potential weight updates defined by the backprojections, but do not apply them. Next, we feed the same input/target sequence into an ANN with weights set to $\boldsymbol{W}^{\text{PP}}_{\ell,\ell-1}$ and layer size and activation as defined in Table 6.1. For this ANN, we calculate the output layer error as the difference between target and voltage (linear activation on output layer). The weight update given in this ANN is then compared to the recorded update for the microcircuit model.
Figure 6.8 (Teacher-student setup): We initialize the teacher with weights $\boldsymbol{W}^{\text{PP}}_{2,1} = \boldsymbol{W}^{\text{PP}}_{1,0} = 2$. For the student models with feedback alignment, the same parameters as for PAL are used, but with $\eta^{\text{bw}} = \eta^{\text{PI}} = 0$, no noise injection, and without the low-pass filter on $\Delta\boldsymbol{W}^{\text{PP}}_{\ell,\ell-1}$. For the BP reference model, we additionally set $\boldsymbol{B}^{\text{PP}}_{\ell-1,\ell} = [\boldsymbol{W}^{\text{PP}}_{\ell,\ell-1}]^T$ and $\boldsymbol{B}^{\text{PI}}_{\ell-1,\ell} = -\boldsymbol{B}^{\text{PP}}_{\ell-1,\ell}$ after every update.
Figure 6.8 (Classification tasks): For the Yin-Yang task, we used datasets of size 6000 for training, 900 for validation, and 900 for testing. For MNIST digit classification, sets were 50000 for training, 10000 for validation, and 10000 for testing. In either case, FA runs use the same parameters as PAL, but with fixed $\boldsymbol{B}^{\text{PP}}_{\ell,\ell+1}$ and $\boldsymbol{B}^{\text{PI}}_{\ell,\ell}$ and no noise injection. The reference network is an ANN trained with BP using a Cross-Entropy loss and ADAM with default parameters of PyTorch. In this case, 10 seeds were trained.

Algorithm 1: Dendritic cortical microcircuits with prospective coding and PAL

Input: Data stream encoded as rate vector $\boldsymbol{r}_0^{\mathrm{P}}[t]$, target voltage vector $\boldsymbol{u}^{\mathrm{tgt}}[t]$
Parameters: Network with layers 1 to N; effective neuron time constants $\tau_\ell^{\mathrm{eff,P}}, \tau_\ell^{\mathrm{eff,I}}$

Update instantaneous rates and compartment potentials
  for ℓ *in* $range(1, N)$ **do**
    $\breve{\boldsymbol{u}}_\ell^{\mathrm{P}}[t] \leftarrow \boldsymbol{u}_\ell^{\mathrm{P}}[t-dt] + \frac{\tau_\ell^{\mathrm{eff,P}}}{dt}\,(\boldsymbol{u}_\ell^{\mathrm{P}}[t] - \boldsymbol{u}_\ell^{\mathrm{P}}[t-dt])$
    $\breve{\boldsymbol{u}}^{\mathrm{I}}[t] \leftarrow \boldsymbol{u}^{\mathrm{I}}[t-dt] + \frac{\tau_\ell^{\mathrm{eff,I}}}{dt}\,(\boldsymbol{u}^{\mathrm{I}}[t] - \boldsymbol{u}^{\mathrm{I}}[t-dt])$
    $\boldsymbol{r}_\ell^{\mathrm{P}}[t] \leftarrow \varphi\big(\breve{\boldsymbol{u}}_\ell^{\mathrm{P}}[t]\big)$
    $\boldsymbol{r}_\ell^{\mathrm{I}}[t] \leftarrow \varphi\big(\breve{\boldsymbol{u}}_\ell^{\mathrm{I}}[t]\big)$
  for ℓ *in* $range(1, N)$ **do**
    $\boldsymbol{v}_\ell^{\mathrm{bas}}[t] \leftarrow \boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}[t]\;\boldsymbol{r}_{\ell-1}^{\mathrm{P}}[t]$
    $\boldsymbol{v}_\ell^{\mathrm{api}}[t] \leftarrow \boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}[t]\;\boldsymbol{r}_{\ell+1}^{\mathrm{P}}[t] + \boldsymbol{B}_{\ell,\ell}^{\mathrm{PI}}[t]\;\boldsymbol{r}_\ell^{\mathrm{I}}[t]$
    $\boldsymbol{v}_\ell^{\mathrm{den}}[t] \leftarrow \boldsymbol{W}_{\ell,\ell}^{\mathrm{IP}}[t]\;\boldsymbol{r}_\ell^{\mathrm{P}}[t]$

Update high-pass filtered rates
  for ℓ *in* $range(1, N)$ **do**
    $\widehat{\boldsymbol{r}}_\ell^{\mathrm{P}}[t] \leftarrow \widehat{\boldsymbol{r}}_\ell^{\mathrm{P}}[t-dt] + \boldsymbol{r}_\ell^{\mathrm{P}}[t] - \boldsymbol{r}_\ell^{\mathrm{P}}[t-dt] - \frac{dt}{\tau_{\mathrm{hp}}}\widehat{\boldsymbol{r}}_\ell^{\mathrm{P}}[t-dt]$

Update noise vectors
  for ℓ *in* $range(1, N-1)$ **do**
    $\boldsymbol{\mu}_\ell \sim \mathcal{N}(0, \mathbb{I})$
    $\boldsymbol{\xi}_\ell[t] \leftarrow \boldsymbol{\xi}_\ell[t-dt] + \frac{1}{\tau_\xi}\big(\sqrt{\tau_\xi\,dt}\,\sigma_\ell\,\boldsymbol{\mu}_\ell - dt\;\boldsymbol{\xi}_\ell[t-dt]\big)$

Update somatic potentials with noise injection
  for ℓ *in* $range(1, N-1)$ **do**
    $\boldsymbol{u}_\ell^{\mathrm{eff,I}}[t] \leftarrow \tau_\ell^{\mathrm{eff,I}}\left\{g^{\mathrm{den}}\,\boldsymbol{v}_\ell^{\mathrm{den}}[t] + g^{\mathrm{nudge,I}}\,\breve{\boldsymbol{u}}_\ell^{\mathrm{P}}[t]\right\}$
    $\Delta\boldsymbol{u}_\ell^{\mathrm{I}}[t] \leftarrow \frac{dt}{\tau_\ell^{\mathrm{eff,I}}}\left\{\boldsymbol{u}_\ell^{\mathrm{eff,I}}[t] - \boldsymbol{u}_\ell^{\mathrm{I}}[t]\right\}$
    $\boldsymbol{u}_\ell^{\mathrm{eff,P}}[t] \leftarrow \tau_\ell^{\mathrm{eff,P}}\left\{g^{\mathrm{bas}}\,\boldsymbol{v}_\ell^{\mathrm{bas}}[t] + g^{\mathrm{api}}\,(\boldsymbol{v}_\ell^{\mathrm{api}}[t] + \boldsymbol{\xi}_\ell[t])\right\}$
    $\Delta\boldsymbol{u}_\ell^{\mathrm{P}}[t] \leftarrow \frac{dt}{\tau_\ell^{\mathrm{eff,P}}}\left\{\boldsymbol{u}_\ell^{\mathrm{eff,P}}[t] - \boldsymbol{u}_\ell^{\mathrm{P}}[t]\right\}$
  $\boldsymbol{u}_N^{\mathrm{eff,P}}[t] \leftarrow \tau_N^{\mathrm{eff,P}}\left\{g^{\mathrm{bas}}\,\boldsymbol{v}_N^{\mathrm{bas}}[t] + g^{\mathrm{nudge,tgt}}\,\boldsymbol{u}^{\mathrm{tgt}}[t]\right\}$
  $\Delta\boldsymbol{u}_N^{\mathrm{P}}[t] \leftarrow \frac{dt}{\tau_N^{\mathrm{eff,P}}}\left\{\boldsymbol{u}_N^{\mathrm{eff,P}}[t] - \boldsymbol{u}_N^{\mathrm{P}}[t]\right\}$
  apply voltage updates: $\boldsymbol{u}_\ell^{\mathrm{P}} + \Delta\boldsymbol{u}_\ell^{\mathrm{P}}, \boldsymbol{u}_\ell^{\mathrm{I}} + \Delta\boldsymbol{u}_\ell^{\mathrm{I}}\;\;\forall\,\ell$

Update weights, incl. low-pass filtering feedforward weight updates
  for ℓ *in* $range(1, N-1)$ **do**
    $\Delta\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}} \leftarrow dt\;\eta_\ell^{\mathrm{fw}}\left\{\boldsymbol{r}_\ell^{\mathrm{P}}[t] - \varphi\big(\frac{g^{\mathrm{bas}}}{g_{\mathrm{l}}+g^{\mathrm{bas}}+g^{\mathrm{api}}}\boldsymbol{v}_\ell^{\mathrm{bas}}[t-dt]\big)\right\}\;\boldsymbol{r}_{\ell-1}^{\mathrm{P}}[t-dt]$
    $\Delta\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}} \leftarrow dt\;\eta_\ell^{\mathrm{bw}}\left\{\boldsymbol{\xi}_\ell[t]\,\widehat{\boldsymbol{r}}_{\ell+1}^{\mathrm{P}}[t-dt] - \alpha_\ell\,\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}[t]\right\}$
    $\Delta\boldsymbol{B}_{\ell,\ell}^{\mathrm{PI}} \leftarrow -dt\;\eta_\ell^{\mathrm{PI}}\,\boldsymbol{v}_\ell^{\mathrm{api}}[t-dt]\,\boldsymbol{r}_\ell^{\mathrm{I}}[t-dt]$
    $\Delta\boldsymbol{W}_{\ell,\ell}^{\mathrm{IP}} \leftarrow dt\;\eta^{\mathrm{IP}}\left\{\boldsymbol{r}_\ell^{\mathrm{I}}[t] - \varphi\big(\frac{g^{\mathrm{den}}}{g_{\mathrm{l}}+g^{\mathrm{den}}}\boldsymbol{v}_\ell^{\mathrm{den}}[t-dt]\big)\right\}\;\boldsymbol{r}_\ell^{\mathrm{P}}[t-dt]$
  $\Delta\boldsymbol{W}_{N,N-1}^{\mathrm{PP}} \leftarrow dt\;\eta_\ell^{\mathrm{fw}}\left\{\boldsymbol{r}_N^{\mathrm{P}}[t] - \varphi\big(\frac{g^{\mathrm{bas}}}{g_{\mathrm{l}}+g^{\mathrm{bas}}}\boldsymbol{v}_N^{\mathrm{bas}}[t-dt]\big)\right\}\;\boldsymbol{r}_{N-1}^{\mathrm{P}}[t-dt]$
  for ℓ *in* $range(1, N)$ **do**
    $\overline{\Delta\boldsymbol{W}}_{\ell,\ell-1}^{\mathrm{PP}} \leftarrow \overline{\Delta\boldsymbol{W}}_{\ell,\ell-1}^{\mathrm{PP}}[t-dt] + \frac{dt}{\tau_{\mathrm{lo}}}\big(\Delta\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}}[t-dt] - \overline{\Delta\boldsymbol{W}}_{N,N-1}^{\mathrm{PP}}[t-dt]\big)$
  apply weight updates:
  $\boldsymbol{W}_{\ell,\ell-1}^{\mathrm{PP}} + \overline{\Delta\boldsymbol{W}}_{\ell,\ell-1}^{\mathrm{PP}}, \boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}} + \Delta\boldsymbol{B}_{\ell,\ell+1}^{\mathrm{PP}}, \boldsymbol{B}_{\ell,\ell}^{\mathrm{PI}} + \Delta\boldsymbol{B}_{\ell,\ell}^{\mathrm{PI}}, \boldsymbol{W}_{\ell,\ell}^{\mathrm{IP}} + \Delta\boldsymbol{W}_{\ell,\ell}^{\mathrm{IP}}\;\;\forall\,\ell$

Table 6.1: Parameters for microcircuit model simulations.

	Fig. 6.7 (a+b+c)	Fig. 6.7 (d+e+f)	Fig. 6.8 (a+b)	Fig. 6.8 (c+d)	Fig. 6.8 (e+f)
dt [ms]	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
T_{pres} [ms]	1	1	1	1	1
τ_{hp} [ms]	0.1	0.1	0.1	0.1	0.1
τ_{lo} [ms]	—[1]	—[1]	10^2	10^2	10^2
τ_ξ [ms]	0.1	0.1	0.1	0.1	0.1
noise scale $\sigma_\ell \,\forall\, \ell$	5×10^{-2}	5×10^{-2}	10^{-2}	10^{-2}	10^{-2}
regularizer α	10^{-5}	10^{-5}	10^{-6}	10^{-6}	10^{-6}
g_l [ms^{-1}]	0.03	0.03	0.03	0.03	0.03
g^{bas} [ms^{-1}]	0.1	0.1	0.1	0.1	0.1
g^{api} [ms^{-1}]	0.06	0.06	0.06	0.06	0.06
g^{den} [ms^{-1}]	0.1	0.1	0.1	0.1	0.1
$g^{\text{nudge, I}}$ [ms^{-1}]	0.06	0.06	0.06	0.06	0.06
$g^{\text{nudge,tgt}}$ [ms^{-1}]	0.06	0.06	0.06	0.06	0.06
input	$\mathcal{U}[0, 1]$	$\mathcal{U}[0, 1]$	$\mathcal{U}[0, 1]$	Yin-Yang	MNIST
dataset size	100	100	100	6000	50000
epochs	100	500	5000	400	100
network size	[5-20-10-20-5]	[5-20-10-20-5]	[1-1-1]	[4-30-3]	[784-100-10]
activation	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}$	$\frac{1}{1+e^{-x}}$
η^{fw} [ms^{-1}]	$0, 0, 0, 0$	$0, 0, 0, 0$	$2, 0.5$	$50, 0.01$	$1.0, 5 \times 10^{-3}$
η^{bw} [ms^{-1}]	$50, 50, 50$	$20, 20, 20$	20	0.5	0.2
η^{IP} [ms^{-1}]	$0, 0, 0$	$0, 0, 0$	10	0.05	0.02
η^{PI} [ms^{-1}]	$5, 5, 5$	$0.5, 0.5, 0.5$	0.5	0.02	0.02
weight init $\boldsymbol{W}^{\text{PP}}_{\ell,\ell-1}$	$\mathcal{U}[-1, 1]$	$\mathcal{U}[-5, 5]$	$\mathcal{U}[-1, 0]$	$\mathcal{U}[-0.1, 0.1]$	$\mathcal{U}[-0.1, 0.1]$
weight init $\boldsymbol{B}^{\text{PP}}_{\ell-1,\ell}$	$\mathcal{U}[-1, 1]$	$\mathcal{U}[-5, 5]$	$\mathcal{U}[-1, 0]$	$\mathcal{U}[-1, 1]$	$\mathcal{U}[-1, 1]$

[1] No forward weight updates applied.

Efficient credit assignment in deep networks

We detail the parameters for the general LI-model simulation below:

Table 6.2: Simulation parameters for the autoencoder simulation with the general LI-model.

	Fig. 6.9
dt [ms]	10^{-1}
T_{pres} [ms]	10
τ_{hp} [ms]	1
τ_{lo} [ms]	10^3
τ_ξ [ms]	1
noise scale $\sigma_\ell \,\forall\, \ell$	10^{-2}
regularizer α	10^{-4}
input	MNIST
epochs	10
batch size	32
network size	[784-200-2-200-784]
activation	$\tanh$, linear, $\tanh$, linear
η^{fw} [ms^{-1}]	$8 \times 10^{-4}, 8 \times 10^{-4}, 8 \times 10^{-3}, 8 \times 10^{-3}$
η^{bw} [ms^{-1}]	$0.8, 0.8, 0.8$
weight init $\boldsymbol{W}_{\ell,\ell-1}$	$\mathcal{N}(0, 0.05)$
weight init $\boldsymbol{B}_{\ell-1,\ell}$	$\mathcal{N}(0, 0.05)$
bias init $\boldsymbol{b}_\ell$	$\mathcal{N}(0, 0.05)$

Errors on the output layer are defined as $\boldsymbol{e}_N = \beta\,(\boldsymbol{u}^{\text{tgt}} - \breve{\boldsymbol{u}}_N)$, and we have simulated with $\beta = 0.1$. Weight updates were calculated by taking the mean $\Delta \boldsymbol{W}_{\ell,\ell-1}$ over batches of size 32; we stress that these weight updates are applied at all time steps dt. The dataset sizes for the MNIST autoencoder task are 50000 for training, 10000 for validation, and 10000 for testing.

6.5 Event-based communication in dendritic microcircuits

The original model of the dendritic microcircuits is based on rate-based communication between neurons and rate-based learning rules. In this form it is not amenable to an implementation on spiking neuromorphic platforms. This section outlines two potential approaches to bridge this gap.

6.5.1 Point neuron microcircuits

This first approach is centered around the idea of recreating the functional principles of the dendritic microcircuit out of components that are available on a broad range of current spiking neuromorphic platforms. In particular, we identify the LIF neuron as one of the most popular and widely used neuron models in the neuromorphic field (Pfeil et al., 2013; Furber et al., 2014; Akopyan et al., 2015; Davies et al., 2018; Billaudelle et al., 2022)[15] and use it as a basis for our alternative dendritic microcircuit. This entails two crucial changes: On the one hand we move from a rate-based neuron model to a spiking one and on the other we make the step from compartmental to point neurons.

Author contributions

The project idea for a recreation of the dendritic microcircuit's functional principle with LIF neurons was developed in a collaborative effort by Laura Kriener (LK), Sebastian Billaudelle, Benjamin Cramer, Matteo Cartiglia and Mihai Petrovici (MAP) during the *2019 Capo Caccia Neuromorphic Engineering Workshop*. The workshop's results were extended and refined in first simulations by LK under the joint supervision of MAP, Jakob Jordan and Walter Senn. The project was continued by a collaboration of Ben von Hünerbein (BvH) and Ismael Jaras (IJ) under the supervision of LK and MAP. BvH and IJ simulated and refined the model. Afterwards, IJ implemented the model on the BrainScaleS-2 hardware. The results of this collaboration were presented as an abstract and poster at the *6th HBP Student Conference on Interdisciplinary Brain Research* under the title *Towards fully embedded biologically inspired deep learning on neuromorphic hardware*[16]. BvH, IJ and LK share the first-authorship.

Approximating rate-based neurons with LIF neurons

The dynamics and synaptic communication mechanisms of LIF neurons are radically different from the previously used rate-based leaky-integrator neuron models. Nevertheless, LIF neurons can be used to approximate the input-output relations of the neurons described in Sacramento et al. (2018). In this section we provide an intuition on why this is the case.

[15] This includes also other popular neuron models such as the adaptive exponential leaky integrate-and-fire (AdEx) neuron that can be reduced back to the LIF neuron through parameter choice.

[16] (von Hünerbein et al., 2022)

If LIF neurons are provided with spike input of a constant input rate r_{in}, we can see in Fig. 6.11 a that after a short initial phase an equilibrium is reached. The average synaptic current in that equilibrium $\bar{I}_{\text{syn}}$ is proportional to the product of input weight w_{in} and input rate r_{in} (see Fig. 6.11 c, d):

$$\bar{I}_{\text{syn}} \propto w_{\text{in}} r_{\text{in}}$$

This shows the similarity between the rate-based neuron with $\dot{u} \propto w_{\text{in}} r_{\text{in}} + \text{leak} + \text{bias}$ and the LIF neuron where we have $\dot{u} \propto I_{\text{syn}} + I_{\text{leak}}$. Note that for the latter we have absorbed the bias current into the leak term. Furthermore, by focusing on the free membrane potential u_{free}, which is the membrane voltage that the LIF neuron would have if its spiking threshold were infinitely high, we see another parallel to rate-based leaky-integrator neurons. The (free) membrane potential is, like the voltage in the rate-based case, a low-pass filter of the synaptic input. Additionally, the steady state of the free membrane $\bar{u}_{\text{free}}$, after subtraction of the leak potential, is proportional to $w_{\text{in}} r_{\text{in}}$, just like the steady-state membrane voltage in a rate-based neuron after subtraction of bias and leak (see Fig. 6.11 e). Finally, in Fig. 6.11 f we see the shape of the activation function of the LIF neuron. The time constants, in particular the refractory period τ_{ref}, and the leak potential control the shape of the activation function. τ_{ref} determines the maximum firing rate of the neuron with $r_{\max} = \frac{1}{\tau_{\text{ref}}}$. With an appropriately chosen bias, i.e. leak potential, and very short τ_{ref}, such that $r_{\max}$ is far above the usual firing rates of the network, the LIF neuron's activation function can be used to approximate a rectified linear unit (ReLU).

Note that the activation function shown here is not perfectly monotonous. This is due to aliasing effects caused by the regular input. The impact of this can be weakened by introducing jitter on the regular spike times and longer synaptic time constants. If a neuron receives multiple inputs of different frequencies, the effect is typically also less pronounced. Additionally, since the output spike trains that neurons with multiple inputs produce are not perfectly regular, the issue is less relevant for higher layers of hierarchical networks. Finally, we could also choose to replace the regularly-spaced input with random spike trains of the same average firing rate, for example Poisson spike trains. This however, adds a significant amount of additional noise to the system, which, in practice, had a detrimental effect on learning.

In the following we will make use of the correspondence between the LIF neurons and rate-based neurons and use the same notation as for the original dendritic microcircuits.

Approximating multi-compartment microcircuit mechanisms with point neurons

The original microcircuit model features three different types of neurons: the three-compartment pyramidal neurons in the hidden layers of the network, the two-compartment interneurons and the two-compartment pyramidal neurons in the top layer. While the dendritic compartments collect the synaptic inputs, the soma integrates the dendritic signals

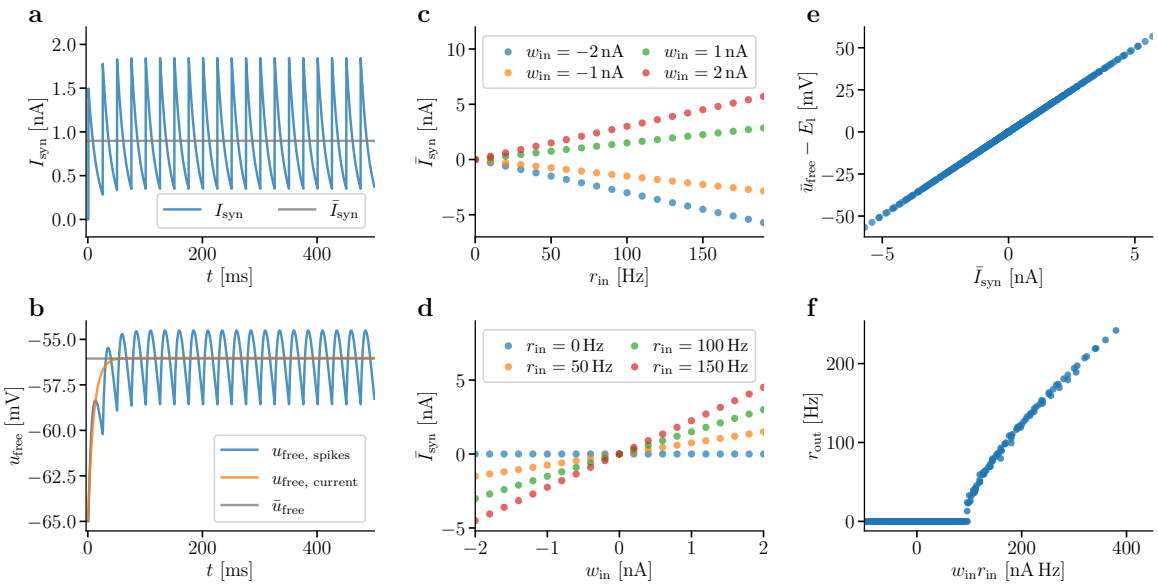

Figure 6.11: Illustration of how a LIF neuron can be used to approximate a rate-based neuron. **a**: Synaptic current of an LIF with current-based exponential synaptic kernels that receives regular spiking input with the synaptic weight w_{in} and rate r_{in} (blue). The average synaptic current (excluding a short initial phase) $\bar{I}_{\text{syn}}$ is marked in gray. **b**: Free membrane voltage of the same neuron as in (a) is drawn in blue. The free membrane voltage of another neuron which does not receive spiking input but the average current $\bar{I}_{\text{syn}}$ is shown in orange. The average free membrane potential after the initial phase is marked in gray. **c** and **d**: Illustration of the proportionality of $\bar{I}_{\text{syn}}$ to the input rate r_{in} and the input weight w_{in} and therefore by extension to their product. **e**: Illustration of the proportionality of the steady state free membrane $\bar{u}_{\text{free}} - E_l$ to the average input current $\bar{I}_{\text{syn}}$. **f**: Activation function of the LIF neuron as output firing rate r_{out} over the input strength $r_{\text{in}} w_{\text{in}}$. The parameters used for these simulations can be found in Table B.2.

and produces the neuron's output. Fig. 6.12 shows the transition of the original microcircuit to a point neuron version. By making the step from multi-compartment neurons to point neurons, the number of dynamic variables per neuron decreases: instead of multiple dendritic voltages plus the somatic voltage there is only the one membrane voltage. It is no longer possible to store multiple "signals" (e.g. bottom-up input in the basal dendrite, top-down input in the apical dendrite and the linear combination of both in the soma) within the same neuron. Therefore, in the point neuron microcircuit model a single multi-compartment pyramidal neuron is replaced by two LIF neurons (see Fig. 6.12). We call one of them the "error neuron" which holds the signal formerly stored in the apical compartment and the other keeps the name pyramidal neuron, as it holds the same information as the soma of the multi-compartment pyramidal neuron. Note that we do not need the information of the basal dendrites to be stored separately, so there is no "basal neuron",

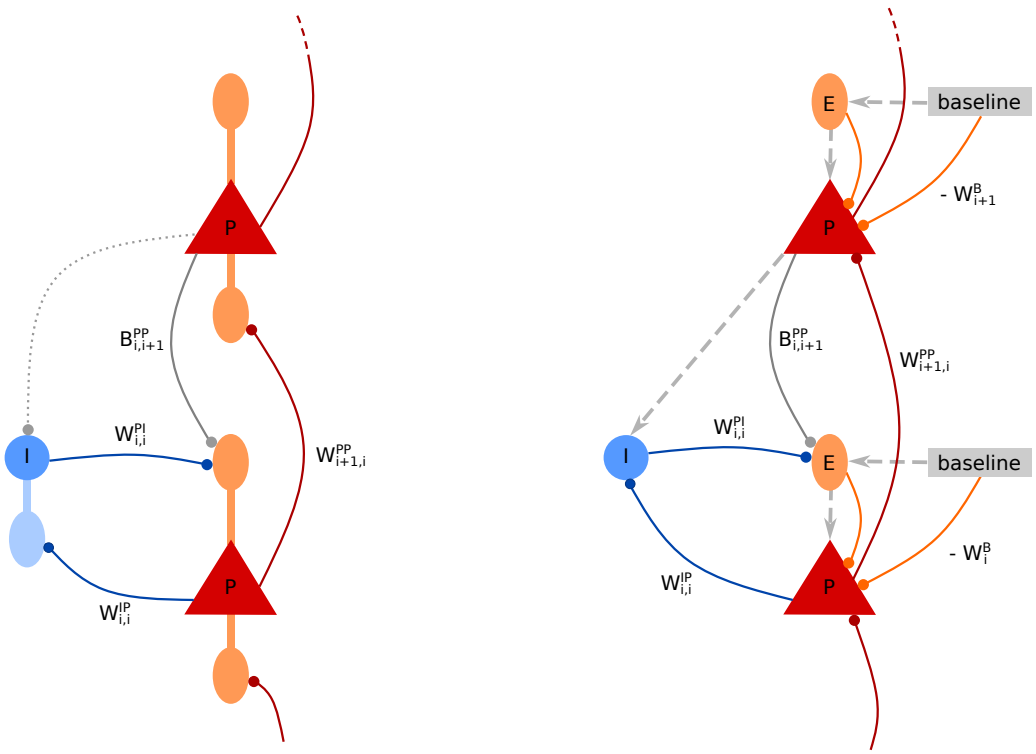

Figure 6.12: Comparison of the schematic of the original microcircuit and the point neuron microcircuit. Left: Schematic drawing of the original dendritic microcircuit as in Sacramento et al. (2018). **Right:** Adaptation to LIF point neurons. The two-compartment interneuron on the left (blue) is replaced by a single LIF neuron while the three-compartment pyramidal neuron is replaced by an "error neuron" (orange) and a pyramidal (red) LIF neuron. Neurons firing constantly at baseline frequency are introduced (gray box) as well as an additional connection from error and baseline neuron to the pyramidal neuron. Dashed gray arrows denote one-to-one target signals used for plasticity.

but instead the bottom-up input is sent directly into the pyramidal neuron. Similarly, the multi-compartment interneuron is replaced by a single LIF interneuron.
In the dendritic microcircuit model the apical dendrites store a local error signal for the pyramidal neuron and nudge its somatic voltage, via conductive coupling, towards a local target. Since the error neuron and the pyramidal neuron are separate cells now, the nudging has to happen via a synaptic connection (orange connections in Fig. 6.12). This introduces the complication that, while the apical voltages and their impact on the somatic voltage can be both positive and negative, the impact of the synaptic connection can not, because firing rates of neurons are strictly nonnegative. We solve this issue by increasing the leak potential E_l of the LIF neurons such that they fire with a baseline frequency of r^B. By connecting the error neuron as well as a "baseline neuron" (which always fires with the baseline frequency) with the weight $+W^B$ and $-W^B$ to the pyramidal neuron, the effective signal

nudging the pyramidal neuron is $W^{\mathrm{B}}\left(r^{\mathrm{E}}-r^{\mathrm{B}}\right)$ which can have positive as well as negative values. Note that each error neuron, while receiving input from all above-layer pyramidal neurons and interneurons, is only connected to one of the pyramidal neurons in the same layer. For more details on the nudging via a synaptic connection see Appendix A.3.1.

The switch to point neurons also impacts the learning mechanism. The weight updates in the original microcircuit models are based on firing rates. Since we have shown previously that firing rates of LIF neurons correctly approximate firing rates of rate-based neurons, we can do the same here. In practice, this means that we present each input sample as regular input spike trains for a certain duration T and record all spiking activity in the network. Then, after the time T, we calculate all firing rates of the neurons as $r=\frac{n_{\text{spikes}}}{T}$ and perform one weight update based on these firing rates.

The weight update in Urbanczik-Senn plasticity rule (Urbanczik and Senn, 2014) in Eqn. (6.7) is proportional to the difference between a signal which is nudged towards a target $\varphi(u)$ and a non-nudged signal $\varphi(v^*)$. Both signals are present in the same neuron, in the soma and the basal dendrite respectively. This is not directly realizable with a point neuron. Therefore, we have to replace one of the two neuron-internal signals with an external one: The former — the nudged signal — is replaced by an external target, while the latter is represented by the point neuron's output. Other neurons in the network serve as targets (dashed arrows in the Fig. 6.12).

For the interneurons, the target is most easily determined: As each interneuron is supposed to mimic a pyramidal neuron in the layer above, the firing rate of that pyramidal neuron is the target for the interneuron

$$\begin{aligned}\dot{\boldsymbol{W}}_{\ell,\ell}^{\mathrm{IP}} &= \eta^{\mathrm{IP}}\ \left[\boldsymbol{r}_{\ell+1}^{\mathrm{P}}-\boldsymbol{r}_{\ell}^{\mathrm{I}}\right]\ \boldsymbol{r}_{\ell}^{\mathrm{P}} && (6.80)\\ &\approx \eta^{\mathrm{IP}}\ \left[\varphi\left(\boldsymbol{u}_{\ell+1}^{\mathrm{P}}\right)-\varphi\left(\boldsymbol{u}_{\ell}^{\mathrm{I}}\right)\right]\ \varphi\left(\boldsymbol{u}_{\ell}^{\mathrm{P}}\right)\ . && (6.81)\end{aligned}$$

The weights from the interneurons to the error neurons (called $\boldsymbol{W}^{\mathrm{PI}}$ to show the analogy to the original microcircuits) only need to be learned during the setup of the self-predicting state. In that setting the network receives random input and no target and since no target is provided to the network, all internal error signal should be zero. For the point neuron microcircuit this results in the baseline firing rate as target for the error neurons and the weight update rule

$$\begin{aligned}\dot{\boldsymbol{W}}_{\ell,\ell}^{\mathrm{PI}} &= \eta^{\mathrm{PI}}\ \left[\boldsymbol{r}^{\mathrm{B}}-\boldsymbol{r}_{\ell}^{\mathrm{E}}\right]\ \boldsymbol{r}_{\ell}^{\mathrm{I}} && (6.82)\\ &\approx \eta^{\mathrm{PI}}\ \left[\varphi(\mathbf{0})-\varphi\left(\boldsymbol{u}_{\ell}^{\mathrm{E}}\right)\right]\ \varphi\left(\boldsymbol{u}_{\ell}^{\mathrm{I}}\right)\ . && (6.83)\end{aligned}$$

For the feedforward weights the target rate is the sum of the pyramidal firing rate and the error signal in the layer

$$\dot{\boldsymbol{W}}^{\mathrm{PP}}_{\ell,\ell-1} = \eta^{\mathrm{PP}} \left[\left(\boldsymbol{r}^{\mathrm{P}}_{\ell} + \boldsymbol{r}^{\mathrm{E}}_{\ell} - \boldsymbol{r}^{\mathrm{B}}\right) - \boldsymbol{r}^{\mathrm{P}}_{\ell}\right] \boldsymbol{r}^{\mathrm{P}}_{\ell-1} \tag{6.84}$$

$$= \eta^{\mathrm{PP}} \left[\boldsymbol{r}^{\mathrm{E}}_{\ell} - \boldsymbol{r}^{\mathrm{B}}\right] \boldsymbol{r}^{\mathrm{P}}_{\ell-1} \tag{6.85}$$

$$\approx \eta^{\mathrm{PP}} \left[\varphi\left(\boldsymbol{u}^{\mathrm{E}}_{\ell}\right) - \varphi(\boldsymbol{0})\right] \varphi\left(\boldsymbol{u}^{\mathrm{P}}_{\ell-1}\right) . \tag{6.86}$$

A detailed description on how the error propagation mechanisms of this setup correspond to the mechanisms in the original microcircuit is given in Appendix A.3.2.

Modifications for practical implementability

After having discussed the conceptual changes for going from rate-based multi-compartment neurons to spiking point neurons, we here highlight further implementation details that facilitate learning in a neuromorphic but also in a simulation setting.
So far we have not specified how exactly the error signal is given into the error neurons of the top layer. There are multiple possibilities, for example providing the error neurons with a current proportional to the difference between the target and the network's output. The most easily realizable solution in practice is to have a "target neuron" fire with the target rate and connect it with a constant weight W^{tgt} to the error neuron while at the same time connecting the pyramidal neuron of the output layer to the error neuron with the weight $-W^{\mathrm{tgt}}$. This effectively provides the error neuron with the signal $e_N = W^{\mathrm{tgt}}\ (r^{\mathrm{tgt}} - r^{\mathrm{P}}_N)$. The approach of providing only the target signal to the error neuron directly and locally calculating the error, i.e. the difference between the target and the network's output, is more practically feasible. This is due to the fact that the target is already known before the start of the simulation and therefore this signal can be provided via a predetermined spike-source (both in simulation and on a neuromorphic chip), while an error signal would need to be calculated and generated while the experiment is already running, which is not possible in all simulators or on all neuromorphic platforms.
The nudging of the pyramidal neurons in the point neuron model is performed via synaptic connections in contrast to a conductive coupling in the original model. Therefore, the nudging signal is a firing rate instead of a voltage. If the activation function of the error neuron is highly non-linear, the nudging signal is strongly distorted, which can affect learning performance. To prevent this, we tune the error neurons to have an activation function which strongly resembles a ReLU with a bias. With that we can assume for most cases that the nudging signal transported by the rate is proportional to the nudging signal that would be there in case of a conductive coupling. While a ReLU-like activation function is desirable for the error neurons, for all other neurons it is beneficial to have an upper bound on the firing rate. This allows for example to avoid communication bandwidth limitations on neuromorphic platforms and is easily achieved by increasing the refractory period of the LIF neurons.

On neuromorphic platforms neuronal and synaptic resources are often limited and resource-efficient implementations can be crucial. A subtle difference between the original and the point neuron microcircuit allows us to reduce the required synapse count for the point neuron microcircuit. In the original microcircuit model the nudging of the soma via the apical compartments in the pyramidal neurons serves two purposes: Firstly, it creates the difference between the two signals used in the weight update of the feedforward weights, the soma which is nudged towards a target and the basal dendrite (Eqn. (6.60)). Secondly, via the nudging of the pyramidal neurons and the backward connections the error signal is transported to lower layers in the network. In contrast to that, in the point neuron microcircuit the nudging serves only one purpose: Since the weight update for the feedforward weights is changed (Eqn. (6.85)) and no longer relies on the nudging, only the transportation of the error signal downwards remains. This is only necessary in all layers above the lowest hidden layer. Below the lowest hidden layer lies only the input layer which does not need the error signal since no learning takes place there. Therefore, in the point neuron microcircuit, the nudging of the pyramidal neurons in the lowest hidden layer serves no purpose and can be omitted, especially if the available connectivity is limited on a neuromorphic platform.
The task of the interneuron is to match the activity of the pyramidal neuron in the layer above, as reflected in the learning rule for W^{IP} Eqn. (6.80). This matching works as intended in the learning of the self-predicting state, however during the learning of the task, the pyramidal neurons are nudged by the error neurons. Therefore, the target for the interneuron is a nudged version of the pyramidal activities and it learns to mimic that. This directly counteracts the error propagation mechanism which relies on the difference between the nudged pyramidal neuron and the interneuron. If both are the same, there is no error signal in the layer below. This phenomenon of the interneuron learning to mimic the pyramidal activity *including the nudging* is also present in the original microcircuit model, however it is less pronounced and can be controlled through careful tuning of the learning rate ratios between η^{IP} and η^{PP}. For the point neuron microcircuits the required level of fine-tuning on the learning rates is higher and practically unfeasible. Alternatively, the learning rule for the interneuron can be changed for the learning of the task: We assume the learning of the task starts in the self-predicting state. At this point, the value of W^{IP} is such that the interneuron produces the same activity as the pyramidal neuron *without nudging*. If pyramidal and interneuron have the same activation function (which we assume here) then this relationship can be kept if whenever an update is applied to W^{PP} the same weight update is applied to W^{IP}. We realize this by employing the learning rule for W^{PP} (Eqn. (6.85)) also for W^{IP}

$$\dot{\boldsymbol{W}}^{\mathrm{IP}}_{\ell,\ell} = \eta^{\mathrm{IP}} \left[\boldsymbol{r}^{\mathrm{E}}_{\ell} - \boldsymbol{r}^{\mathrm{B}}\right] \boldsymbol{r}^{\mathrm{P}}_{\ell}\,. \tag{6.87}$$

The final point we address here is mostly relevant for mixed-signal or analog platforms. Due to device mismatch no two neurons or synaptic circuits behave exactly identical even with

identical parametrization. The microcircuit model however is based on the idea that if two neurons receive the same input via synapses with the same weight, they will produce the same output (e.g. interneurons matching pyramidal neurons). On neuromorphic hardware this is not necessarily the case. Therefore, the difference of a target rate and the postsynaptic rate which is part of the learning rules will, in most cases, not be zero even for an optimal tuning of the synaptic weights. To prevent weight updates caused solely by this effect, a thresholding mechanism can be introduced in the learning rules: If the difference between target and output is smaller than a certain threshold θ, then no weight update is applied

$$\Delta W = \begin{cases} 0 & \text{if } |r^{\text{tgt}} - r^{\text{out}}| < \theta \\ \eta \, (r^{\text{tgt}} - r^{\text{out}}) \, r^{\text{in}} & \text{otherwise.} \end{cases} \tag{6.88}$$

Preliminary results

Here we show preliminary results on how the introduced point neuron microcircuit functions and reproduces the key functionalities of the original model.

In Fig. 6.13 we see simulation results for a single microcircuit (as illustrated in Fig. 6.12) which first sets itself up in a self-predicting state while receiving random input and then learns to match a predefined input-output relationship. During the learning of the self-predicting state the same mechanisms as in the original model apply: The synaptic weights W^{PI} and W^{IP} are adapted such that the interneuron mimics the upper layer pyramidal neuron and the firing rates of the error neurons decay to the baseline, as no external target is present, and therefore no error signal should propagate through the network (Fig. 6.13 a – c). Once the self-predicting state is reached, the microcircuit is taught to reproduce fixed and predefined input-output pairs (Fig. 6.13 d – f). During learning the difference between the output of the pyramidal neuron in the upper layer and the desired output shrinks and correspondingly, the error signals present in the network, which are represented as the firing rates of the error neurons, decrease. Once the network's output matches the target the error neurons fire at their baseline frequency.

The same experiment was repeated on the mixed-signal neuromorphic platform BrainScaleS-2 (Fig. 6.14). While we see that the general mechanisms during the learning of the self-predicting state are the same, we also see that the firing rates in the hardware emulation are much more noisy compared to the software simulation. These fluctuations in the firing rates for constant inputs are caused by a combination of several effects such as thermal noise on the membrane voltages, intermittent drifts or drops in supply voltages due to increased load, jitter in the spike timing during communication as well as potential spike-loss if communication bandwidths are reached during phases of high spike counts. Additionally, fixed-pattern noise causes each neuron to be slightly different and therefore to have, among other effects, a slightly different baseline firing rate. Calibration mechanisms were employed to reduce the impact of the fixed-pattern noise, but it was not possible to

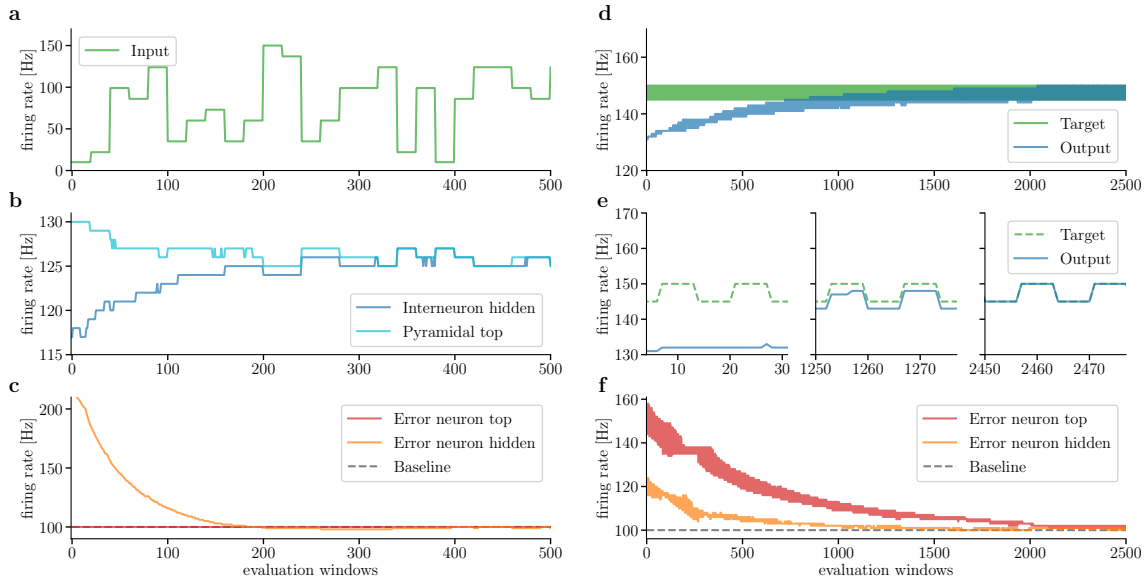

Figure 6.13: Functional principles of the point neuron microcircuits in simulation. This figure was adapted from the poster corresponding to (von Hünerbein et al., 2022). The simulations were performed using the PyNN simulator (Davison et al., 2009), the parameters can be found in Table B.3. Left: Learning of the self-predicting state. **a:** Input firing rates into a point neuron microcircuit for learning the self-predicting state. The x-axis is given in evaluation windows which mark the presentation of one input sample. After each evaluation window the firing rates of the neurons in the microcircuits are determined via the spike counts and the weights are updated. **b:** Firing rates of the interneuron and the pyramidal neuron in the upper layer. The interneuron learns to match the firing rate of the pyramidal neuron. **c:** Firing rates of the error neurons during the learning of the self-predicting state. Since no target is present, the error neuron in the upper layer fires with its baseline frequency from the start. The rate of the lower error neuron decreases to baseline once the self-predicting state is reached. Right: Learning of the task. **d:** Comparison of the pyramidal firing rate and the target rate while the microcircuit learns to match the provided target. **e:** Zoom-in on the data shown in (d) during different stages of training (left: early, center: middle, right:late). **f:** Firing rates of the error neurons in the upper and hidden layer during the learning of the task. The errors decrease towards baseline when the output matches the target more closely.

remove it entirely. Therefore, the baseline firing rate for each error neuron must be measured before the start of the experiment and then the measured, instead of the ideal value, needs to be used in the weight updates (Fig. 6.14 c, f). In spite of the challenges introduced by noisy firing rates as well as fixed-patter noise on neural and synaptic circuits, the microcircuit is both able to successfully learn the self-predicting state and reproduce the desired input-output relationship.

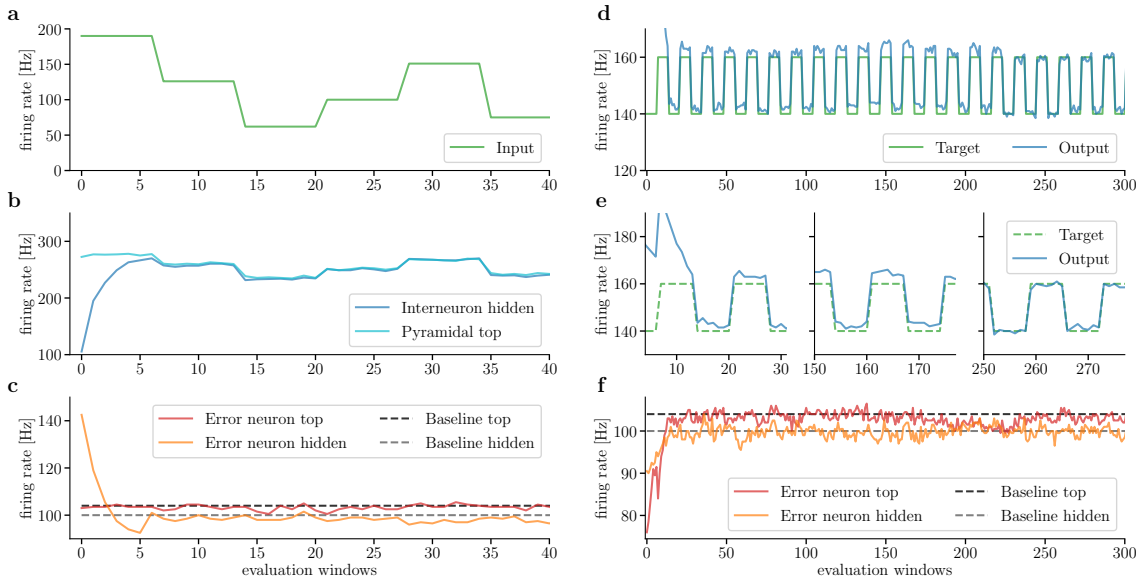

Figure 6.14: Emulation of the point neuron microcircuit on the BrainScaleS-2 hardware. This figure was adapted from the poster corresponding to (von Hünerbein et al., 2022). The parameters for this experiment can be found in Table B.4. All quantities recorded on the hardware were scaled to match the biological time domain. Left: Learning of the self-predicting state. **a:** Input firing rates into a point neuron microcircuit for learning the self-predicting state. **b:** Firing rates of the interneuron and the pyramidal neuron in the upper layer. The interneuron learns to match the firing rate of the pyramidal neuron, however a perfect match is not achieved due to slightly noisy firing rates. **c:** Firing rates of the error neurons during the learning of the self-predicting state. Since no target is present, the error neuron in the upper layer fires with its baseline frequency from the start. The rate of the lower error neuron decreases to baseline once the self-predicting state is reached. Note that the two baselines are slightly different as no two neurons on the BrainScaleS chip are perfectly identical. Right: Learning of the task. **d:** Comparison of the pyramidal firing rate and the target rate while the microcircuit learns to match the provided target. **e:** Zoom-in on the data shown in (d) during different stages of training (left: early, center: middle, right:late). **f:** Firing rates of the error neurons in the upper and hidden layer during the learning of the task.

Challenges and drawbacks

In spite of the successful demonstration of the point neuron microcircuit's general functionality (Fig. 6.13 and Fig. 6.14) in both simulation and neuromorphic emulation, its applicability for larger and more complex tasks is limited for multiple reasons: Due to the mechanism of determining firing rates via a spike count which is accumulated over a significant time window (typically around 1 s) software simulations in particular but also hardware emulations take a long time. For every weight update the network is simulated/emulated for the duration of one window, then the simulation/emulation is interrupted for the calculation of the weight update. This interruption is necessary as neither the currently used

simulator PyNN (Davison et al., 2009) nor our current implementation on BrainScaleS-2 is able to perform the required weight updates at runtime. In particular, for larger networks in simulation the combination of long simulations and the interruptions of the simulations for every weight update lead to prohibitively long experiment durations.
The long simulation/emulation durations have the additional knock-on effect of making parameter tuning (e.g. tuning of learning rates and weight initializations) very time-consuming up to the point of rendering fine-tuning unfeasible. This is problematic as all forms of the microcircuit model are quite sensitive to the parameter configurations especially the ratios of different learning rates. In addition to that the learning rate ratios are highly dependent on network size and the specific task, which makes it difficult to transfer parameters obtained for a smaller network (which can be simulated in reasonable time) to a larger network with a different task.
While the accelerated emulation speed of the BrainScaleS-2 neuromorphic hardware at least partially alleviates the effects of long experiment durations, the different forms of noise as described above provide challenges for the learning and error transport mechanisms. The microcircuit relies on different neurons matching each other's activity closely and the activities of different neurons cancelling each other out at a shared postsynaptic partner. In a noise-free simulation this is easily achieved given the correct parameters and synaptic weights, but on a mixed-signal neuromorphic platform this is much more difficult. Especially fixed-pattern noise and the quantized and limited weight resolution prevent the exact matching of neuronal activities. Therefore, the errors transported through the network are noisy and only approximate the true errors that would lead to an optimal learning of the task. The easier the task (such as the one shown previously), the less the noisy errors hamper learning. However, for a more complex task that requires the network to learn highly tuned feedforward weights, this effect becomes detrimental.
Finally, as discussed in Appendix A.3.2, the point neuron microcircuit only approximately implements the error backpropagation mechanism. It might be the case that for more difficult tasks which require a very precise error signal the approximations made by the point neuron microcircuit are not good enough. While there have been indications for this during attempts to learn a more difficult task, the effect is hard to pinpoint exactly and has proven difficult to clearly demonstrate.

6.5.2 Event-based approximation of rates

In this section we outline an alternative idea on how the rate-based dendritic microcircuits could be realized on a spiking system. The approach is centered around what is called "spikes with payloads"[17] which is a feature of increasing popularity among neuromorphic platforms currently under development. In contrast to biology on neuromorphic systems a spike signal is, from a technical perspective, often not a 1 bit signal, but typically carries

[17] Spikes with payloads are sometimes also referred to as graded spikes.

additional information (e.g. the identity of the presynaptic neuron) which is necessary to route the spike signal to the correct postsynaptic partners.

The infrastructure for routing signals that carry more than a single bit of information therefore already exists and it is in principle straightforward to let the signal carry some more bits of information, which we call the payload[18]. Generally speaking, this feature is easier to implement on digital neuromorphic platforms as they offer more flexibility. A popular example for this is the second generation of the Loihi chip which is expected to support spikes with payloads (IntelLabs, 2021). On other platforms like e.g. SpiNNaker spike payloads can in principle be realized but are not the intended mode of operation and can therefore not be accessed directly via the high-level user interface[19]. Even though the discussion about the inclusion of spike payloads is much more prominent around digital neuromorphic platforms, the event-handling as well as the synapse drivers and synapse circuits on the mixed-signal BrainScaleS-2 HICANN-X platform could in principle also handle them (Billaudelle, 2022, Chapter 3.2)[20].

In the following we will assume that our target neuromorphic platform supports spike payloads and processes the neuron and synapse dynamics using discrete time steps (i.e. we focus on digital platforms). Since we just outline a general concept here, we will neglect potential hardware effects such as limited precision or fixed-point arithmetic for now. Given this, there is an obvious way of implementing a rate-based model on such a platform: If each neuron at every time step emits an event[21] which carries the neuron's current instantaneous firing rate as payload, we achieve the same setup as in a simulation with discrete time steps on conventional hardware. This is, however, not optimal, as the communication architectures of spiking neuromorphic chips are optimized for temporally sparse communication. By letting each neuron spike at each time step we would most likely hit bandwidth restrictions and destroy potential efficiency gains. The next two sections will discuss two potential ideas on how this can be alleviated by sparsifying the events produced by each neuron in time.

Events at regularly spaced time intervals

In the previously described scenario every neuron sends out events with an inter-event time interval of $\Delta t = \mathrm{d}t$, where $\mathrm{d}t$ is the time step of the simulation. The amount of events can be reduced by increasing the time interval Δt between events for every neuron. The

[18] Note that while there is some biological evidence for spikes carrying more than one bit of information (e.g. bursts or dendritic spikes), this feature is mainly motivated by the technological possibility rather than by the biological example.

[19] Personal communication with Oliver Rhodes, September 2019.

[20] Note that even though parts of the infrastructure could handle spike payloads, it is not the intended use of the chip and e.g. the neuron circuits or the high-level user interface are not designed for it.

[21] From now on we will call a spike with payload an event.

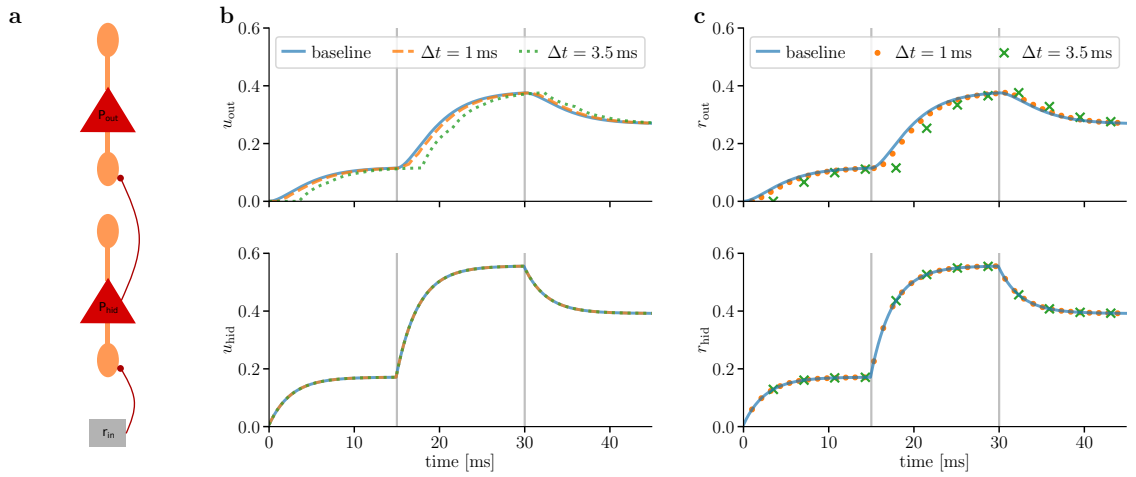

Figure 6.15: Dynamics of pyramidal neurons with regular event-based communication. a: Chain of two pyramidal neurons. The lower pyramidal neuron receives the input r_{in} from a source outputting an instantaneous firing rate (this source is not simulated in the event-based fashion). The points in time when r_{in} changes are marked with vertical gray lines in (b) and (c). **b:** Somatic voltages of the hidden (bottom) and output (top) pyramidal neuron for different Δt (orange and green) used to produce the pyramidal output events. Additionally, the result for a normal rate-based simulation is shown as baseline (blue). The lower pyramidal neuron receives only input from the source which does not use the event-based scheme, therefore the dynamics for all settings of Δt are the same. The upper pyramidal neuron receives input from the lower pyramidal neuron which produces its output in event-based fashion, therefore the pyramidal voltages differ depending on Δt. **c:** Output events of the hidden (bottom) and output (top) pyramidal neuron indicated at the time points of their emission (orange and green) in comparison with the rate-based baseline simulation. The simulation parameters for this figure can be found in Table B.5.

presynaptic instantaneous firing rate received by the postsynaptic neuron then is

$$r_{\text{pre}}(t) = \begin{cases} p_{\text{event}} & \text{if event arrives in this time step} \\ p_{\text{last event}} & \text{if no event arrives} \end{cases} \tag{6.89}$$

where p is the payload of an arriving event. Colloquially speaking, if no event arrives with new information, the neuron "assumes that nothing has changed" compared to the last time step and uses the old information for the calculation of its dynamics.

Figure 6.15 illustrates the resulting dynamics for a simple chain of pyramidal neurons using this regular event-based output scheme. We see that the smaller Δt is chosen, the closer the resulting voltages and output rates match the baseline of a rate-based simulation. This is intuitive because, as discussed before, a rate-based simulation with discrete time steps $\mathrm{d}t$ is the same as an event-based simulation with $\Delta t = \mathrm{d}t$. We have purposefully chosen one $\Delta t = 3.5\,\text{ms}$ such that the presentation time of the input is not a multiple of Δt, to

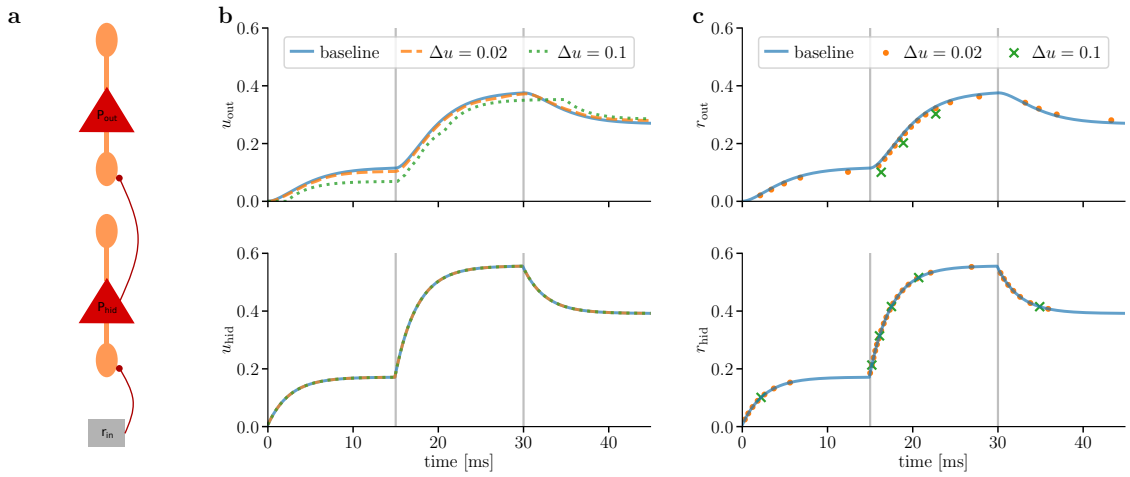

Figure 6.16: Dynamics of pyramidal neurons with voltage triggered event-based communication. **a:** Same experimental setup as in Fig. 6.15. The only difference is the event trigger mechanism of the pyramidal neurons. **b:** Somatic voltage of the hidden (bottom) and output (top) pyramidal neuron for different Δu (orange and green) used to produce the pyramidal output events. The rate-based simulation is shown as baseline (blue). **c:** Output events of the hidden (bottom) and output (top) pyramidal neuron indicated at the time points of their emission (orange and green) in comparison with the rate-based baseline simulation. The simulation parameters for this figure can be found in Table B.5.

illustrate that in this case we induce a delay in the information propagation: As the events are rather sparse in time, information about a change in input takes a noticeable amount of time to reach the top pyramidal neuron in Fig. 6.15 c.

While this method of sending events at regular time intervals decreases the overall number of events throughout a simulation, it can still cause bandwidth issues. If, in the most straightforward implementation, Δt is equal for all neurons in a network, they will all emit their events in the same time step. This means that there is a peak load on the event routing mechanism for this time step while there is no load at all during the others. We can circumvent this by varying Δt for the neurons or by keeping the same Δt for all neurons but adding varying timing offsets. The latter is preferable, as it allows us to choose one specific Δt which offers the right trade-off between event sparsity and faithful reproduction of the original rate-based dynamics.

Events triggered by voltage changes

Alternatively, the sending of an output event can also be triggered by a large enough change in the neuron's voltage. In this case an output event carrying the instantaneous rate as payload is sent if the difference between the current voltage and the voltage at the time of the last event is large enough $|u(t) - u(t_{\text{last event}})| \geq \Delta u$.

Figure 6.16 shows a repetition of the experiment in Fig. 6.15 with this voltage-based event trigger mechanism. We see that the smaller the threshold Δu, the closer the dynamics match the rate-based baseline. In contrast to Fig. 6.15 c, where the events are separated by equal distances in time (i.e. on the x-axis), here the output events are approximately equidistant in the output rate (i.e. on the y-axis)[22]. Therefore, this mechanism automatically alleviates the synchronization of events that occurs in the regular event-based scenario.
As a neuron only produces an output event if its voltage changes enough, the threshold for triggering the output must be chosen carefully. Otherwise, a small change in input, which leads only to a small change in the membrane voltage, might not trigger an output event and therefore this information will not travel further in the network. We expect the choice of an appropriate Δu to depend on parameters such as the input encoding of the task or the network size. To alleviate this effect, hybrids of the two event trigger mechanisms might also be feasible. The voltage-triggered mechanism with a high Δu could for example be accompanied by a regular trigger with a rather high Δt. This would prevent small changes from not being transmitted at all, while also allowing to quickly transmit the, presumably more important, larger changes.

Inclusion of the LE mechanism

Both methods discussed above have advantages and disadvantages and depending on the specific neuromorphic platform either might be more suitable. However, if we include the LE mechanism into the simulated dynamics, a clear favorite emerges. Figure 6.17 repeats the experiments shown in Figs. 6.15 and 6.16 with the LE mechanism included in the neuron dynamics. We see that, while the outcome for the regularly timed events is similar to before, the number of events per input sample for the voltage triggered mechanism is reduced to one[23]. This is due to the fact that the prospective pyramidal voltages react instantaneously to a change in the input and since there is only one input change per input sample, there is also only one pyramidal voltage change causing an event. This is clearly advantageous since it on the one hand greatly reduces the overall number of events necessary and on the other hand also ensures rapid information propagation due to the instantaneity. However, it also reintroduces a synchronization within the network which, as for the regularly timed events, can lead to high load peaks on the communication network which can cause potential bandwidth issues[24].
Note, that these results are specific to our chosen form of inputs, step-wise constant r_{in}. While this setup is common for e.g. static image classification, the advantage of the voltage triggered mechanism is less obvious for continuously changing inputs.

[22]Note that this is only the case for ReLU or linear activation functions.

[23]Due to the inclusion of the LE mechanism, the output rate of the neuron no longer is a function of u but of $\breve{u}$. We have therefore also defined the trigger of the output events on the prospective voltage $|\breve{u}(t) - \breve{u}(t_{\text{last event}})| \geq \Delta u$.

[24]Note here that the effect is slightly less pronounced as for the regular mechanism. This is due to the fact that for the regular mechanism all neurons with the same Δt spike at the same time. Here however, only neurons in the same layer can spike simultaneously, since even with LE information needs one time step per layer to travel.

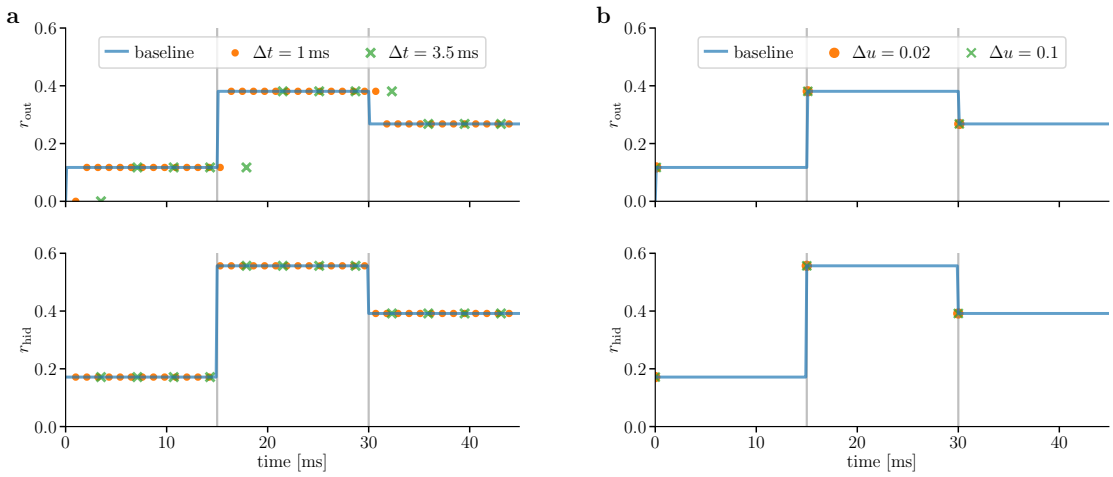

Figure 6.17: Dynamics of event-based pyramidal neuron with LE. a: Same setup as in regular event-based setup of Fig. 6.15 but with the LE mechanism in the pyramidal neurons. Bottom: Input rate into the lower pyramidal neuron. Middle and Top: Output events of the pyramidal neuron indicated at the time points of their emission (orange and green) in comparison with the rate-based baseline simulation. **b:** Same setup as in voltage-triggered event-based setup of Fig. 6.16 but with the LE mechanism in the pyramidal neurons. As the LE mechanism causes instantaneously changing (prospective) voltages, they only change once per input change and therefore only one event per input is necessary. The simulation parameters for this figure can be found in Table B.5.

Chapter 7

Discussion and Outlook

The research presented in this thesis is located at the intersection between the fields of deep learning and neuromorphic computing. While deep learning has produced revolutionary results in many areas of machine intelligence (Krizhevsky et al., 2012a; Brown et al., 2020; OpenAI, 2022a), the field faces increasing challenges due to its ever-growing demand for computational power (Schwartz et al., 2020; Thompson et al., 2020) and the corresponding rise in energy consumption. In contrast to the conventional computing architectures that currently back most deep learning models, the field of neuromorphic engineering aims at developing novel computing hardware that is more closely inspired by the mechanisms in the brain (Schuman et al., 2017). By mimicking functional principles of the brain, neuromorphic systems hope to inherit the energy efficiency of their biological archetype.

The work presented in this thesis is motivated by the prospect of combining the powerful training capabilities for neural networks developed in the field of deep learning with the efficient computing offered by neuromorphic architectures and implementations. By leveraging their inherent parallelism as well as the efficiency of sparse spiking communication, this approach promises a more energy-efficient computing strategy for neural networks than classical von Neumann architectures (Roy et al., 2019).

However, although deep learning and neuromorphic computing are, at their roots, both inspired by the brain, the research fields have evolved largely independently of each other and are not directly compatible. In particular, the error backpropagation algorithm, the central algorithm for training artificial neural networks (ANNs), can in its original form not operate on spiking neural networks (SNNs) like they are found on neuromorphic devices. Additionally, it is at odds with several biological principles (see Section 2.4.3) that are, to a varying degree, also reflected in neuromorphic architectures.

To tackle this incompatibility of error backpropagation with neuronal and neuromorphic systems, this thesis explored two different approaches: On the one hand we formulated a novel method of performing error backpropagation on the spike times of leaky integrate-and-fire (LIF) neurons (Chapter 5), one of the most commonly used neuron models on neuromorphic systems. On the other hand we started out from a biologically plausible approx-

imation of error backpropagation and modified it to make it more amenable to a neuromorphic implementation (Chapter 6). During the development and neuromorphic deployment of the above-mentioned algorithms we observed a lack of suitable datasets for testing and prototyping scenarios and developed the Yin-Yang dataset to fill this gap (Chapter 4).
In the following, we will discuss the results of each chapter separatly. In Section 7.1 we start with the Yin-Yang dataset, as it was a prerequisite for parts of the results presented in the following sections. It is followed in Sections 7.2 and 7.3 by the discussions of our method for error backpropagation of first spike times of LIF neurons and our adaptations of the dendritic microcircuits for neuromorphic implementation. Both of which include outlined ideas for project extensions in Section 7.2.1 and Section 7.3.1 respectively. We have explored both a bottom-up or "device-up" (Chapter 5) and a top-down/"algorithm-down" approach (Chapter 6) for implementing error backpropagation on neuromorphic systems in this thesis. In Section 7.4 we conclude by contrasting the outcome of the two approaches and discussing their advantages and drawbacks.

7.1 The Yin-Yang dataset

The Yin-Yang dataset introduced in Chapter 4 was developed for algorithmic and neuromorphic prototyping. To be suited for these scenarios, we require a dataset to be difficult or sensitive enough, such that it is able to expose potential flaws of an algorithmic or implementation. At the same time, it also must be deployable on neuromorphic platforms in their prototype stages, which often feature relatively few neurons and might have limited input-output interfaces. Finally, simulations of networks trained on the dataset must complete within reasonable times to be amenable for iterative algorithmic development or debugging.
The Yin-Yang dataset fulfills the difficulty requirements, as shown in Section 4.3, while requiring only a network of between 20 and 30 neurons to be solved and having only 4 input dimensions. This is small enough to be deployable on a typical neuromorphic prototype (Moradi and Indiveri, 2013; Schemmel et al., 2017; Frenkel et al., 2018; Nair and Indiveri, 2019; Billaudelle et al., 2020). Being tractable also for small networks and by comprising only few input samples — an order of magnitude less than the popular MNIST dataset (LeCun et al., 1998) — comparatively short simulation times.
The most important feature of the Yin-Yang dataset however, is the fact that it allows for a better performance evaluation of competing models or neuromorphic implementations. In contrast to the XOR problem and the classification of MNIST, the two tasks most commonly used in testing and prototyping, allows to more clearly differentiate and compare the performance of competing models: XOR, as there are only four different inputs, only has a small discrete set of resulting test accuracies. The MNIST dataset has a continuous range of potential output accuracies, but even a linear classifier, arguably the most basic model available, can achieve a test accuracy of around 92 % (LeCun et al., 1998). Therefore, the

range over which models of different quality are distributed is small, hampering a meaningful comparison. This issue is exemplified by our results in Section 6.4, where we present results on MNIST and the Yin-Yang dataset in a direct comparison. While both show the same general trends, the differences between the compared models are more clearly visible on the Yin-Yang dataset. This is due to the fact that the results of the different models are spread over a wider accuracy range. But not only do the MNIST results show the desired effects less clearly, they additionally took a factor of 10 times longer to produce.
Finally, for the study of error backpropagation, the Yin-Yang dataset allows us to clearly distinguish a network that receives proper error signals in the lower layers and a network that does not. This is, again, in contrasted by classification of the MNIST dataset, where the typically used network sizes are large enough that, even with no error signals propagating back to the lower layers, a high accuracy can be reached due to the kernel trick (Scholkopf, 2001). The Yin-Yang dataset avoids this, as it can be solved with small enough networks so that the kernel trick is less effective.
Since its publication, the Yin-Yang dataset has already found significant usage: Results from the dataset are directly featured in the following publications Göltz et al. (2021); Wunderlich and Pehle (2021); Max et al. (2022); Müller et al. (2022); Lee et al. (2022). Additionally, the dataset was used during the development phase of the following projects but was not featured as final result in the paper Haider et al. (2021); Cramer et al. (2022)[1]. Finally, the relatively fast training and evaluation times make the dataset suitable for demonstrations and tutorials such as the one included in the BrainScaleS-2 demo set (Electronic Vision(s), 2022, Tutorial 7).
So far the Yin-Yang dataset is not widely know, and we have accompanied results on it with results on MNIST in our publications (Göltz et al., 2021; Max et al., 2022). Since however the Yin-Yang dataset has clear advantages compared to the commonly used benchmark tasks, we expect it to establish itself in due time as its own widely-known and accepted reference in the field of neuromorphics.

7.2 Error backpropagation with first-spike times of LIF neurons

Developing a form of error backpropagation that is compatible with spiking neurons has been an active field of research in recent years (Mostafa, 2017; Neftci et al., 2019). In Chapter 5 we presented an algorithm for the exact optimization of first-spike times of LIF neurons with a time constant ratio of either $\tau_\mathrm{m} = \tau_\mathrm{s}$ or $\tau_\mathrm{m} = 2\tau_\mathrm{s}$. The resulting update rules for the synaptic weights can be applied in a variety of scenarios, including different network architectures or information coding schemes (see Section 5.2). When applied to a layered feedforward network topology, the plasticity rules result in an exact form of spike-based error backpropagation. We chose a time-to-first-spike (TTFS) encoding for both the input

[1]Even though Cramer et al. (2022) does not include the results obtained with the Yin-Yang dataset, the PhD theses of both main authors (Billaudelle (2022, Chapter 6.6), Cramer (2021, Chapter 5.3)) do.

and output of the network. TTFS is an appealing coding scheme because of its inherent speed and temporal sparseness. In the case of static image classification a sample can be classified with at most one spike per neuron in the network and typically within 1 to 2 τ_{m} after the stimulus onset.
While it was clear that this efficient and fast coding scheme would pair well with the accelerated mixed-signal neuromorphic platform BrainScaleS-2, at first glance the requirement of precise time constant ratios appeared at odds with the analog nature of the neuron and synapse circuits (Schemmel et al., 2022). However, simulation studies on the robustness to fixed-pattern noise showed that, while the derivations in theory required precise time constants, in practice the requirements are significantly softer. This is mainly due to our approach of basing the derivative calculation on the actual output of a noisy network instead of the theoretical spike times (see Section 5.2.3). We note here that our robustness studies benefitted greatly from the comparatively fast training times on the Yin-Yang dataset and would not have been practically feasible to the same extent if for example the MNIST dataset was the only dataset available.
The suitability of our algorithm to neuromorphic deployment was not only investigated using the robustness studies in simulations, but also clearly confirmed by our implementation on the BrainScaleS-2 chip. The combination of an accelerated mixed-signal hardware platform with our TTFS coding scheme resulted in a competitive classification rate of 20 kHz and an energy consumption of 8.4 µJ per sample. This compares favorably with other neuromorphic implementations as shown in Table 5.1 and Supplementary Information, Table SI.F3. Keeping in mind that the BrainScaleS-2 hardware platform is a general research platform with a large variety of features, we would expect that a platform specifically tailored to our approach would further increase the energy efficiency and speed.

7.2.1 Future work

There are two main areas where the work presented by Göltz et al. (2021) can be extended. On the one hand the in-the-loop training approach could be replaced by an on-chip training and on the other hand the algorithm could be extended to not only optimize the first spike of a neuron but also the following ones.

From in-the-loop training to on-chip plasticity

So far the training of our network on the BrainScaleS-2 chip is performed in an in-the-loop fashion: the synaptic weight updates are calculated off-chip on a host computer. The calculations are based on the spike times that were recorded from the chip and sent to the host. Once the updates are calculated, the neural network on the chip is reconfigured with the updated synaptic weights and the next inputs are sent to the chip. This method of in-the-loop plasticity has the advantage of imposing very few requirements on the neuromorphic chip. The only infrastructure required (besides the capability of accommodating networks

of LIF neurons with configurable time constants) is the possibility to record all spiking activity and send it off-chip to a host computer.

The main disadvantage of the in-the-loop approach is its speed. Alternating between calculations performed on the host and the hardware runs introduces a significant overhead and slows the training process. Learning on chip and using the host computer only to provide the input samples but not for plasticity, cuts out the communication overhead and is expected to significantly speed up training.

On-chip plasticity, especially on mixed-signal hardware, is however more difficult to realize than in-the-loop training. Different neuromorphic platforms provide varying levels of support for on-chip plasticity, but generally speaking at least the following infrastructure needs to be available: The weight updates need to be calculated by circuitry, either analog or digital, that is able to perform the calculations prescribed by the plasticity rule. Furthermore, all quantities that are used in the plasticity rule need to be available to those circuits at the point in time when the update is calculated. This means that there needs to be circuitry on the chip to record and potentially buffer all required quantities. Also, since the location where the quantities are measured on chip can be physically distant from where the plasticity is calculated, there needs to be a mechanism in place to transport information about the measured quantities to the appropriate location. Finally, the required quantities need to be in an appropriate format to be processed by the plasticity circuit: a digital circuit needs the quantities it operates on to be digital, while an analog circuit would require analog signals.

The capabilities of implementing on-chip plasticity and the available flexibility therein vary widely across different neuromorphic platforms. Since we have already shown that the in-the-loop implementation of our algorithm is possible on BrainScaleS-2, we propose this system for an attempt at implementing on-chip plasticity. For all other neuromorphic systems we highly recommend to establish the in-the-loop version first, before taking the step to the on-chip approach.

Commonly, the plasticity circuitry is part of the synaptic circuits. On BrainScaleS-2 however, the weight updates are calculated by a general purpose microprocessor on the chip, the plasticity processing unit (PPU). In principle, the plasticity rules that the PPU applies are freely programmable. However, the weight updates for our algorithm (see Section 5.4), using for example the Lambert $\mathcal{W}$ function and exponential functions, are too mathematically complex to compute with feasible turn-around times. In order to make our algorithm compatible with the mathematical capabilities of the PPU we have already explored the possibility of replacing the exact derivatives out of which the weight updates are composed with simplified approximations. Preliminary experiments in Supplementary Information SI.D of our manuscript indicate that while a highly simplified, PPU compatible, learning rule comes at some performance cost, it is still able to successfully train a network on the Yin-Yang dataset. Supplementary Information SI.D explores one specific simplification of the learning rule. There are however different potential approximations that can be chosen.

Before deploying any of the resulting learning rules on the PPU, a more thorough investigation of the trade-off between learning rule simplicity and accuracy penalty is required. In addition to a potential performance loss due to the simplification of the learning rule, we expect some impact on performance caused by the fixed-point arithmetic employed on the PPU. How severely this reduced precision impacts performance is however impossible to predict without further experimental evidence.
For in-the-loop plasticity it is beneficial that our learning rules only depend on the spike times of the neurons because the spikes are typically the quantity for which every hardware has infrastructure to record them and send them off-chip. This is also true for the BrainScaleS-2 system, unfortunately however, these recordings of the spike times are not accessible by the PPU. There are two ways to circumvent this problem:
As shown in Supplementary Information SI.D the simplified learning rule does not depend on the absolute spike times but only on the difference between the pre- and postsynaptic spike time. The exponential of this difference is a measurable quantity on BrainScaleS-2 and is available to the PPU[2]. The circuits measuring the difference are part of the analog synaptic circuits and are called correlation sensors. It might be possible to use the measurements of the correlation sensors as a proxy for the time difference between the pre- and postsynaptic spike time. Since the sensor circuits are analog, they are however subject to significant fixed-pattern noise (Wunderlich et al., 2019, Figure 2). Whether the impact of the fixed-pattern noise and the exponential transformation on the spike time differences is detrimental to the learning performance needs to be investigated.
Alternatively, we suggest to extend the current chip design by additional digital registers available to the PPU where the spike times can be recorded. This requires changes to the circuitry on chip and therefore the tapeout of a new chip version. However, it could already be preliminarily tested in the current chip generation. The new spike time registers can be implemented as part of the external PPU memory on the field-programmable gate array (FPGA) that handles the chip's communication with the host. While this prototype design is expected to be significantly slower than the actual implementation, it does not require any changes to the current chip design and allows to test the algorithm before committing to a new chip version. We expect this second approach to have a higher chance of successfully enabling on-chip learning, as the digital nature of this measurement method both avoids the fixed-pattern noise and the exponential transformation of the correlation sensors.

Recurrent networks and multiple spikes per neuron

While first-spike time codings can be advantageous for fast information processing, there are situations that require operating on longer time scales. For example, in a speech classification task such as the Heidelberg Spiking Digit dataset (Cramer et al., 2020c) each sample has a duration of several hundred milliseconds. This is a far longer timescale than the

[2]Its intended use is for correlation-based learning rules like for example spike-timing-dependent plasticity (STDP) implementations (Billaudelle, 2022, Chapter 3).

typical time-to-classification in our setup (around $1 - 2\tau_{\mathrm{m}}$). Typically, these speech classification or similar temporal tasks require some sort of memory, operating on time scales significantly longer than the membrane time constants of the neurons in the classifying network.

In order to solve temporal tasks such as the Heidelberg Spiking Digit dataset, we need to extend our training algorithm to be able to train recurrent networks. The recurrency in the network automatically introduces, through the loops in the connectivity structure, a second, longer timescale into the network. This recurrency however, only becomes relevant if more than one spike per neuron is allowed. We therefore need to be able to not only calculate the time of the first spike of a neuron, given all its input spike times and synaptic weights, but also the times of all subsequent spikes.

Assuming we already know the time of the first output spike T_1, we can, relative to that, calculate the timing of the second spike. For this we define the point in time when the refractory period after the first spike ends $t = T_1 + \tau_{\text{ref}}$ as $t' = 0$. From this new starting point the second output spike time T_2 can be calculated as was done for T_1 if the differing initial conditions

$$u(t' = 0) = V_{\text{reset}} \tag{7.1}$$

$$I_{\text{syn}}(t' = 0) \neq 0 \tag{7.2}$$

are taken into consideration. The initial value for the synaptic current at $t' = 0$ can be calculated from the sum of all synaptic currents that are induced by spikes arriving during the neuron's refractory period and the residual synaptic current that remains from the input that triggered the first spike. In practice, it might be possible, depending on the length of the refractory period, to neglect the residual synaptic current after the refractory period. With these new initial conditions, all output spike times of the neuron can be calculated iteratively. While the resulting equations for the output spike time do change slightly, they are still differentiable and with that the methods for optimizing the synaptic weights via gradient descent discussed in Chapter 5 should be transferrable.

While the extension to more than the first spike does not entail large changes to the mathematical formulation, it does bring significant new challenges for the practical implementation: First, the software architecture for the training process needs to be adjusted. So far, it is optimized for a layer-wise calculation of the output spike time. In a first-spike and layer-wise setting, all input spikes to a neuron over the whole duration of a sample are known, as they all come from the lower layer, which is calculated before the current layer. Therefore, the output spike times of the current layer can be directly calculated using the equations described in Section 5.4. In a recurrent setting this is not the case: an output spike of the current neuron can, through the recurrent connectivity, cause another input spike into the current neuron at a later point in time. This prevents the direct calculation of all output spike times of a neuron. By not calculating the spike times but instead recording

them either from a neuromorphic emulation or a network simulation we can side-step this issue.
Secondly, we expect the choice of appropriate time constants for the neurons to be more difficult. They still have to (approximately) satisfy either the property of $\tau_{\mathrm{m}} = \tau_{\mathrm{s}}$ or $\tau_{\mathrm{m}} = 2\tau_{\mathrm{s}}$, but the choice of the absolute value is less clear than for the static image classification. For static images encoded via TTFS coding, it has proven to be beneficial to adjust the time constants such that the sample duration is around double τ_{m}. In a more temporal task and a recurrent network there is an additional timescale involved and it is not a-priori clear, and might very well be task-dependent, what the ratios between the timescales of the task, the recurrency and the neuron dynamics should be. In principle, a multidimensional parameter sweep could solve this issue, but in practice this is often unfeasible due to the required amount of compute. Finally, it might even be beneficial to not have the same parameters for all neurons, but to introduce heterogeneity into the network dynamics to improve performance (Perez-Nieves et al., 2021).
The extension to multiple spikes per neuron and recurrent network models requires additional efforts in all areas of the project, in the mathematical foundations, the software architecture, the modeling and parametrization and finally the mechanisms for neuromorphic deployment. Nevertheless, we expect it to be worthwhile as it opens up the capabilities of our algorithm to a whole class of formerly unsuited tasks, where we can potentially provide a fast and efficient solution (Orchard et al., 2015; Amir et al., 2017; Cramer et al., 2020c).

7.3 Towards dendritic microcircuits on neuromorphic hardware

In Chapter 6 we evaluated the dendritic microcircuit model by Sacramento et al. (2018) based on its practical feasibility and neuromorphic implementability and identified three areas for improvement:

- The slow, leaky-integrator neuron dynamics cause a delayed neuronal response to a change in its input. This delay not only slows down information propagation through a deep network but also, more crucially, hampers learning through the introduction of artificial, wrong, error signals.
- The dendritic microcircuits rely on the mechanism of feedback alignment (FA) to avoid the weight transport problem. FA, however, is not well-suited for all network architectures and its performance can be task-dependent.
- The rate-based communication and plasticity mechanisms prevent a direct implementation on spiking neuromorphic hardware.

In Section 6.3 and Haider et al. (2021) we introduced the Latent Equilibrium (LE) framework. Through the mechanism of prospective coding, LE offers a solution to the delayed information propagation and the induced disturbance of learning caused by slow neuronal

dynamics ("relaxation problem"). In this thesis we have mainly focussed on how the LE mechanism can be incorporated into the dendritic microcircuit. We have seen that for the inclusion of LE it is only necessary to modify the activation function: Instead of depending on the normal somatic voltage, the activation function with LE depends on the prospective somatic voltage. The network architecture, error transport mechanisms and learning rules remain unchanged. While the required changes to the model of the dendritic microcircuits are small, their impact on its learning performance is profound: We have illustrated how the required presentation times for an input sample shrink from a minimum of a hundred times τ_{eff} for the original model to below a single τ_{eff}. This results in a shortening of the required simulation times by two orders of magnitude. The practical impact of this can be seen when comparing the training of dendritic microcircuits on the MNIST dataset shown in the original publication of the model (Sacramento et al., 2018) and in Max et al. (2022): In Sacramento et al. (2018) a highly simplified steady-state approximation of the model was used, because the full model could not be trained on a large dataset within reasonable simulation times. In contrast to that, the inclusion of the LE mechanism allowed us in Max et al. (2022) to train on the MNIST dataset while simulating the network with full dynamics, even despite increased model complexity through the learning of the backward synapses.

While the impact of the LE mechanism on the practical feasibility of the dendritic microcircuit model is crucial for the work in this thesis, it is also important to note that LE is not specifically designed for nor tailored towards this model. It can be applied flexibly to any network model that includes leaky-integrator neuron dynamics. In particular, it is expected to be beneficial to a whole array of models targeting biologically plausible error backpropagation that suffer from the relaxation problem.

Besides the inclusion in already existing network models to improve their performance, the ideas by Haider et al. (2021) open up several areas for further research: So far for example, the time constant for the calculation of the prospective voltage was equal to the time constant of the neuronal low-pass filter. As the neuronal components that provide the prospective voltage are, however, most likely not the same ones as those that apply the low-pass filter, the two time constants do not necessarily have to be the same. If they differ, we lose the instantaneity of the information propagation, which would be detrimental for the static image classification tasks shown in Haider et al. (2021). However, for tasks that operate on temporal signals, this might prove beneficial. By introducing different ratios between the low-pass and prospective time constants for different neurons, the network includes neurons that delay an incoming signal, neurons that instantaneously react and neurons for which the output looks into the future. With that, an input signal into the network is processed on multiple time-scales simultaneously and the network also automatically stores information about the input signal at different points in time within the neuron activities. Another wide open area of research is the step from prospective rate-based neurons to prospective spiking or event-based neurons. While it is clear that some form of spiking or event-based communication mechanism is required for an efficient de-

ployment on a neuromorphic platform, whether it is best based on event-based solutions like in Section 6.5.2 or on e.g. encoding information inter-spike-intervals or bursts remains an open question.

In Section 6.4 and Max et al. (2022) we have introduced the phaseless alignment learning (PAL) framework for learning feedback synaptic weights in hierarchical, dynamical network models. Building on and extending the work by Meulemans et al. (2020) and Ernoult et al. (2022), PAL focuses on biological plausibility and implementability in physical systems. In particular, it implements the learning of all synaptic weights in the network in a fully phaseless manner and bases the learning mechanisms solely on quantities locally available in space and time. PAL leverages the noise found in bio-physical systems and uses the correlations between the noise on different signals as an additional carrier of information. With that, the information carried by the signals travelling through the network can be used for the learning of the feedforward weights, while the correlations between the noise on top of the signals provides the information necessary for the learning of the feedback weights. By disentangling the two information carriers using simple low- and highpass filters, which are implementable by biological components, both noise and signal can be present in the network at the same time and can be leveraged for the simultaneous learning of all synaptic weights.

The presented mechanism and theoretical derivations are accompanied by their application to the dendritic microcircuit. The performed experiments demonstrate the ability of PAL to improve a network model's performance compared to when the simple method of FA is used for the avoidance of the weight transport problem. The experiments also directly showcase the value of our previous work in Kriener et al. (2022) by using the Yin-Yang dataset to more clearly demonstrate the advantages of PAL compared to FA in comparison to MNIST and the work in Haider et al. (2021) by allowing us to train comparatively large dendritic microcircuit networks with full dynamics. The results with dendritic microcircuit networks are accompanied by an additional experiment demonstrating PAL's ability to operate on multiple hidden layers. For this purpose dendritic microcircuits are unsuitable due to the reasons discussed in Appendix A.2.3, instead a similar network setup as in Haider et al. (2021, Figure 2) was used.

While we demonstrated PAL's applicability to biologically plausible network models using the dendritic microcircuits, the PAL framework is, similarly to LE, not specific to one network architecture. We expect it to be applicable and beneficial to many other network models that so far employ FA as a solution to the weight transport problem. Furthermore, also similarly to LE, one of the biggest open questions for further research on PAL is how the mechanism can be realized in a spiking or event-based setting. This would not only address the current limitation to rate-based models, but also open up the possibility of a neuromorphic realization.

Finally, in Section 6.5 we addressed the issue of a rate-based dendritic microcircuit not being directly implementable on a spiking neuromorphic platform. We described two potential approaches:

In Section 6.5.1 we developed an alternative, LIF-neuron-based network architecture that mimics the functional principles of the dendritic microcircuits. For this we separated the compartments of the multi-compartment neurons into multiple LIF point neurons. The plasticity mechanisms remained rate-based and we calculated the firing rates from the spike counts of the LIF neurons. We could show that this spiking point neuron setup approximated the functional principles of the original dendritic microcircuit model and with that implemented an approximation of the error backpropagation algorithm.

We were able to demonstrate the functionality of learning the self-predicting state and a small proof-of-concept task both in simulation and on the BrainScaleS-2 hardware. Despite the successful small-scale demonstration, we observed significant problems in scaling to larger networks and more difficult tasks. On the one hand our approach of calculating firing rates by accumulating spike counts resulted in prohibitively long simulation times. Particularly in simulation this issue becomes more severe with increasing network size. On the other hand, we also observed that the error transport mechanism, i.e. reconstructing an error signal from the difference of the activities of interneurons and pyramidal neurons in the layer above, is error-prone and susceptible to disturbance by noise and circuit mismatch. This results in a noisy and potentially unstable error transportation which severely impacts learning performance. We will highlight a potential solution for this in Section 7.3.1.

The observed shortcomings of our LIF-neuron-based approximation for a spiking implementation of dendritic microcircuits lead us to consider an alternative approach: The ideas presented in Section 6.5.2 are based on spikes with payload, a feature that has seen increased popularity on newer neuromorphic systems (IntelLabs, 2021). It allows for a spike signal to carry more than a single bit of information. We can use this to transport information about the firing rates of neurons between pre- and post-synaptic partners. With this, we can have an intermediate solution between a fully rate-based model and a spiking one. As the ideas outlined in Section 6.5.2 are preliminary and have not been extensively tested, more work on this topic, in particular on the areas outlined in the following section, is required.

7.3.1 Future work

For a successful and scalable implementation of a dendritic microcircuit network on a neuromorphic platform two main challenges remain: Firstly, since our first approach to a spiking implemenation with LIF neuron in Section 6.5.1 proved difficult to scale to larger tasks, the alternative ideas suggested in Section 6.5.2 should be investigated in more detail. Secondly, one of the other causes of the scaling difficulties, the error transport mechanism, which is not robust to disruption by noise, should be addressed.

Event-based communication as an intermediate between spikes and rates

The ideas on event-based transmission of neuron activations (firing rates) sketched in Section 6.5.2 are just a first step in the exploration of the potential of spikes with payloads. While we have so far illustrated the mechanistic principles, a more thorough investigation of their practical feasibility is necessary. Here, we will highlight three crucial areas that have not been covered so far.

Firstly, not only firing rate transmission but also plasticity has to be adressed: In the most basic form of event-based firing rate communication, where every neuron sends an event at every time step, we expect neuromorphic implementations to struggle with bandwidth restrictions and communication bottlenecks[3]. It is therefore beneficial to enforce a sparsity in time for the events. Similarly, the calculation of a weight update for every synapse at every time step is expected to be difficult to handle on hardware and might require a reduction in emulation speed. An event-based form of the plasticity mechanisms, where updates are triggered by either pre- or postsynaptic events, would address this issue.

Secondly, the simulations in Section 6.5.2 cover only a highly simplified network structure. The mechanisms need to be tested in a setting where multiple inputs with different event timings connect to one neuron. Additionally, the potential impacts of the recurrent connectivity in the microcircuit architecture on the event rates as well as the precision of information propagation should be investigated. Also, it should be evaluated whether there is a difference in learning performance between the original, rate-based, microcircuit and the microcircuit with event-based communication and plasticity. If there is a performance drop caused by the event-based implementation, its dependence on the parameter choices for the event-based mechanism should be investigated.

Finally, as a last step before the deployment on a neuromorphic platform, the impact of specific hardware restrictions such as fixed-precision integer arithmetics, bandwidth and connectivity limitations or restrictions on the available plasticity mechanisms need to be investigated. As the different hardware platforms vary significantly in their specifications, this final step is specific to the chosen platform. Among the currently available hardware platforms we expect the Loihi 2 chip to be the one with the highest chance for a successful implementation, as it promises native support for spikes with payloads as well as flexible neuron models and plasticity rules (IntelLabs, 2021).

Error transport mechanisms

The dendritic microcircuit model does not directly transport error signals between network layers. Instead, the errors are constructed in each layer through differences in activities of pyramidal and interneurons. In theory, this is an elegant way of avoiding the issues that a direct transport of error signals entails, for example the necessity of communicating

[3]Note that this setup would be a misuse of the event-based neuromorphic infrastructure which is optimized for temporally sparse communication.

positive and negative valued signals via a strictly positive firing rate. In practice, we have however found it to be highly sensitive to imperfections in the network setup, e.g. the self-predicting state (see Appendix A.2.3) and hardware effects such as fixed-pattern noise and quantized weights (see Section 6.5.1):

Learning is driven by the error signals that are constructed in the apical dendrites from the differences in activities between pyramidal and interneurons. Both neuron types receive the same input, but the pyramidal neurons are additionally nudged towards the desired behavior of the network, while the interneurons do not receive the nudging signal. A difference between the two signals therefore reconstructs the nudging, i.e. the error signal. In a setting where the network already solves the task correctly and therefore no error signal is present, the activities of pyramidal and interneurons should match exactly and therefore all signals arriving at the apical compartments should cancel out to zero. In practice this is however not the case for multiple reasons. Even in an ideal, noise free, simulation the assumption that pyramidal and interneurons produce the exact same output in absence of nudging is flawed: While both receive the same input firing rates, as they are connected to the same presynaptic partners, the pyramidal neurons receive their input via synapses with the weights $\boldsymbol{W}^{\mathrm{PP}}$ while the synapses of the interneurons have the weight $\boldsymbol{W}^{\mathrm{IP}}$. As the synaptic weights $\boldsymbol{W}^{\mathrm{PP}}$ constantly evolve during learning, the synapses of the interneurons never exactly match them. This mismatch always introduces an additional, wrong, error signal that disturbs the learning of the task. How severely these wrong errors disrupt learning depends on the relative magnitudes of the actual error signals and the wrong ones. As we have seen in Appendix A.2.3, the magnitude of the actual error signals, decreases with increasing distance to the top layer due to the nudging mechanism. Therefore, in the lower layers, the wrong errors dominate over the actual error signals and prevent learning.

A neuromorphic implementation, in particular on a mixed-signal platform, compounds this problem, since, due to fixed-pattern noise, interneurons and pyramidal neurons cannot produce the exact same output even if they received the exact same inputs via the exact same weights. Also, since parts of the synaptic circuits are analog as well, they are also influenced by fixed-pattern noise. Therefore, even if the same weight value is configured, the effective synaptic weight is not the same. A correction mechanism for that is difficult to realize due to the quantization of the synaptic weight values.

The issues detailed above show that the approach of reconstructing error signals from differences in activities induces a significant amount of practical problems and is in particular not well-suited to an implementation on mixed-signal neuromorphic platforms. We therefore suggest to rethink the error transportation mechanism. An alternative approach could be the direct propagation of error signals via a set of error neurons as for example in Pozzi et al. (2020). We expect this approach to be less susceptible to distortive effects as it is significantly less reliant on fine-tuned interactions between different parts of the network. It however reintroduces the necessity to communicate a signed error signal. Drawing inspiration from neuroscience we can identify two potential ways how this can be realized: One

of them is the method employed in Section 6.5.1 where the error neurons have a baseline activity and a negative error is encoded by activity lower than the baseline (Schultz et al., 1997). Alternatively, we can use two types of error neurons, one coding for the positive and the other for the negative errors (Keller and Mrsic-Flogel, 2018; Hertäg and Sprekeler, 2020; Wilmes et al., 2022). We expect both of these approaches to be more robust than activity differences, but which one is easier to implement and most stable in practice needs to be investigated.
A microcircuit model with a replaced error transport mechanism, in its basic form, will most likely suffer from the same issues introduced by slow neuron dynamics as the dendritic microcircuits. As the LE mechanism (Haider et al., 2021) is rather general and in particular not specifically tailored to the dendritic microcircuits, we expect it to be equally applicable and beneficial to a revised microcircuit model. The same statement holds for the PAL algorithm (Max et al., 2022) which should be used to solve the weight transport problem and align feedback to feedforward weights in the network. Finally, the ideas outlined in Section 6.5.2 should also be transferrable and put a robust and efficient implementation of the revised microcircuit model on event-based hardware systems within reach.

7.4 Conclusions

In this thesis we have utilized two different approaches to bring the concepts of deep learning and error backpropagation onto neuromorphic hardware. The first approach can be described as bottom-up or "device-up", as it aims to develop mechanisms for error backpropagation using the components of a neuromorphic system. The second one represents a top-down or "algorithm-down" strategy, since it starts from an already existing algorithmic model and amends that to be deployable on neuromorphic systems.
Comparing the outcome of the two approaches, at first glance the bottom-up approach appears to have been more fruitful as it led to a functional neuromorphic implementation that is competitive both with regards to algorithmic performance and efficiency. The main reason for this is that a bottom-up approach benefits from the intertwined algorithmic development and neuromorphic implementation. Since it is fundamentally based on existing neuromorphic components, prototypes and subcomponents can be tested early and potential incompatibilities with the neuromorphic platforms can be remedied already during the algorithmic design process. An example for this is our simulated test for robustness to fixed pattern noise: Since we knew our target neuromorphic platform from the beginning, we could estimate the levels of e.g. fixed-pattern noise and evaluate the algorithm's robustness. If these tests had revealed a strong detrimental effect, we would have been able to incorporate these findings in the algorithmic design process and improve robustness. Additionally, since bottom-up approaches typically target one specific or a set of similar neuromorphic systems, they can be specialized to those and optimally exploit the platform's strengths. In our case the choice of the fast TTFS coding scheme paired well with the accelerated emula-

tion of the BrainScaleS-2 system and allowed us to achieve competitively high classification rates.

However, a bottom-up approach is, by design, focussed on achieving functionality within the scope of already existing hardware technologies. In contrast to that, the top-down approach offers more freedom in algorithmic design choices and promotes innovative ideas. We have seen in this thesis how this can lead to more widely applicable concepts: While both the Latent Equilibrium mechanism and the Phaseless Algignment Learning algorithm address weaknesses of the dendritic microcircuits, our solutions remain general and are applicable to a wide spectrum of other models that suffer from similar limitations. In general, a top-down approach settles on an algorithmic solution first and then adapts it to be deployable on a neuromorphic platform. This however, has the significant disadvantage that neuromorphic restrictions are often only considered at the end of the algorithmic design process. This is problematic, since it can be very difficult to exactly predict how the required adaptations and the hardware restrictions impact algorithmic performance. Our suggested adaptation of the dendritic microcircuits, an LIF-neuron-based variant deployed on the BrainScaleS-2 system, suffered from the error-transport mechanism's instability to fixed-pattern noise and a large increase in required training time. These issues, while predictable to a certain degree, were significantly more pronounced than expected and rendered the approach practically infeasible. While this example of the microcircuit's adaptation to LIF-based hardware illustrates the potential pitfalls of a top-down approach, our alternative adaptation based on spike signals with payloads also highlights how the adaptations of an algorithm for deployment can inform innovative ideas for new hardware design principles.

From these considerations we see that our bottom-up approach resulted in an efficient and specialized solution for a currently available neuromorphic system, while the larger freedom provided by the top-down approach has inspired more general algorithmic ideas as well as provided insights into the value of new design principles for future hardware generations.

Appendices

Appendix A

Error Backpropagation in ANNs and Microcircuits

A.1 Derivations for ANNs

Here we derive the weight update and recursive error calculation for a fully connected feedforward network. The output activations $\boldsymbol{y}_n$of the neurons in layer n are given as

$$\boldsymbol{a}_n = \boldsymbol{w}_{n,n-1}\boldsymbol{y}_{n-1} + \boldsymbol{b}_n \tag{A.1}$$

$$\boldsymbol{y}_n = \varphi\left(\boldsymbol{a}_n\right) \tag{A.2}$$

where φ is a differentiable activation function.

We derive the update of the synaptic weights from layer $n-1$ to n via gradient descent on the loss function L

$$\Delta \boldsymbol{w}_{n,n-1} = -\eta \nabla_{\boldsymbol{w}} \mathcal{L}. \tag{A.3}$$

If we consider one element in the weight matrix, the weight connecting neuron i in layer $n-1$ to neuron j in layer n, we can expand the above equation using the chain rule

$$\Delta w^{ij}_{n,n-1} = -\eta \frac{\partial \mathcal{L}}{\partial w^{ij}_{n,n-1}} \tag{A.4}$$

$$= -\eta \frac{\partial \mathcal{L}}{\partial y^i_n} \frac{\partial y^i_n}{\partial a^i_n} \frac{\partial a^i_n}{\partial w^{ij}_{n,n-1}} \tag{A.5}$$

$$= -\eta \frac{\partial \mathcal{L}}{\partial y^i_n} \varphi'\left(a^i_n\right) y^j_{n-1}. \tag{A.6}$$

For the full weight matrix this can be written as

$$\Delta \boldsymbol{w}_{n,n-1} = -\eta \underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi'(\boldsymbol{a}_n)}_{=\boldsymbol{e}_n} \boldsymbol{y}_{n-1}^{\mathrm{T}} \tag{A.7}$$

$$= -\eta\, \boldsymbol{e}_n\, \boldsymbol{y}_{n-1}^{\mathrm{T}} \tag{A.8}$$

with

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial y_n^1} \\ \frac{\partial \mathcal{L}}{\partial y_n^2} \\ \vdots \end{pmatrix} \quad \text{and} \quad \varphi'(\boldsymbol{a}_n) = \begin{pmatrix} \varphi'(a_n^1) \\ \varphi'(a_n^2) \\ \vdots \end{pmatrix} \tag{A.9}$$

and $\odot$ an element-wise multiplication. Here we have defined the error signal as $\boldsymbol{e}_n = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi'(\boldsymbol{a}_n)$. Note that this error signal is not consistently defined in the machine learning literature but the definition chosen here is among the commonly used ones (e.g. Lillicrap et al. (2020)).

To derive the layerwise recursive definition of the error, we first need a layerwise recursive form of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n}$. This can be calculated again for a single element i using the chain rule.

$$\frac{\partial \mathcal{L}}{\partial y_n^i} = \sum_{\substack{j \\ \text{in layer} \\ n+1}} \frac{\partial \mathcal{L}}{\partial y_{n+1}^j} \frac{\partial y_{n+1}^j}{\partial a_{n+1}^j} \frac{\partial a_{n+1}^j}{\partial y_n^i} \tag{A.10}$$

$$= \sum_{\substack{j \\ \text{in layer} \\ n+1}} \frac{\partial \mathcal{L}}{\partial y_{n+1}^j} \varphi'\left(a_{n+1}^j\right) w_{n+1,n}^{ji} \tag{A.11}$$

For the full vector this results in

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} = \boldsymbol{w}_{n+1,n}^{\mathrm{T}} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{n+1}} \odot \varphi'(\boldsymbol{a}_{n+1}) \right). \tag{A.12}$$

With that the recursive definition of the error signal is

$$\boldsymbol{e}_n = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_n} \odot \varphi'(\boldsymbol{a}_n) \tag{A.13}$$

$$= \left[\boldsymbol{w}_{n+1,n}^{\mathrm{T}} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{n+1}} \odot \varphi'(\boldsymbol{a}_{n+1}) \right) \right] \odot \varphi'(\boldsymbol{a}_n) \tag{A.14}$$

$$= \left[\boldsymbol{w}_{n+1,n}^{\mathrm{T}} \boldsymbol{e}_{n+1} \right] \odot \varphi'(\boldsymbol{a}_n). \tag{A.15}$$

A.2 Error backpropagation in dendritic microcircuits

The derivations in this chapter are performed under the following assumptions:

- No feedback alignment (FA) but $\boldsymbol{B}^{\text{PP}}_{\ell-1,\ell} = \boldsymbol{W}^{\text{PP, T}}_{\ell,\ell-1}$
- Perfect self-predicting state at all times
- Interneurons match their corresponding above-layer pyramidal neurons perfectly and the nudging on the interneurons is negligible such that $\boldsymbol{u}^{\text{I}}_{\ell} = \boldsymbol{v}^{\text{bas,*}}_{\ell+1}$
- Only the steady state solutions of the somatic membrane voltages (i.e. the value to which the somatic membrane voltage settles for constant input) are regarded.

To simplify the notation we consider a network with only a single pyramidal neuron per layer. Nevertheless, the methods and calculations are transferrable to networks with larger layers, albeit with more complicated notation.

A.2.1 Hidden layers

A pyramidal neuron in the ℓ-th layer is described by its somatic voltage u^{P}_{ℓ} and its dendritic voltages v^{bas}_{ℓ} and v^{api}_{ℓ}. Given constant inputs (and therefore constant dendritic potentials) the somatic voltage converges to its steady state of

$$u^{\text{P}}_{\ell} = \frac{g^{\text{bas}} v^{\text{bas}}_{\ell} + g^{\text{api}} v^{\text{api}}_{\ell}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}} \tag{A.16}$$

$$= \underbrace{\frac{g^{\text{bas}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}} v^{\text{bas}}_{\ell}}_{v^{\text{bas,*}}_{\ell}} + \underbrace{\frac{g^{\text{api}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}}}_{\lambda} v^{\text{api}}_{\ell} \tag{A.17}$$

$$= v^{\text{bas,*}}_{\ell} + \lambda v^{\text{api}}_{\ell}. \tag{A.18}$$

The scaled dendritic voltage $v^{\text{bas,*}}_{\ell}$ represents the state of the soma in a setting where there is no backward information flow in the network. This corresponds to the purely feedforward "membrane voltages" $\boldsymbol{a}_{\ell}$ in an ANN.

Using a taylor expansion around $v^{\text{bas,*}}_{\ell+1}$, thereby assuming that the impact of the backward information flow is weak, we can express the apical voltages in the hidden layers in a re-

cursive form:

$$v_\ell^{\text{api}} = W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(u_{\ell+1}^{\text{P}}\right) - \varphi\left(u_\ell^{\text{I}}\right)\right] \tag{A.19}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*} + \lambda v_{\ell+1}^{\text{api}}\right) - \varphi\left(v_{\ell+1}^{\text{bas},*}\right)\right] \tag{A.20}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*}\right) + \lambda\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) - \varphi\left(v_{\ell+1}^{\text{bas},*}\right)\right] \tag{A.21}$$

$$= \lambda\, W_{\ell+1,\ell}^{\text{PP, T}}\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) \tag{A.22}$$

For a layer with multiple neurons this results in

$$\boldsymbol{v}_\ell^{\text{api}} = \lambda\, \boldsymbol{W}_{\ell+1,\ell}^{\text{PP, T}} \left(\boldsymbol{v}_{\ell+1}^{\text{api}} \odot \varphi'\left(\boldsymbol{v}_{\ell+1}^{\text{bas},*}\right)\right) \tag{A.23}$$

By again using a taylor expansion we can also rewrite the update rule for the feedforward weights

$$\dot{W}_{\ell,\ell-1}^{\text{PP}} = \eta \left[\varphi\left(u_\ell^{\text{P}}\right) - \varphi\left(v_\ell^{\text{bas},*}\right)\right] \varphi\left(u_{\ell-1}^{\text{P}}\right) \tag{A.24}$$

$$= \eta \left[\varphi\left(v_\ell^{\text{bas},*} + \lambda v_\ell^{\text{api}}\right) - \varphi\left(v_\ell^{\text{bas},*}\right)\right] \varphi\left(u_{\ell-1}^{\text{P}}\right) \tag{A.25}$$

$$= \eta \left[\varphi\left(v_\ell^{\text{bas},*}\right) + \lambda\, v_\ell^{\text{api}}\, \varphi'\left(v_\ell^{\text{bas},*}\right) - \varphi\left(v_\ell^{\text{bas},*}\right)\right] \varphi\left(u_{\ell-1}^{\text{P}}\right) \tag{A.26}$$

$$= \eta\, \lambda\, v_\ell^{\text{api}}\, \varphi'\left(v_\ell^{\text{bas},*}\right)\, \varphi\left(u_{\ell-1}^{\text{P}}\right). \tag{A.27}$$

For a full network this then results in

$$\dot{\boldsymbol{W}}_{\ell,\ell-1}^{\text{PP}} = \eta\, \lambda\, \boldsymbol{v}_\ell^{\text{api}} \odot \varphi'\left(\boldsymbol{v}_\ell^{\text{bas},*}\right)\, \varphi\left(\boldsymbol{u}_{\ell-1}^{\text{P}}\right)^{\text{T}}. \tag{A.28}$$

By comparing Eqn. (A.28) to the weight updates in error backpropagation in Eqn. (A.7) we see a clear correspondence if it can be shown that $\boldsymbol{v}_\ell^{\text{api}} \propto \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_\ell}$. We also find that the layer-wise recursive forms for the apical voltages in Eqn. (A.23) and for $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_\ell}$ in Eqn. (A.12) match.

A.2.2 Top layer

Due to the recursivness $\boldsymbol{v}_\ell^{\text{api}} \propto \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_\ell}$ can be confirmed by showing $\boldsymbol{v}_{N-1}^{\text{api}} \propto \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{N-1}}$ for the highest layer that contains apical compartments.

We start be rewriting the steady state of the somatic voltage in the top layer u_N^{P}

$$u_N^{\mathrm{P}} = \frac{g^{\mathrm{bas}}}{g_{\mathrm{l}} + g^{\mathrm{bas}} + g^{\mathrm{nudge,tgt}}}\, v_N^{\mathrm{bas}} + \frac{g^{\mathrm{nudge,tgt}}}{g_{\mathrm{l}} + g^{\mathrm{bas}} + g^{\mathrm{nudge,tgt}}}\, u^{\mathrm{tgt}} \tag{A.29}$$

$$= \frac{g^{\mathrm{bas}}}{g_{\mathrm{l}} + g^{\mathrm{bas}} + g^{\mathrm{nudge,tgt}}}\, v_N^{\mathrm{bas}} + \alpha\, u^{\mathrm{tgt}} \tag{A.30}$$

$$= (1-\alpha)\; v_N^{\mathrm{bas},*} + \alpha\, u^{\mathrm{tgt}} \tag{A.31}$$

$$= v_N^{\mathrm{bas},*} + \alpha\, \left(u^{\mathrm{tgt}} - v_N^{\mathrm{bas},*}\right) \tag{A.32}$$

with $\alpha = \frac{g^{\mathrm{nudge,tgt}}}{g_{\mathrm{l}}+g^{\mathrm{bas}}+g^{\mathrm{nudge,tgt}}}$ and $v_N^{\mathrm{bas},*} = \frac{g^{\mathrm{bas}}}{g_{\mathrm{l}}+g^{\mathrm{bas}}}\, v_N^{\mathrm{bas}}$ which is the value u_N^{P} converges towards in the absence of nudging. With that we can write the apical voltage as

$$v_{N-1}^{\mathrm{api}} = W_{N,N-1}^{\mathrm{PP,T}}\; \left[\varphi\left(u_N^{\mathrm{P}}\right) - \varphi\left(u_{N-1}^{\mathrm{I}}\right)\right] \tag{A.33}$$

$$= W_{N,N-1}^{\mathrm{PP,T}}\; \left[\varphi\left(v_N^{\mathrm{bas},*} + \alpha\left(u^{\mathrm{tgt}} - v_N^{\mathrm{bas},*}\right)\right) - \varphi\left(v_N^{\mathrm{bas},*}\right)\right] \tag{A.34}$$

$$= W_{N,N-1}^{\mathrm{PP,T}}\; \left[\varphi\left(v_N^{\mathrm{bas},*}\right) + \alpha\; \left(u^{\mathrm{tgt}} - v_N^{\mathrm{bas},*}\right) \varphi'\left(v_N^{\mathrm{bas},*}\right) - \varphi\left(v_N^{\mathrm{bas},*}\right)\right] \tag{A.35}$$

$$= \alpha\, W_{N,N-1}^{\mathrm{PP,T}}\; \left(u^{\mathrm{tgt}} - v_N^{\mathrm{bas},*}\right) \varphi'\left(v_N^{\mathrm{bas},*}\right). \tag{A.36}$$

For the ANN we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{N-1}} = \boldsymbol{w}_{N,N-1}^{\mathrm{T}}\; \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_N} \odot \varphi'\left(\boldsymbol{a}_N\right)\right). \tag{A.37}$$

We now assume the loss function for the dendritic microcircuits to be

$$\mathcal{L} = \frac{1}{2}\; \left(\tilde{u}_N^{\mathrm{P}} - u^{\mathrm{tgt}}\right)^2 \tag{A.38}$$

where $\tilde{u}_N^{\mathrm{P}}$ is the voltage of the top layer if no nudging is applied (we want the network to solve the task also in the absence of a guiding nudging signal). We can then calculate

$$\frac{\partial \mathcal{L}}{\partial \tilde{u}_N^{\mathrm{P}}} = \tilde{u}_N^{\mathrm{P}} - u^{\mathrm{tgt}} \tag{A.39}$$

$$= v_N^{\mathrm{bas},*} - u^{\mathrm{tgt}} \tag{A.40}$$

using that the unnudged potential is by definition $v^{\mathrm{bas},*}$ and with that have shown that indeed v_{N-1}^{api} corresponds to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{N-1}}$. Therefore, we can conclude that under the given assumptions the dendritic microcircuits approximate error backpropagation.

However, we have assumed the outputs of the top-layer pyramidal neurons to be their somatic voltages $\tilde{u}_N^{\mathrm{P}}$ instead of their somatic firing rates $\varphi(\tilde{u}_N^{\mathrm{P}})$. This is technically only

half correct, as the somatic voltages are the "outputs" used for calculating the loss function and reading out the network result, but they are not the output that travels back along the backward connections in the network. In theory, either the loss function should be calculated based on the somatic firing rates or the output that travels backwards should be the somatic voltage. The latter is not possible due to the fact that somatic voltages can be negative and for biological plausibility reasons only strictly positive firing rates can travel along synaptic connections. The former is not advisable since we know from machine learning that the application of an activation function in the output layer hampers learning performance. In practice the applied mixture of outputs in the top layer of the microcircuit networks does not seem to significantly impact the learning performance, but it should be noted that it is strictly speaking neither correct nor biologically plausible.

A.2.3 Multiple hidden layers

Even though the derivation of the error backpropagation mechanism in the dendritic microcircuits is derived for an arbitrary number of hidden layers in a network, it is in practice very hard (to the extent that it has not yet been performed successfully) to train a microcircuit network with full dynamics and more than one hidden layer. An intuition on why this is the case, can be found by revisiting the recursive expression for the apical voltages:

$$v_\ell^{\text{api}} = W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(u_{\ell+1}^{\text{P}}\right) - \varphi\left(u_\ell^{\text{I}}\right)\right] \tag{A.41}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*} + \lambda v_{\ell+1}^{\text{api}}\right) - \varphi\left(u_\ell^{\text{I}}\right)\right] \tag{A.42}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*}\right) + \lambda\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) - \varphi\left(u_\ell^{\text{I}}\right)\right] \tag{A.43}$$

At this point in the derivation we normally assume a perfect self-predicting state and set $u_\ell^{\text{I}} = v_{\ell+1}^{\text{bas},*}$. In practice however the network never is in a perfect self-predicting state since the feedforward weights are constantly changing and the lateral weights are not capable to perfectly keep up. We now include this by setting $u_\ell^{\text{I}} = v_{\ell+1}^{\text{bas},*} + \xi$ where ξ is a mismatch between the pyramidal neuron and the interneuron caused by an imperfect self-predicting state. The magnitude of ξ is mainly determined by the level of fine-tuning on the learning rates. The optimal learning rates are highly dependent on the task, the network size and neuron parameters as well as the weight initialization.

Including the mismatch in the derivation we get

$$v_\ell^{\text{api}} = W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*}\right) + \lambda\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) - \varphi\left(v_{\ell+1}^{\text{bas},*} + \xi\right) \right] \tag{A.44}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\varphi\left(v_{\ell+1}^{\text{bas},*}\right) + \lambda\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) - \left[\varphi\left(v_{\ell+1}^{\text{bas},*}\right) + \varphi'\left(v_{\ell+1}^{\text{bas},*}\right)\, \xi \right] \right] \tag{A.45}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \left[\lambda\, v_{\ell+1}^{\text{api}}\, \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) - \varphi'\left(v_{\ell+1}^{\text{bas},*}\right)\, \xi \right] \tag{A.46}$$

$$= W_{\ell+1,\ell}^{\text{PP, T}} \varphi'\left(v_{\ell+1}^{\text{bas},*}\right) \left[\lambda\, v_{\ell+1}^{\text{api}} - \xi \right] . \tag{A.47}$$

Here we now see that due to the recursion the apical voltage accumulates a factor of $\lambda \ll 1$ for every layer it is below the top layer. The error signal used for learning is therefore downscaled by $\lambda^{N-(\ell+1)}$ once it reaches the layer ℓ. In contrast to that the mismatch ξ is introduced locally and therefore of approximately equal size in all layers. If the learning rates are not fine-tuned enough there exists a layer ℓ below which the mismatch ξ drowns out the error signal and prevents learning. In practice, it turns out that with the typically feasible amount of fine-tuning this layer is already the second one below the top layer.

This finding is seemingly contradicted by the experiment discussed in Sacramento et al. (2018) where a network with two hidden layers (500 neurons each) is trained to solve the MNIST dataset. However, the model there is not simulated with full dynamics but only in the steady state with only one weight update for each input sample. This makes keeping up the self-predicted state significantly easier. In addition to that the chosen method for approximating the steady-state of the network is rather crude and results in a forward information flow in the network that is not influenced by the backward flowing signals. Thereby this approximation effectively removes a crucial characteristic by, for the forward-pass through the network, ignoring the inherent recurrency of the dendritic microcircuit and making it more akin to an ANN than to a dendritic microcircuit simulated with full dynamics. Finally, as shown in Max et al. (2022) an ANN with a single hidden layer of 100 neurons is already capable of achieving similar accuracies on the MNIST dataset. Therefore, the results in Sacramento et al. (2018) could have easily been achieved even if no meaningful learning signal reached the lowest layer.

A.3 Error backpropagation in point neuron microcircuits

In this section we use the approximation in Section 6.5.1 where it was shown that the firing rates of LIF neurons can act similarly to the leaky-integrator neurons used in the original microcircuit models. We can therefore, in the following sections use the same notation as

before and write

$$r^{\text{out}} = \varphi(u) \tag{A.48}$$

$$= \varphi\left(\alpha \sum_{\text{inputs}} W^{\text{in}} r^{\text{in}}\right) \tag{A.49}$$

where α is some parameter dependent proportionality factor. Note that for the case of LIF neurons u is not the actual membrane voltage, which is affected by resets after spikes, but the artificial variable of the free membrane potential.

A.3.1 Nudging via synaptic connections

In contrast to the original microcircuit, where the apical compartment is connected to the soma of the pyramidal neuron via a conductance, the error neuron which replaces the apical compartment connects to the pyramidal neuron via a synapse. In the following we show that for small error signals this synaptic connection has approximately the same effect on the pyramidal neuron as the conductive coupling. Eqn. (A.18) shows that for the original microcircuit the somatic voltage can be described as

$$u_\ell^{\text{P}} = \frac{g^{\text{bas}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}} v_\ell^{\text{bas}} + \frac{g^{\text{api}}}{g_{\text{l}} + g^{\text{bas}} + g^{\text{api}}} v_\ell^{\text{api}} \tag{A.50}$$

which is a sum of the bottom-up signal and the apical signal, both scaled by parameter-dependent constant factors. For the point neuron microcircuit we can describe the pyramidal neuron similarly with

$$u_\ell^{\text{P}} = \alpha\, W_{\ell,\ell-1}^{\text{PP}} r_{\ell-1}^{\text{P}} + \alpha\, W^{\text{B}} \left[r_\ell^{\text{E}} - r^{\text{B}}\right] \tag{A.51}$$

$$= \alpha\, W_{\ell,\ell-1}^{\text{PP}} r_{\ell-1}^{\text{P}} + \alpha\, W^{\text{B}} \left[\varphi\left(u_\ell^{\text{E}}\right) - \varphi\left(0\right)\right] \tag{A.52}$$

where we have assumed the leak of the error neuron to be at zero ($E_{\text{l}} = 0$). For small errors we can assume that u_ℓ^{E} is close to the leak potential, and we can do a taylor expansion

$$u_\ell^{\text{P}} = \alpha\, W_{\ell,\ell-1}^{\text{PP}} r_{\ell-1}^{\text{P}} + \alpha\, W^{\text{B}} \left[\varphi\left(E_{\text{l}}\right) + \varphi'\left(0\right) u_\ell^{\text{E}} - \varphi\left(0\right)\right] \tag{A.53}$$

$$= \alpha\, W_{\ell,\ell-1}^{\text{PP}} r_{\ell-1}^{\text{P}} + \alpha\, W^{\text{B}} \varphi'\left(0\right) u_\ell^{\text{E}}. \tag{A.54}$$

Since both W^{B} and $\varphi'\left(0\right)$ are both constant, we arrive at a similar relationship of u^{P} with the error signal as in the original model.

A.3.2 Correspondence to propagation mechanisms in original microcircuit

Under the same conditions as in Appendix A.2 we derive a recursive form for the error neurons voltage, which replaces the voltage in the apical compartments of the original model

$$u_\ell^{\mathrm{E}} = \tilde{\alpha}\, W_{\ell+1,\ell}^{\mathrm{PP,\,T}} \left[r_{\ell+1}^{\mathrm{P}} - r_\ell^{\mathrm{I}} \right] \tag{A.55}$$

$$= \tilde{\alpha}\, W_{\ell+1,\ell}^{\mathrm{PP,\,T}} \left[\varphi\left(u_{\ell+1}^{\mathrm{P}}\right) - \varphi\left(u_\ell^{\mathrm{I}}\right) \right] \tag{A.56}$$

$$= \tilde{\alpha}\, W_{\ell+1,\ell}^{\mathrm{PP,\,T}} \left[\varphi\left(\alpha\, W_{\ell+1,\ell}^{\mathrm{PP}} r_\ell^{\mathrm{P}} + \alpha\, W^{\mathrm{B}} \varphi'\left(0\right) u_{\ell+1}^{\mathrm{E}}\right) - \varphi\left(u_\ell^{\mathrm{I}}\right) \right]. \tag{A.57}$$

To show the analogy to the original microcircuit we call the bottom-up input $\alpha\, W_{\ell+1,\ell}^{\mathrm{PP}} r_\ell^{\mathrm{P}}$ the basal voltage $v_{\ell+1}^{\mathrm{bas},*}$ even though the basal compartment was absorbed into the pyramidal point neuron. Finally, we use the assumption of a perfect self-predicting state and write $u_\ell^{\mathrm{I}} = v_{\ell+1}^{\mathrm{bas},*}$. Then another taylor expansion shows the layer-wise recursive form for the error neurons:

$$u_\ell^{\mathrm{E}} = \tilde{\alpha}\, W_{\ell+1,\ell}^{\mathrm{PP,\,T}} \left[\varphi\left(v_{\ell+1}^{\mathrm{bas},*} + \alpha\, W^{\mathrm{B}} \varphi'\left(0\right) u_{\ell+1}^{\mathrm{E}}\right) - \varphi\left(v_{\ell+1}^{\mathrm{bas},*}\right) \right] \tag{A.58}$$

$$= \tilde{\alpha}\, W_{\ell+1,\ell}^{\mathrm{PP,\,T}} \varphi'\left(v^{\mathrm{bas},*}\right)\, \alpha\, W^{\mathrm{B}}\, \varphi'(0)\, u_{\ell+1}^{\mathrm{E}} \tag{A.59}$$

This is, up to constant factors, the same as for the same relationship as for the apical compartments in the original microcircuits (Eqn. (A.23)).
In contrast to the error signal, which exactly matches the one in the original microcircuit we can compare the resulting weight updates for the feedforward connections and see a slight deviation there:

$$W_{\ell,\ell-1}^{\mathrm{PP}} = \eta \left[r^{\mathrm{E}} - r^{\mathrm{B}} \right]\, r_{\ell-1}^{\mathrm{P}} \tag{A.60}$$

$$= \eta \left[\varphi\left(u_\ell^{\mathrm{E}}\right) - \varphi(0) \right]\, r_{\ell-1}^{\mathrm{P}} \tag{A.61}$$

$$= \eta\, \varphi'(0)\, u_\ell^{\mathrm{E}}\, r_{\ell-1}^{\mathrm{P}} \tag{A.62}$$

This corresponds to Eqn. (A.27) except for the missing $\varphi'\left(v_\ell^{\mathrm{bas},*}\right)$ in the point neuron microcircuit. Note that the $\varphi'\left(v^{bas,*}\right)$ of the other layers are implicitly included in Eqn. (A.62) through the recursive definition of u^{E} but the derivative of the activation function in the current layer ℓ is missing. Therefore, the weight updates in the point neuron microcircuit only approximately correspond to the error backpropagation weight updates in the original microcircuit model.

Appendix B

Parameter tables

The simulation and emulation parameters for the publications in Chapter 4, Chapter 5 and Section 6.4 are part of the manuscripts and can therefore be found in the respective chapters. The parameters for the summarized results in Section 6.3 can be found in the supplementary information for Haider et al. (2021).

B.1 Background

Table B.1: This table includes the parameters for the simulations shown in Fig. 2.3 and Fig. 2.5. The simulations were performed using the iaf_psc_exp neuron model in the version 2.20.2 of the NEST simulator (Fardet et al., 2021).

Parameter name	Value
Simulator	NEST
Version	2.20.2
Time step	0.1 ms
C_m	1000.0 pF
E_L	−75.0 mV
V_reset	−65.0 mV
V_th	−55 mV
tau_m	10.0 ms
tau_syn_ex	5.0 ms
tau_syn_in	5.0 ms
t_ref	15.0 ms
I_e	0.0 nA

B.2 Point neuron Microcircuits

Table B.2: This table includes the parameters for the simulations shown in Fig. 6.11. The simulations were performed using the iaf_psc_exp neuron model in the version 2.20.2 of the NEST simulator (Fardet et al., 2021).

Parameter name	Value
Simulator	NEST
Version	2.20.2
Time step	0.1 ms
C_m	1000.0 pF
E_L	−65.0 mV
V_reset	−65.0 mV
V_th	−50 mV
tau_m	5.0 ms
tau_syn_ex	15.0 ms
tau_syn_in	15.0 ms
t_ref	1.0 ms
I_e	0.0 nA

Table B.3: This table includes the parameters for the simulations shown in Fig. 6.13.

Parameter name	Value
General	
Simulator	PyNN
Backend	NEST
Time step	0.1 ms
Simulation window	1000.0 ms
Neuron model	IF_curr_exp
Stimulus repetition: (SP[1], learning)	20, 7
Input-Target Pairs	(20, 145), (190, 150) Hz
Network	
Layers size (input-hidden-output)	1–1–1
baseline firing freq.	100.0 Hz
Weight init W^{PP}	$\mathcal{U}[0, 1.5]$ nA
Weight init W^{IP}	$\mathcal{U}[-1.5, 1.5]$ nA
Weight init W^{PI}	$\mathcal{U}[0.6, 1.0]$ nA
Weight init B^{PP}	1.2 nA
Weight init $W_1^{\mathrm{B}}, W_2^{\mathrm{B}}$	0.8 nA, 0.8 nA
Weight range[3]	[-10, 10] nA
Learning rate $\eta_{1,0}^{\mathrm{PP}}$ (SP, learning)	0.0, $1\mathrm{e}{-6}$
Learning rate $\eta_{2,1}^{\mathrm{PP}}$ (SP, learning)	0.0, $5\mathrm{e}{-7}$
Learning rate η^{IP} (SP, learning)	$7\mathrm{e}{-6}$, $5\mathrm{e}{-7}$
Learning rate η^{PI} (SP, learning)	$3\mathrm{e}{-7}$, 0.0
Pyr.[4]+ Interneuron[4]	
cm	1.0 nF
tau_m	10.0 ms
tau_syn_E	15.0 ms
tau_syn_I	15.0 ms
tau_refrac	5.0 ms
Error neuron[4]	
cm	1.0 nF
tau_m	20.0 ms
tau_syn_E	50.0 ms
tau_syn_I	50.0 ms
tau_refrac	0.1 ms

[1] All parameters marked with "SP" were only used in the simulations for learning the self-predicting state.

[2] It proved beneficial to present each stimulus several times before switching to the next sample.

[3] On neuromorphic platforms the range of available rates is typically limited. We model this here, by artificially only allowing synaptic weights in this range. Additionally, the cap on weights can prevent the weights from drifting to excessively high values in case of "wrong error signals" caused by noise or non-perfect self-predicting states.

[4] All potentials were chosen such that the baseline frequency is as given above.

Table B.4: This table includes the parameters for the emulations shown in Fig. 6.14.

Parameter name	Value
General	
Frontend	PyNN
Backend	BrainScaleS-2 (HICANN-X v2)
Simulation window	2000.0 µs[1]
Neuron model	LIF (AdEx circuits switched off)
Stimulus repetition: (SP[2], learning)	7, 7
Input-Target Pairs	(50, 140), (190, 160) kHz
Network	
Layers size (input-hidden-output)	1–1–1
baseline firing freq.	100.0 kHz
Weight init W^{PP}	$\mathcal{U}[0, 50]$
Weight init W^{IP}	$\mathcal{U}[-50, 50]$
Weight init W^{PI}	$\mathcal{U}[15, 63]$
Weight init B^{PP}	$\mathcal{U}[0, 50]$
Weight init $W_1^{\mathrm{B}}, W_2^{\mathrm{B}}$	0, 15
Weight range	[-50, 50]
Learning rate $\eta_{1,0}^{\mathrm{PP}}$ (SP, learning)	0.0, $9\mathrm{e}{-4}$
Learning rate $\eta_{2,1}^{\mathrm{PP}}$ (SP, learning)	0.0, $7\mathrm{e}{-4}$
Learning rate η^{IP} (SP, learning)	$5\mathrm{e}{-4}$, $6\mathrm{e}{-4}$
Learning rate η^{PI} (SP, learning)	$3\mathrm{e}{-4}$, 0.0
Learning threshold θ	0 kHz
Neuron parameters determined individually through calibration routine[3]	

[1] While the plot shows the recorded traces in the biological timescale, here the hardware timescale (1000x accelerated) is used.

[2] All parameters marked with "SP" were only used in the simulations for learning the self-predicting state.

[3] In addition to the standard calibration routine another step of adjusting the firing threshold is added in order to set all neurons up with an as similar as possible baseline firing frequency.

B.3 Event-based Microcircuits

Table B.5: This table includes the parameters for the simulations shown in Figs. 6.15 to 6.17.

Parameter name	Value
Time step	0.1
T_{pres}	15.0
C_{m}	1.0
E_{l}	0.0
g_{l}	0.06
g^{bas}	0.2
g^{api}	0.12
$W_{1,0}^{\text{PP}}$	1.7
$W_{2,1}^{\text{PP}}$	1.3
Activation function	ReLU
Δt (Figs. 6.15 and 6.17)	[1.0, 3.5]
Δu (Figs. 6.16 and 6.17)	[0.02, 0.1]

List of Figures

List of Tables

Acronyms

AdEx adaptive exponential leaky integrate-and-fire 23, 70, 137

ANN artificial neural network 2, 20, 25–28, 50, 55, 58, 70, 88, 94, 95, 104, 105, 112, 114, 115, 124, 133, 153, 173, 175, 177

ASIC application-specific integrated circuit 20, 23, 70, 71

CMOS complementary metal-oxide-semiconductor 21–23

CoBa conductance-based 15, 17, 23, 24, 75

CPU central processing unit 20

CuBa current-based 15–17, 23, 24, 48, 50, 71, 75

FA feedback alignment 29, 87, 94, 95, 105, 107, 114, 115, 117, 160, 162

FPGA field-programmable gate array 20, 59, 71, 158

GeNN GPU enhanced Neuronal Network simulation environment 124

GPU graphical processing unit 20, 31, 124

LE Latent Equilibrium 99–102, 151, 152, 160–162, 166, 186, 187

LIF leaky integrate-and-fire 3, 10–13, 16, 22, 23, 33, 48, 50, 52, 53, 57, 65, 71, 87, 137–142, 153–155, 157, 163, 177, 178, 187

LTD long-term depression 9

LTP long-term potentiation 9, 18

MLP multilayer perceptron 25

MOS metal-oxide-semiconductor 18

nLIF non-leaky integrate-and-fire 50, 65

PAL phaseless alignment learning 109–119, 123–127, 131–134, 162, 166

PPU plasticity processing unit 25, 80, 157, 158

PSC postsynaptic current 8, 15, 16

PSP postsynaptic potential 7, 8, 16, 49, 50, 62, 65, 67, 73, 79

ReLU rectified linear unit 25, 138, 142

SNN spiking neural network 50, 51, 153

SRAM static random-access memory 24, 25

STDP spike-timing-dependent plasticity 17, 21, 158

STP short-term plasticity 9

TTFS time-to-first-spike 47–50, 63, 65, 155, 156, 160, 166

VLSI very-large-scale integration 22, 48

Bibliography

S. A. Aamir, P. Müller, G. Kiene, L. Kriener, Y. Stradmann, A. Grübl, J. Schemmel, and K. Meier. A mixed-signal structured adex neuron for accelerated neuromorphic cores. *IEEE Transactions on Biomedical Circuits and Systems*, 12(5):1027–1037, October 2018a. doi: 10.1109/TBCAS.2018.2848203.

S. A. Aamir, Y. Stradmann, P. Müller, C. Pehle, A. Hartel, A. Grübl, J. Schemmel, and K. 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, December 2018b. doi: 10.1109/TCSI.2018.2840718.

Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

Simone Acciarito, Alessandro Cristini, Luca Di Nunzio, Gaurav Mani Khanal, and Gianluca Susi. An a vlsi driving circuit for memristor-based stdp. In *2016 12th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, pages 1–4. IEEE, 2016.

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. In Martin A. Fischler and Oscar Firschein, editors, *Readings in Computer Vision*, pages 522–533. Morgan Kaufmann, 1987. ISBN 978-0-08-051581-6. doi: 10.1016/B978-0-08-051581-6.50053-2. URL `https://www.sciencedirect.com/science/article/pii/B9780080515816500532`.

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.

Mohamed Akrout, Collin Wilson, Peter C Humphreys, Timothy Lillicrap, and Douglas Tweed. Deep learning without weight transport. *arXiv preprint arXiv:1904.05391*, 2019.

Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza,

et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

Srdjan D Antic, Wen-Liang Zhou, Anna R Moore, Shaina M Short, and Katerina D Ikonomu. The decade of the dendritic nmda spike. *Journal of neuroscience research*, 88(14):2991–3001, 2010.

John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31, 2018.

Bruce P Bean. The action potential in mammalian central neurons. *Nature Reviews Neuroscience*, 8(6):451–465, 2007.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

Yoshua Bengio and Asja Fischer. Early inference in energy-based models approximates back-propagation. *arXiv preprint arXiv:1510.02777*, 2015.

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.

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.

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. *From transistors to learning systems: Circuits and algorithms for brain-inspired computing*. Phd thesis, Universität Heidelberg, May 2022.

Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Domnik Dold, Julian Göltz, Akos F. Kungl, Timo C. Wunderlich, Andreas Hartel, Eric Müller, Oliver J. Breitwieser, Christian Mauch, Mitja Kleider, Andreas Grübl, David Stöckel, Christian Pehle, Arthur Heimbrecht, Philipp Spilger, Gerd Kiene, Vitali Karasenko, Walter Senn, Karlheinz Meier, Johannes Schemmel, and Mihai A. Petrovici. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. *arXiv preprint arXiv:1912.12980*, 2019.

Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Dominik Dold, Julian Göltz, Akos F Kungl, Timo C Wunderlich, Andreas Hartel, et al. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

Sebastian Billaudelle, Benjamin Cramer, Mihai A Petrovici, Korbinian Schreiber, David Kappel, Johannes Schemmel, and Karlheinz Meier. Structural plasticity on an accelerated analog neuromorphic hardware system. *Neural networks*, 133:11–20, 2021.

Sebastian Billaudelle, Johannes Weis, Philipp Dauer, and Johannes Schemmel. An accurate and flexible analog emulation of adex neuron dynamics in silicon. *arXiv preprint arXiv:2209.09280*, 2022.

Jonathan Binas, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. Precise neural network computation with imprecise analog devices. *arXiv preprint arXiv:1606.07786*, 2016.

Tim VP Bliss and Terje Lømo. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of physiology*, 232(2):331–356, 1973.

Mark Bohr. A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007.

Sander M Bohte, Joost N Kok, and Johannes A La Poutré. Spikeprop: backpropagation for networks of spiking neurons. *ESANN*, pages 419–424, 2000.

Jacopo Bono and Claudia Clopath. Modeling somatic and dendritic spike mediated plasticity at the single neuron and network level. 8(1):706, 2017. ISSN 2041-1723.

R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94:3637 – 3642, 2005. doi: NA.

Rodney Brooks, Demis Hassabis, Dennis Bray, and Amnon Shashua. Is the brain a good model for machine intelligence? *Nature*, 482(7386):462, 2012.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

Leonardo Cerliani, Ritu Bhandari, Lorenzo De Angelis, Wietske van der Zwaag, Pierre-Louis Bazin, Valeria Gazzola, and Christian Keysers. Predictive coding during action observation – a depth-resolved intersubject functional correlation study at 7t. *Cortex*, 148:121–138, 2022. ISSN 0010-9452. doi: https://doi.org/10.1016/j.cortex.2021.12.008. URL https://www.sciencedirect.com/science/article/pii/S0010945222000016.

Gregory K Chen, Raghavan Kumar, H Ekin Sumbul, Phil C Knag, and Ram K Krishnamurthy. A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos. *IEEE Journal of Solid-State Circuits*, 54 (4):992–1002, 2018.

Chris73. Action potential. https://commons.wikimedia.org/wiki/File:Action_potential.svg, June 2007. URL https://commons.wikimedia.org/wiki/File:Action_potential.svg. Accessed: 2022-09-22.

Brian D Clark, Ethan M Goldberg, and Bernardo Rudy. Electrogenic tuning of the axon initial segment. *The Neuroscientist*, 15(6):651–668, 2009.

Claudia Clopath, Lars Büsing, Eleni Vasilaki, and Wulfram Gerstner. Connectivity reflects coding: a model of voltage-based STDP with homeostasis. 13(3):344–352, 2010. ISSN 1546-1726. doi: 10.1038/nn.2479.

Iulia M Comsa, Thomas Fischbacher, Krzysztof Potempa, Andrea Gesmundo, Luca Versari, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533, 2020.

Jonathan Cornford, Damjan Kalajdzievski, Marco Leite, Amélie Lamarquette, Dimitri M. Kullmann, and Blake Richards. Learning to live with dale's principle: Anns with separate excitatory and inhibitory units. *bioRxiv*, 2021. doi: 10.1101/2020.11.02.364968.

Rui Costa, Ioannis Alexandros Assael, Brendan Shillingford, Nando de Freitas, and Tim Vogels. Cortical microcircuits as gated-recurrent neural networks. *Advances in neural information processing systems*, 30, 2017.

E Covi, S Brivio, M Fanciulli, and S Spiga. Synaptic potentiation and depression in al: Hfo2-based memristor. *Microelectronic Engineering*, 147:41–44, 2015.

Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019.

Benjamin Cramer. *Optimizing Spiking Neuromorphic Networks for Information Processing*. Phd thesis, Universität Heidelberg, December 2021.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Johannes Schemmel, and Friedemann Zenke. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv preprint arXiv:2006.07239*, 2020a.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *arXiv preprint arXiv:2006.07239*, 2020b.

Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020c.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022.

Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

Stefanie Czischek, Andreas Baumbach, Sebastian Billaudelle, Benjamin Cramer, Lukas Kades, Jan M Pawlowski, Markus Oberthaler, Johannes Schemmel, Mihai A Petrovici, Thomas Gasenzer, et al. Spiking neuromorphic chip learns entangled quantum states. *SciPost Physics*, 12(1):039, 2022.

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*. MIT press, 2005.

Deepmind. Alphafold: a solution to a 50-year-old grand challenge in biology. `https://www.deepmind.com/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology`, 2020. Accessed: 2022-10-19.

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.

Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, 2016.

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*, 119:200–213, 2019.

Thomas Dunwiddie and Gary Lynch. Long-term potentiation and depression of synaptic responses in the rat hippocampus: localization and frequency dependency. *The Journal of physiology*, 276(1):353–367, 1978.

The Economist. Showdown. `https://www.economist.com/science-and-technology/2016/03/12/showdown`, 2016. Accessed: 2022-10-19.

Electronic Vision(s). Brainscales-2: Demos and tutorials. `https://github.com/electronicvisions/brainscales2-demos`, 2022. Accessed: 2022-11-07.

Maxence M Ernoult, Fabrice Normandin, Abhinav Moudgil, Sean Spinney, Eugene Belilovsky, Irina Rish, Blake Richards, and Yoshua Bengio. Towards scaling difference target propagation by learning backprop targets. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5968–5987. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/ernoult22a.html`.

Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

A. Aldo Faisal, Luc P. J. Selen, and Daniel M. Wolpert. Noise in the nervous system. *Nature Reviews Neuroscience*, 9(4):292–303, 2008. ISSN 1471-0048. doi: 10.1038/nrn2258.

Tanguy Fardet, Stine Brekke Vennemo, Jessica Mitchell, Håkon Mørk, Steffen Graber, Jan Hahne, Sebastian Spreizer, Rajalekshmi Deepu, Guido Trensch, Philipp Weidel, Jakob Jordan, Jochen Martin Eppler, Dennis Terhorst, Abigail Morrison, Charl Linssen, Alberto Antonietti, Kael Dai, Alexey Serenko, Binghuang Cai, Piotr Kubaj, Robin Gutzen, Hanjia Jiang, Itaru Kitayama, Björn Jürgens, Sara Konradi, Jasper Albers, and Hans Ekkehard Plesser. Nest 2.20.2, August 2021. URL `https://doi.org/10.5281/zenodo.5242954`.

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.

Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016.

Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm ^{2} 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE transactions on biomedical circuits and systems*, 13(1):145–158, 2018.

Charlotte Frenkel, Jean-Didier Legat, and David Bol. A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas. *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020.

S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier. Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Transactions on Biomedical Circuits and Systems*, 11(1):128–142, 2017. doi: 10.1109/TBCAS.2016.2579164.

Simon Friedmann, Nicolas Frémaux, Johannes Schemmel, Wulfram Gerstner, and Karlheinz Meier. Reward-based learning under hardware constraints—using a risc processor embedded in a neuromorphic substrate. *Frontiers in Neuroscience*, 7:160, 2013.

Johannes Friedrich, Robert Urbanczik, and Walter Senn. Spatio-temporal credit assignment in neuronal population learning. 7(6):e1002092, 2011. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1002092.

Steve Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001, 2016.

Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

Wulfram Gerstner. Spiking neurons. *MIT Press*, 1998.

Wulfram Gerstner. What is different with spiking neurons? *Plausible neural networks for biological modelling*, pages 23–48, 2001.

Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

Wulfram Gerstner and Richard Naud. How good are neuron models? *Science*, 326(5951): 379–380, 2009.

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*, 12:53, 2018.

Sanggyun Gi, Injune Yeo, Myunglae Chu, Seunghun Kim, and Byunggeun Lee. Fundamental issues of implementing hardware neural networks using memristor. In *2015 International SoC Design Conference (ISOCC)*, pages 215–216. IEEE, 2015.

Tim Gollisch and Markus Meister. Rapid neural coding in the retina with relative spike latencies. *Science*, 319(5866):1108–1111, 2008.

Julian Göltz. Training deep networks with time-to-first-spike coding on the brainscales wafer-scale system. Master's thesis, Universität Heidelberg, April 2019. URL `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3909`.

Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Fast and deep: energy-efficient neuromorphic learning with first-spike times. *arXiv:1912.11443*, 2019.

Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A Petrovici. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence*, 3(9):823–835, 2021.

Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Akos Kungl, and Mihai Alexandru Petrovici. Fast and energy-efficient neuromorphic deep learning with first-spike times, July 2021. URL `https://doi.org/10.5281/zenodo.5115007`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Henry Gray. Anatomy of the human body, 1918. URL `https://commons.wikimedia.org/wiki/File:Gray728.svg`.

Will Greedy, Heng Wei Zhu, Joseph Pemberton, Jack Mellor, and Rui Ponte Costa. Single-phase deep learning in cortico-cortical networks. *arXiv preprint arXiv:2206.11769*, 2022.

Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017.

Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature Neuroscience*, 9(3):420, 2006.

Koen V. Haak and Christian F. Beckmann. Objective analysis of the topological organization of the human cortical visual connectome suggests three visual pathways. *Cortex*, 98:73–83, 2018. ISSN 0010-9452. doi: https://doi.org/10.1016/j.cortex.2017.03.020. URL `https://www.sciencedirect.com/science/article/pii/S0010945217301004`.

Paul Haider, Benjamin Ellenberger, Laura Kriener, Jakob Jordan, Walter Senn, and Mihai A Petrovici. Latent equilibrium: Arbitrarily fast computation with arbitrarily slow neurons. *Advances in Neural Information Processing Systems*, 34:17839–17851, 2021.

Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. This edition was published in 2009 by Psychology Press, 1949.

Loreen Hertäg and Henning Sprekeler. Learning prediction error neurons in a canonical interneuron circuit. *Elife*, 9:e57541, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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, 1952.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. *Advances in Neural Information Processing Systems 31*, pages 1433–1443, 2018.

Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*, 2016.

Bernd Illing, Wulfram Gerstner, and Johanni Brea. Biologically plausible deep learning–but how far can we go with shallow networks? *Neural Networks*, 2019.

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.

IntelLabs. Taking neuromorphic computing to the next level with loihi 2. https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf, 2021. Accessed: 2022-09-16.

Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004.

Heidi Johansen-Berg. Structural plasticity: rewiring the brain. *Current Biology*, 17(4):R141–R144, 2007.

Roland S Johansson and Ingvars Birznieks. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience*, 7(2):170, 2004.

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.

Rebecca Jordan and Georg B Keller. Opposing influence of top-down and bottom-up input on excitatory layer 2/3 neurons in mouse primary visual cortex. *Neuron*, 108(6):1194–1206, 2020.

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.

Georg B Keller and Thomas D Mrsic-Flogel. Predictive processing: a canonical cortical computation. *Neuron*, 100(2):424–435, 2018.

Saeed Reza Kheradpisheh and Timothée Masquelier. S4nn: temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(6):2050027, 2020.

Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

James C Knight, Anton Komissarov, and Thomas Nowotny. Pygenn: a python library for gpu-enhanced neural networks. *Frontiers in Neuroinformatics*, 15:659005, 2021.

Christoph Koke. Device Variability in Synapses of Neuromorphic Circuits. *PhD thesis Heidelberg University*, 2017. doi: 10.11588/heidok.00022742.

J.F. Kolen and J.B. Pollack. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1375–1380 vol.3, 1994. doi: 10.1109/ICNN.1994.374486.

Laura Kriener, Julian Göltz, and Mihai A. Petrovici. The yin-yang dataset. *arXiv preprint arXiv:2102.08211*, 2021a.

Laura Kriener, Julian Göltz, and Mihai A Petrovici. Yin-yang dataset repository. `https://github.com/lkriener/yin_yang_data_set`, 2021b. Accessed: 2022-03-20.

Laura Kriener, Julian Göltz, and Mihai A. Petrovici. The yin-yang dataset. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, page 107–111, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450395595. doi: 10.1145/3517343.3517380. URL `https://doi.org/10.1145/3517343.3517380`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012a.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012b.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html*, 55, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

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.

Konrad P. Körding and Peter König. Supervised and unsupervised learning with two sites of synaptic integration. 11(3):207–215, 2001. ISSN 1573-6873. doi: 10.1023/A:1013776130161.

Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. *arXiv preprint arXiv:1906.00889*, 2019.

LM Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs. *J. Physiol. Paris.*, 9:620–635, 1907.

Matthew E Larkum, J Julius Zhu, and Bert Sakmann. A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature*, 398(6725):338–341, 1999.

Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.

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.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

Jane H Lee, Saeid Haghighatshoar, and Amin Karbasi. Exact gradient computation for spiking neural networks through forward propagation. *arXiv preprint arXiv:2210.15415*, 2022.

Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. There's plenty of room at the top: What will drive computer performance after moore's law? *Science*, 368(6495):eaam9744, 2020.

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):1–11, 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(1):1–10, 2016.

Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Yongqiang Cao, Mike Davies, Daniel M Lavery, and Hong Wang. Programming spiking neural networks on intel's loihi. *Computer*, 51(3):52–61, 2018.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

Wolfgang Maass. Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, 102(5):860–880, 2014. doi: 10.1109/JPROC.2014.2310593.

Wolfgang Maass. Searching for principles of brain computation. *Current Opinion in Behavioral Sciences*, 11:81–92, 2016.

Adam H. Marblestone, Greg Wayne, and Konrad P. Kording. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94, 2016. ISSN 1662-5188. doi: 10.3389/fncom.2016.00094. URL `https://www.frontiersin.org/article/10.3389/fncom.2016.00094`.

Henry Markram and Misha Tsodyks. Redistribution of synaptic efficacy between neocortical pyramidal neurons. *Nature*, 382(6594):807–810, 1996.

Kevin Max, Laura Kriener, Garibaldi Pineda García, Thomas Nowotny, Walter Senn, and Mihai A Petrovici. Learning efficient backprojections across cortical hierarchies in real time. *arXiv preprint arXiv:2212.10249*, 2022.

Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. *arXiv preprint arXiv:1911.02385*, 2019.

Mark D. McDonnell and Lawrence M. Ward. The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12(7):415–425, 2011. ISSN 1471-0048. doi: 10.1038/nrn3061.

Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*, volume 80. Springer Science & Business Media, 1989.

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.

Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F Grewe. A theoretical framework for target propagation. *Advances in Neural Information Processing Systems*, 33:20024–20036, 2020.

Alexander Meulemans, Matilde Tristany Farinha, Javier Garcia Ordonez, Pau Vilimelis Aceituno, João Sacramento, and Benjamin F Grewe. Credit assignment in neural networks through deep feedback control. *Advances in Neural Information Processing Systems*, 34, 2021.

Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. Predictive Coding Approximates Backprop along Arbitrary Computation Graphs. 2020a.

Beren Millidge, Alexander Tschantz, Christopher L Buckley, and Anil Seth. Activation relaxation: A local dynamical approximation to backpropagation in the brain. 2020b.

Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

Saber Moradi and Giacomo Indiveri. An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE transactions on biomedical circuits and systems*, 8(1):98–107, 2013.

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.

Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs. Fast classification using sparsely active spiking networks. *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. doi: 10.1109/ISCAS.2017.8050527.

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.

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*, 2020.

Eric Müller, Elias Arnold, Oliver Breitwieser, Milena Czierlinski, Arne Emmel, Jakob Kaiser, Christian Mauch, Sebastian Schmitt, Philipp Spilger, Raphael Stock, et al. A scalable approach to modeling on accelerated neuromorphic hardware. *Frontiers in neuroscience*, 16, 2022.

Manu V Nair and Giacomo Indiveri. An ultra-low power sigma-delta neuron circuit. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

Richard Naud, Nicolas Marcille, Claudia Clopath, and Wulfram Gerstner. Firing patterns in the adaptive exponential integrate-and-fire model. *Biological cybernetics*, 99(4-5):335, 2008.

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.

Emre O Neftci, Bruno U Pedroni, Siddharth Joshi, Maruan Al-Shedivat, and Gert Cauwenberghs. Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in Neuroscience*, 10:241, 2016.

Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*, 2019.

Andrew Ng. What artificial intelligence can and can't do right now. *Harvard Business Review*, 9, 2016.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

OpenAI. Chatgpt: Optimizing language models for dialogue. `https://openai.com/blog/chatgpt/`, 2022a. Accessed: 2022-12-07.

OpenAI. Dall-e 2. `https://openai.com/dall-e-2/`, 2022b. Accessed: 2022-12-13.

Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

Randall C. O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8(5):895–938, 1996.

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. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A. Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv 10.1101/2020.03.30.015511*, 2020. doi: 10.1101/2020.03.30.015511.

Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature neuroscience*, 24(7):1010–1019, 2021.

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.

Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. Neural heterogeneity promotes robust learning. *Nature communications*, 12(1):1–9, 2021.

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, Bernhard Vogginger, Paul Müller, Oliver Breitwieser, Mikael Lundqvist, Lyle Muller, Matthias Ehrlich, Alain Destexhe, Anders Lansner, René Schüffny, et al. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PloS one*, 9(10):e108590, 2014.

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 Alexandru Petrovici. Form versus function: theory and models for neuronal substrates. *Springer*, 2016.

Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience*, 12, 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, 2013.

Robinson E Pino, Hai Li, Yiran Chen, Miao Hu, and Beiye Liu. Statistical memristor modeling and case study in neuromorphic computing. In *DAC Design Automation Conference 2012*, pages 585–590. IEEE, 2012.

Bill Podlaski and Christian K. Machens. Biological credit assignment through dynamic inversion of feedforward networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10065–10076. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/7261925973c9bf0a74d85ae968a57e5f-Paper.pdf`.

Isabella Pozzi, Sander Bohté, and Pieter Roelfsema. A biologically plausible learning rule for deep learning in the brain. *arXiv preprint arXiv:1811.01768*, 2018.

Isabella Pozzi, Sander Bohte, and Pieter Roelfsema. Attention-gated brain propagation: How the brain can implement reward-based error backpropagation. *Advances in Neural Information Processing Systems*, 33:2516–2526, 2020.

Themistoklis Prodromakis and Chris Toumazou. A review on memristive devices and applications. *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, pages 934–937, 2010.

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.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

Alexander Rauch, Giancarlo La Camera, Hans-Rudolf Luscher, Walter Senn, and Stefano Fusi. Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo–like input currents. *Journal of Neurophysiology*, 90(3):1598–1612, 2003.

Alpha Renner, Forrest Sheldon, Anatoly Zlotnik, Louis Tao, and Andrew Sornborger. The backpropagation algorithm implemented on spiking neuromorphic hardware. *arXiv preprint arXiv:2106.07030*, 2021.

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.

Pieter Roelfsema and Arjen Ooyen. Attention-gated reinforcement learning of internal representations for classification. 17:2176–214. doi: 10.1162/0899766054615699.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Nicolas Rougier. Biological neuron schema. https://commons.wikimedia.org/wiki/File:Neuron-figure.svg, June 2007. URL `https://commons.wikimedia.org/wiki/File:Neuron-figure.svg`. Accessed: 2022-09-22.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, pages 533–536, 1986.

Dmitri A. Rusakov, Leonid P. Savtchenko, and Peter E. Latham. Noisy synaptic conductance: Bug or a feature? *Trends in Neurosciences*, 43(6):363–372, 2020. ISSN 0166-2236. doi: 10.1016/j.tins.2020.03.009.

João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in Neural Information Processing Systems*, 31, 2018.

Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2019. doi: 10.1017/9781108186735.

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, Daniel Bruderle, Karlheinz Meier, and Boris Ostendorf. Modeling synaptic plasticity within networks of highly accelerated i&f neurons. In *2007 IEEE international symposium on circuits and systems*, pages 3367–3370. IEEE, 2007.

Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1947–1950, 2010.

Johannes Schemmel, Laura Kriener, Paul Müller, and Karlheinz Meier. An accelerated analog neuromorphic hardware system emulating nmda-and calcium-based non-linear dendrites. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2217–2226, 2017.

Johannes Schemmel, Sebastian Billaudelle, Phillip Dauer, and Johannes Weis. Accelerated analog neuromorphic computing. *arXiv preprint arXiv:2003.11996*, 2020.

Johannes Schemmel, Sebastian Billaudelle, Philipp Dauer, and Johannes Weis. Accelerated analog neuromorphic computing. In *Analog Circuits for Machine Learning, Current/Voltage/Temperature Sensors, and High-speed Communication*, pages 83–102. Springer, 2022.

Sebastian Schmitt, Johann Klähn, Guillaume Bellec, Andreas Grübl, Maurice Güttler, 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. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234, 2017.

Bernhard Scholkopf. The kernel trick for distances. *Advances in neural information processing systems*, pages 301–307, 2001.

Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

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.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Terrence J Sejnowski. The deep learning revolution. *MIT Press*, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 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.

Louis Sokoloff. The metabolism of the central nervous system in vivo. *Handbook of physiology, section I, neurophysiology*, 3:1843–1864, 1960.

Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? – exact implementation of backpropagation in predictive coding networks. *Advances in Neural Information Processing Systems*, 33:22566, 2020.

Yuhang Song, Beren Millidge, Tommaso Salvatori, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Inferring neural activity before plasticity: A foundation for learning beyond backpropagation. *bioRxiv*, 2022. doi: 10.1101/2022.05.17.492325.

Thomas Splettstoesser. Schematic drawing of a chemical synapse. https://commons.wikimedia.org/wiki/File:SynapseSchematic_en.svg, July 2015. URL `https://commons.wikimedia.org/wiki/File:SynapseSchematic_en.svg`. Accessed: 2022-11-02.

Nelson Spruston. Pyramidal neurons: dendritic structure and synaptic integration. 9(3): 206–221, 2008. ISSN 1471-0048. doi: 10.1038/nrn2286.

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. *Internal FACETS Documentation*, 2007.

Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *nature*, 453(7191):80–83, 2008.

Guangzhi Tang, Neelesh Kumar, Ioannis Polykretis, and Konstantinos P Michmizos. Biograd: Biologically plausible gradient-based learning for spiking neural networks. *arXiv preprint arXiv:2110.14092*, 2021.

Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothee Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 2018a.

Amirhossein Tavanaei, Zachary Kirby, and Anthony S Maida. Training spiking convnets by stdp and gradient descent. *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018b.

Corinne Teeter, Ramakrishnan Iyer, Vilas Menon, Nathan Gouwens, David Feng, Jim Berg, Aaron Szafer, Nicholas Cain, Hongkui Zeng, Michael Hawrylycz, et al. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature Communications*, 9(1): 709, 2018.

INA219. Texas Instruments, 12 2015. URL `https://www.ti.com/lit/ds/symlink/ina219.pdf`. Rev. G.

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.

Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520, 1996.

Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14(6-7):715–725, 2001.

Robert Urbanczik and Walter Senn. Learning by the Dendritic Prediction of Somatic Spiking. 81(3):521–528, 2014.

Yoeri van De Burgt, Armantas Melianas, Scott Tom Keene, George Malliaras, and Alberto Salleo. Organic electronics for neuromorphic computing. *Nature Electronics*, 1(7):386–397, 2018.

Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. Alphastar: Mastering the real-time strategy game starcraft ii. `https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii`, 2019. Accessed: 2022-10-19.

Ben von Hünerbein, Ismael Jaras, Laura Kriener, Jakob Jordan, Walter Senn, and Mihai A Petrovici. Towards fully embedded biologically inspired deep learning on neuromorphic hardware. In *6th HBP Student Conference on Interdisciplinary Brain Research*, pages 85–89. Frontiers Event Abstracts, 2022.

John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing, Republication of the original draft in 1993*, 15(4):27–75, 1945.

Runchun M Wang, Chetan S Thakur, and Andre Van Schaik. An fpga-based massively parallel neuromorphic cortex simulator. *Frontiers in neuroscience*, 12:213, 2018.

Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. *System modeling and optimization*, pages 762–770, 1982.

James C. R. Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. 29(5): 1229–1262, 2017.

James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in cognitive sciences*, 23(3):235–250, 2019.

Katharina Wilmes, Constanze Raltchev, Sergej Kasavica, Shankar Babu Sachidhanandam, and Walter Senn. Uncertainty-weighted prediction errors (UPEs) in cortical microcircuits. In *Proceedings of Computational and Systems Neuroscience (Cosyne)*, page 253, 2022.

Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. Deep spiking neural network with spike count based learning rule. *arXiv preprint arXiv:1902.05705*, 2019.

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.

Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):1–17, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

Xiaohui Xie and H. Sebastian Seung. Equivalence of Backpropagation and Contrastive Hebbian Learning in a Layered Network. *Neural Computation*, 15(2):441–454, 02 2003.

Daniel L K Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, 19(3):356–365, March 2016.

Esin Yavuz, James Turner, and Thomas Nowotny. Genn: a code generation framework for accelerated brain simulations. *Scientific reports*, 6(1):1–14, 2016.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

Pawel Zmarz and Georg B Keller. Mismatch receptive fields in mouse visual cortex. *Neuron*, 92(4):766–772, 2016.

Acknowledgments

I would like to thank

Mihai for being all that I could have wished for in a PhD-advisor and even more importantly for being a friend. Thank you for being an inspiring scientist, a great mentor and sometimes (I can't believe I am writing this) for being the voice of reason and diplomacy. Also, though I still don't know how you do it, for always finding a 27th hour in your 26-hour days to talk and help if something is up.

Walter for agreeing to supervise a thesis that focuses on those pesky neuromorphic details that break the nice and clean theories. Thank you for always finding time for a discussion if there is a problem, for your wealth of ideas and for sometimes pulling stunts that e.g. result in us having symposium talks in the research station on top of the Jungfrau Joch!

Prof. Dr. Giacomo Indiveri for being my co-advisor for this thesis and for joining the committee in my midterm exam and the defense. And of course for the yearly Capo Caccia workshops!

Prof. Dr. Steve Furber for taking the time to be external referee for this thesis and for joining the committee in the defense.

Julian for a collaboration that was kicked-off by chance but quickly turned into a valued friendship. Thank you for being an awesome person to share projects, debugging, paper writing, coffee breaks, party pizzas, wine, talks and travels with. Let's claim that we are the only two people ever to take a week-long trip to show up in person for an online conference!

Sebastian for our friendship. From the first day of Mathematischer Vorkurs to a PhD in neuromorphics — what a path to share! Thank you for all your support, kindness and all the fun, both study related and off work. The enthusiasm with which you tend to launch yourself head-first into your work or non-work projects is both inspiring and contagious, and I hope to continue getting drawn into them in the future.

Andreas for both your scientific and non-scientific advice that I could draw on, in particular during different stages of the TTFS work and thesis writing. And of course also for

our many interesting, relaxed, funny and often challenging discussions during not-coffee breaks.

Kevin for our enjoyable and prolific collaboration. It's almost funny how well the combination of you, who stares down a problem with a whiteboard full of equations while turning to the code last, and me, who is the exact opposite, turned out to work. I learned a lot from you and I thoroughly enjoyed our many discussions, both on and off-topic.

Ismael and Ben for allowing me to drag you into this whole microcircuit business. It was amazing to see you two grow together and tackle obstacle after obstacle.

All the proofreaders: Andreas, Julian, Katharina, Luisa, Nicolas, Paul and Sebastian for your time and the valuable feedback. A special thank-you goes to Julian for expertly wielding the languagetool as well as sed and grep and stuff and for not making too much fun of me for all the misspelled micorcirutis. To Luisa for being an English grammar guru and reading almost everything. To Sebastian for all his help with making all the "important chapters" sound nice. And to Paul for always being the first one to read a chapter before I dared to give it to someone else.

Everyone in the NeuroTMA and CompNeuro groups in Bern for being fun and great colleagues.

My family for always being there for me. Thank you for giving me the opportunity to always follow my interests and for supporting me along all the way.

Paul for your love, your calmness and all of your support. Thank you for being who you are. I love you.

Declaration of Originality

Last name, first name: Kriener, Laura Magdalena

Matriculation number: 18–131–680

I hereby declare that this thesis represents my original work and that I have used no other sources except as noted by citations. All data, tables, figures and text citations which have been reproduced from any other source, including the internet, have been explicitly acknowledged as such. I am aware that in case of non-compliance, the Senate is entitled to withdraw the doctorate degree awarded to me on the basis of the present thesis, in accordance with the "Statut der Universität Bern (Universitätsstatut; UniSt)", Art. 69, of 7 June 2011.

Bern, April 25, 2023