

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

J. Göltz*,^{¶,1,2}, L. Kriener*,^{¶,2},

A. Baumbach¹, S. Billaudelle¹, O. Breitwieser¹, B. Cramer¹, D. Dold^{1,3}, A. F. Kungl¹,
W. Senn², J. Schemmel¹, K. Meier^{†,1}, M. A. Petrovici^{¶,2,1}

* Shared first authorship † Deceased

[¶]Corresponding authors (julian.goeltz@kip.uni-heidelberg.de, {laura.kriener,mihai.petrovici}@unibe.ch)

¹Kirchhoff-Institute for Physics, Heidelberg University, 69120 Heidelberg, Germany.

²Department of Physiology, University of Bern, 3012 Bern, Switzerland.

³Siemens 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.

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 [1–3]. It is thus not exaggerated to say that deep learning dominates our understanding of “artificial intelligence” [4–8].

Compared to abstract neural networks used in deep learning, their more biological archetypes — spiking neural networks — still lag behind in performance and scalability [9]. 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 commu-

nication [10, 11], making it more challenging to find reliable coding and computation paradigms [12–14]. Furthermore, one of the major driving forces behind the success of deep learning, the backpropagation of errors algorithm [15–17], remained incompatible with spiking neural networks until only very recently [18, 19].

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 [20], unlike spike-count-based approaches [21–24], 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 [25], 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 [9, 25–27]. This makes spiking neural networks implemented on specialised 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 [9].

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 [28–36] and deep models trained on target spike times by shallow learning rules [37, 38] or using spike-compatible versions of the error backpropagation algorithm [39–41]. 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 [42]. 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 [42–45]. 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 [46–52].

For hierarchical TTFS networks, a gradient-descent-based learning rule was proposed in [53, 54], 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 [25]. 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 [55–57]. An early version of this work was presented in Göltz [58].

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 [48, 59] 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 [33, 34, 60–64].

In the following, we first introduce the CuBa LIF model and the TTFS coding scheme, before we demonstrate how

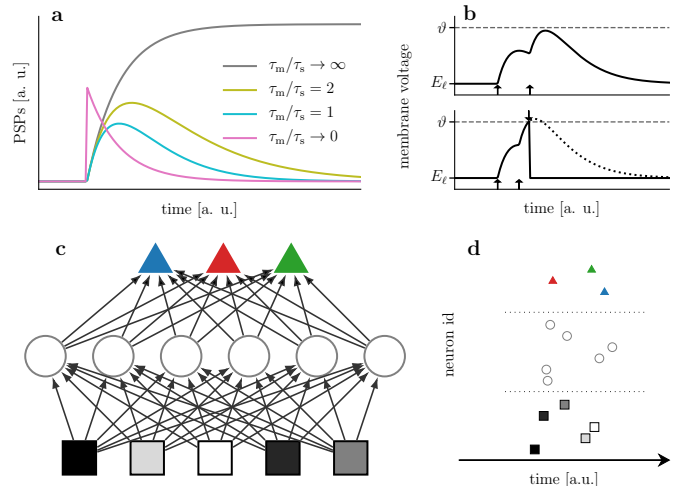


Figure 1: Time-to-first-spike coding and learning. Top: single neurons. (a) Postsynaptic potential (PSP) shapes for different ratios of time constants τ_s and τ_m . The finiteness of time constants causes the neuron to gradually forget prior input. (b) One key challenge of this finite memory arises when 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 \square , hidden \circ , label \triangle). The shading of the input neurons is a representation of the corresponding data, such as pixel brightness ($\blacksquare, \dots, \blacksquare, \dots, \square$). The color of the label neurons represents their respective class ($\blacktriangle, \blacktriangle, \blacktriangle$). (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 (\blacktriangle). In all figures we denote units in square brackets; in particular, we use [a. u.] for arbitrary units, and [1] for dimensionless quantities, and [τ_s] for times that are measured in multiples of the synaptic time constant τ_s .

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.

Results

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

$$C_m \dot{u}(t) = g_\ell [E_\ell - u(t)] + \sum_i w_i \sum_{t_i} \theta(t - t_i) \exp\left(-\frac{t - t_i}{\tau_s}\right), \quad (1)$$

with membrane capacitance C_m , leak conductance g_ℓ (from which the membrane time constant $\tau_m = C_m/g_\ell$ 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 $T \leq t' \leq T + \tau_{\text{ref}}$. For convenience and without loss of generality, we set the leak potential $E_\ell = 0$. Eqn. (1) can be solved analytically and yields subthreshold dynamics as described by Eqn. (9). The choice of τ_m and τ_s ultimately influences the shape of a postsynaptic potential (PSP), starting from a simple exponential ($\tau_m \ll \tau_s$), to a difference of exponentials (with an alpha function for the special case of $\tau_m = \tau_s$) to a graded step function ($\tau_m \gg \tau_s$) (Fig. 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. (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 [53]. 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. 1b).

First-spike times Our spike-timing-based neural code follows an idea first proposed in [53]. 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 manifested 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)

$$T = \tau_s \left\{ \frac{b}{a_1} - \mathcal{W} \left[-\frac{g\ell\vartheta}{a_1} \exp \left(\frac{b}{a_1} \right) \right] \right\} \text{ for } \tau_m = \tau_s \quad (2)$$

and

$$T = 2\tau_s \ln \left[\frac{2a_1}{a_2 + \sqrt{a_2^2 - 4a_1g\ell\vartheta}} \right] \text{ for } \tau_m = 2\tau_s, \quad (3)$$

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. (11) and (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 ($\tau_m = \tau_s$), 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. (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:

$$\frac{\partial t_k}{\partial w_{ki}} = -\frac{1}{a_1} \frac{\exp \left(\frac{t_i}{\tau_s} \right)}{\mathcal{W}(z) + 1} (t_k - t_i), \quad (4)$$

with a_1 and z representing functions of w_{ki} and t_i as defined in Eqns. (11) and (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.

$$\frac{\partial t_k}{\partial t_i} = -\frac{1}{a_1} \frac{\exp\left(\frac{t_i}{\tau_s}\right)}{\mathcal{W}(z) + 1} \frac{w_{ki}}{\tau_s} (t_k - t_i - \tau_s). \quad (5)$$

Together, Eqns. (4) and (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 1c shows a schematic of our feed-forward 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. 1d).

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

To apply the error backpropagation algorithm [15, 17], 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:

$$\begin{aligned} L[\mathbf{t}^{(N)}, n^*] &= \text{dist}\left(t_{n^*}^{(N)}, t_{n \neq n^*}^{(N)}\right) \\ &= \log \left[\sum_n \exp\left(-\frac{t_n^{(N)} - t_{n^*}^{(N)}}{\xi \tau_s}\right) \right], \quad (6) \end{aligned}$$

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. (2) is not defined and neither are its derivatives. We therefore introduce a simple, local heuristic to encourage spiking behaviour 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. (6) can now be easily performed by repeated application of the chain rule. Using the exact derivatives Eqns. (4) and (5), this yields the synaptic plasticity rule

$$\begin{aligned} \Delta w_{ki}^{(l)} &\propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ki}^{(l)}} \quad (7) \\ &= -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}} \underbrace{\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_k^{(l)}}}_{\delta_k^{(l)}} = -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}} \sum_j \frac{\partial t_j^{(l+1)}}{\partial t_k^{(l)}} \delta_j^{(l+1)}. \end{aligned}$$

A compact formulation for hierarchical networks that highlights the backpropagation of errors can be found in Eqns. (38) to (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. (2) or by simulating (or emulating) the LIF dynamics (Eqn. (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. (4) and (5).

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 [65]. For higher-dimensional, discrete input, we used the MNIST data set [66] as a small-scale image classification scenario.

The results in this section are based on Eqn. (2) for calculating the spike times in the forward pass, and Eqn. (40) for calculating weight updates; for details regarding implementation see Methods. For hyperparameters of the discussed experiments see Tables A and B.

Yin-Yang classification task: The first data set consists of points in the yin-yang figure (Fig. 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

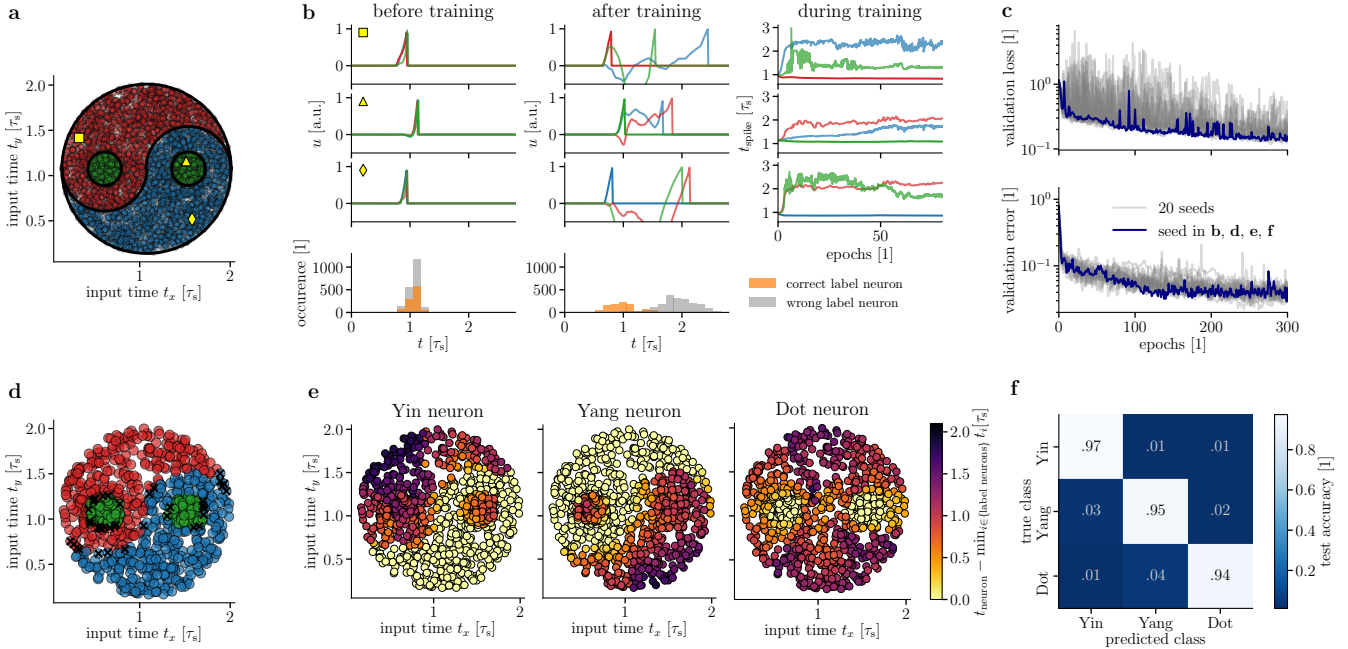


Figure 2: Classification of the Yin-Yang data set. (a) Illustration of the Yin-Yang data set. The samples are separated into three classes, Yin (●), Yang (●) and Dot (●). The yellow symbols (■, ▲, ◆) mark samples for which the training process is illustrated in (b). The input times t_x and t_y correspond 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. (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 motive, the data set is augmented with mirrored coordinates $(1-x, 1-y)$ enabling networks of neurons without trainable bias to learn the task [65]. 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 [67].

After translation of the four features to spike times (see Fig. 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. 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. 2c). The classification failed only for samples that were extremely close to the border between two classes (Fig. 2d). Figure 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. 2e,

left panel).

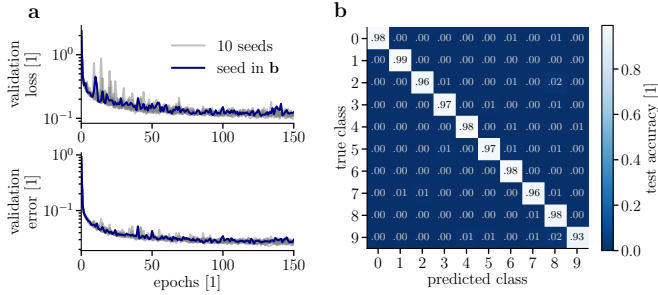


Figure 3: Classification of the MNIST data set. (a) Training progress of a network over 150 epochs for 10 different random initializations. The run drawn in blue is the one which produced the results in (b). (b) Confusion matrix for the test set after training.

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 [66]. Figure 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. 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 [53], 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 & Masquelier [54] report 97.4% using 400 hidden neurons. In Comsa et al. [68], 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. [61] report 92.7% using 512 neurons, while Tavanaei et al. [69] require 1000 hidden neurons to achieve 96.6%.

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 [46, 67], 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 implicitly 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 [72], 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 or spike loss [33, 34, 63, 73].

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 [23, 33, 74], where inference runs emulated on the neuromorphic sub-

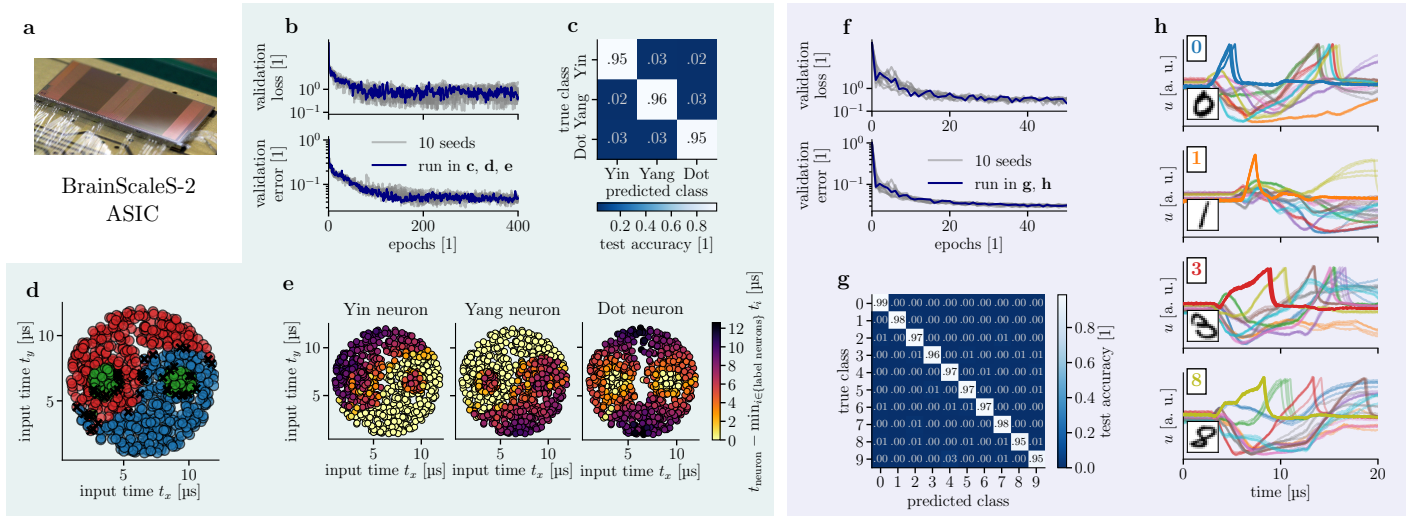


Figure 4: Classification on the BrainScaleS-2 neuromorphic platform. (a) Photograph of a BrainScaleS-2 chip. (b-e) **Yin-Yang data set** (b) Training progress over 200 epochs for 11 different random initializations. The run drawn in blue also produced the results shown in panel (b-d). (c) Confusion matrix 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. 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.

strate 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 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. 2), with the wrong classifications still only happening along the borders of the areas with different labels (Fig. 4c). The remaining difference in performance after training is attributable to the substrate variability (cf. also Fig. 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. 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 2). As shown in Table 2, about one third of the loss in accuracy is due to the downsampling of the data, with the remainder being caused by the variability of the substrate. 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. 4h. In all cases shown here, the correct label neuron spikes before $10 \mu\text{s}$ 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 $480 \mu\text{s}$, 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

Table 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.F1.

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 μ J	125 000	99.2 %	see S.I.E.2
SpiNNaker	digital	130 nm	rate	28×28	784-600-500-10	noisy input encoding	3.3 mJ	91	95.0 %	[70], 2015
True North	digital	28 nm	rate	28×28	CNN	noisy input encoding	0.27 μ J	1000	92.7 %	[61], 2015
True North	digital	28 nm	rate	28×28	CNN	noisy input encoding	108 μ J	1000	99.4 %	[61], 2015
unnamed (Intel)	digital	10 nm	temporal	$(28 \times 28)^3$	236-20	stochastic spike loss	17.1 μ J	6250	89.0 %	[71], 2018
BrainScaleS-2	mixed	65 nm	temporal	16×16	256-246-10	input noise	8.4 μ J	20 800	96.9 %	this work, see also S.I.E.1

¹ Note that some of the 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 S.I.E.2.

³ The 28×28 image is preprocessed using 5×5 Gabor-filters and 3×3 pooling before being sent into the chip.

scalability considerations we refer to Supplementary Information S.I.E. This results in an average energy consumption of 8.4 μ J per classification. For a comparison to other neuromorphic platforms, we refer to Table 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.F1.

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 S.I.E and [74]. 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.

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 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

Table 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

comprehensive study of the robustness of our approach.

Most physical neuronal substrates have several forms of variability in common [75, 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. (4). The output spike time given in Eqn. (2) depends only on neuron parameters, presynaptic spike times and weights, thus its derivatives share the same dependencies (Eqns. (22) and (23)). With some manipulations, the equation for the actual output spike time can be inserted (Eqns. (24) and (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 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 [76]. Our model shows approximately constant performance for weight resolutions down to 5 bit, followed by gradual degradation below (Figure 5b).

Interestingly, adding variability to the synapse and membrane time constants has no discernible effects (Figure 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 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 S1.C we in-

vestigate 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.

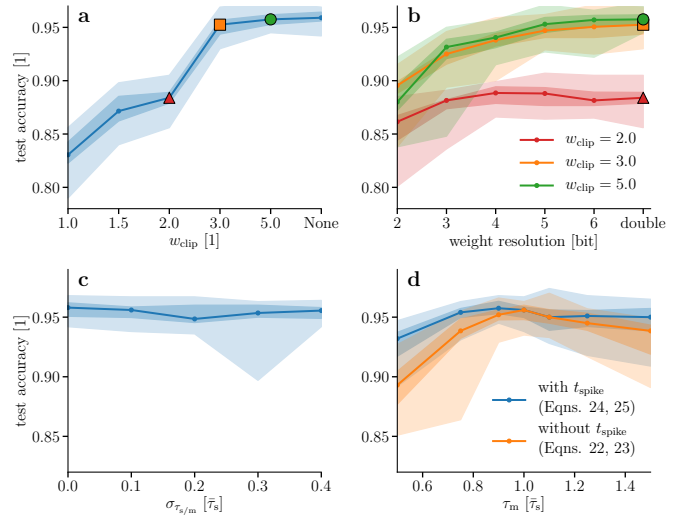


Figure 5: Effects of substrate imperfections. Modeled constraints were added artificially into simulated networks. All panels show median, quartiles, and maximum of the final test accuracy on the Yin-Yang data set for 20 different initializations. **(a)** Limited weight range. The weights were clipped to the range $[-w_{clip}, w_{clip}]$ 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 $[-w_{clip}, w_{clip}]$ 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.

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 ~ 100 ms [42, 43] 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 $10 \mu\text{s}$ 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 $8.4 \mu\text{J}$ 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., [77, 78]. 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.

Methods

Preliminaries In this section we derive the equations from the main manuscript, starting with the learning rule for $\tau_m \rightarrow \infty$, then $\tau_m = \tau_s$, Eqn. (2) and finally $\tau_m = 2\tau_s$, Eqn. (3). The case $\tau_m \rightarrow \infty$ has already been discussed in Mostafa [53] and was reproduced here for completeness and comparison. Due to the symmetry in τ_m and τ_s of the PSP (Eqn. (14)), the $\tau_m = 2\tau_s$ case describes the $\tau_m = \frac{1}{2}\tau_s$ case as well.

For each, a solution for the spike time T , defined by

$$u(T) = \vartheta, \quad (8)$$

has to be found, given LIF dynamics

$$u(t) = \frac{1}{C_m} \frac{\tau_m \tau_s}{\tau_m - \tau_s} \sum_{\text{spikes } t_i} w_i \kappa(t - t_i), \quad (9)$$

$$\kappa(t) = \theta(t) \left[\exp\left(-\frac{t}{\tau_m}\right) - \exp\left(-\frac{t}{\tau_s}\right) \right], \quad (10)$$

with membrane time constant $\tau_m = C_m/g_\ell$ 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. (1) reduces to a single term.

For convenience, we use the following definitions

$$a_n := \sum_{i \in C} w_i \exp\left(\frac{t_i}{n\tau_s}\right), \quad (11)$$

$$b := \sum_{i \in C} w_i \frac{t_i}{\tau_s} \exp\left(\frac{t_i}{\tau_s}\right), \quad (12)$$

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 $\tau_m \rightarrow \infty$ With this choice of τ_m , the first term in Eqn. (10) becomes 1 and we recover the nLIF case discussed in [53]. Given the existence of an output spike, in Eqn. (8) the spike time T appears only in

one place and simple reordering yields

$$\frac{T}{\tau_s} = \ln \left[\frac{a_1}{a_\infty - \vartheta C_m / \tau_s} \right], \quad (13)$$

where we used Eqn. (11) for $n = 1$ and $n = \infty$, the latter being the sum over the weights.

Learning rule for $\tau_m = \tau_s$ According to l'Hôpital's rule, in the limit $\tau_m \rightarrow \tau_s$ Eqn. (9) becomes a sum over α -functions of the form

$$u(t) = \frac{1}{C_m} \sum_i w_i \theta(t - t_i) \cdot (t - t_i) \exp\left(-\frac{t - t_i}{\tau_s}\right). \quad (14)$$

Using these voltage dynamics for the equation of the spike time Eqn. (8), together with the definitions Eqns. (11) and (12) and $\tau_m = C_m/g_\ell$, we get the equation

$$0 = g_\ell \vartheta \exp\left(\frac{T}{\tau_s}\right) + \underbrace{b - a_1 \frac{T}{\tau_s}}_{=: y}. \quad (15)$$

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

$$h \exp(h) = z \quad (16)$$

which can be solved with the differentiable Lambert W function $h = \mathcal{W}(z)$. The goal is now to bring Eqn. (15) into this form, this is achieved by reformulation in terms of y

$$0 = g_\ell \vartheta \exp\left(\frac{b}{a_1}\right) \exp\left(-\frac{y}{a_1}\right) + y \quad (17)$$

$$\underbrace{\frac{y}{a_1}}_{=: h} \exp\left(\frac{y}{a_1}\right) = \underbrace{-\frac{g_\ell \vartheta}{a_1} \exp\left(\frac{b}{a_1}\right)}_{=: z}. \quad (18)$$

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

$$\frac{T}{\tau_s} = \frac{b}{a_1} - \mathcal{W}\left[-\frac{g_\ell \vartheta}{a_1} \exp\left(\frac{b}{a_1}\right)\right]. \quad (19)$$

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. Aa,b). Correspondingly, the Lambert W function (Fig. Ac,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

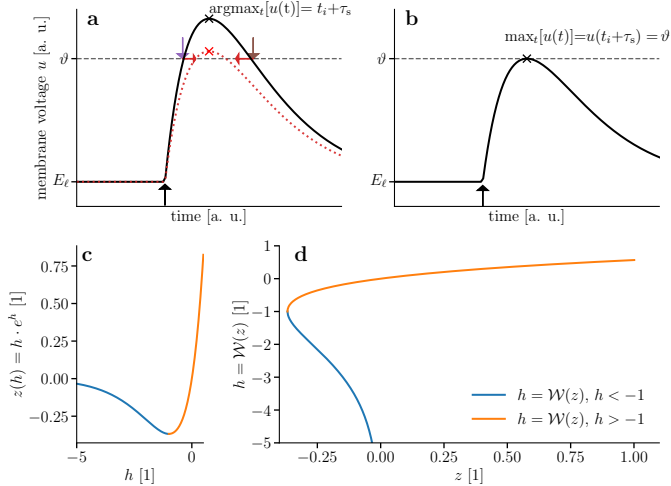


Figure A: (a) Membrane dynamics for one strong input spike at t_i (upward arrow) with two threshold crossings due to leak pullback (earlier violet, later brown). The change induced by a reduction of the input weight is shown in red. (b) Edge case without crossing and exactly one time where $u(t) = \vartheta$.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

look at h from the definition, i.e.

$$h = \frac{y}{a_1} = \frac{b}{a_1} - \frac{T}{\tau_s}. \quad (20)$$

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

$$-1 \leq \frac{t_i - T}{\tau_s} = h. \quad (21)$$

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. Ad), that is where $\mathcal{W} = h > -1$.

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

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) \quad (22)$$

$$= \frac{\tau_s}{a_1} \exp\left(\frac{t_i}{\tau_s}\right) \left[z\mathcal{W}'(z) + \left(\frac{t_i}{\tau_s} - \frac{b}{a_1}\right) (1 - z\mathcal{W}'(z)) \right],$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) \quad (23)$$

$$= \frac{w_i}{a_1} \exp\left(\frac{t_i}{\tau_s}\right) \left[1 + \left(\frac{t_i}{\tau_s} - \frac{b}{a_1}\right) (1 - z\mathcal{W}'(z)) \right].$$

A crucial step is to reinsert the definition of the spike time where it is possible (cf. Fig. 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. (16) with $h = \mathcal{W}(z)$ with respect to z . With this equation one can calculate the derivative of Eqn. (19) with respect to incoming weights and times as functions of presynaptic weights, input spike times and output spike time:

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}, T) = -\frac{1}{a_1} \frac{1}{\mathcal{W}(z) + 1} \exp\left(\frac{t_i}{\tau_s}\right) (T - t_i), \quad (24)$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}, T) = -\frac{1}{a_1} \frac{1}{\mathcal{W}(z) + 1} \exp\left(\frac{t_i}{\tau_s}\right) \frac{w_i}{\tau_s} (T - t_i - \tau_s). \quad (25)$$

These equations are equivalent to the Eqns. (4) and (5) shown in the main text.

Learning rule for $\tau_m = 2\tau_s$ Inserting the voltage (Eqn. (9)) into the spike time (Eqn. (8)) yields

$$g\ell\vartheta = \exp\left(-\frac{T}{\tau_m}\right) \sum_{i \in C} w_i \exp\left(\frac{t_i}{\tau_m}\right) - \exp\left(-\frac{T}{\tau_s}\right) \sum_{i \in C} w_i \exp\left(\frac{t_i}{\tau_s}\right). \quad (26)$$

Reordering and rewriting this in terms of a_1 , a_2 , and τ_s (with $\tau_m = 2\tau_s$) we get

$$0 = -a_1 \left[\exp\left(-\frac{T}{2\tau_s}\right) \right]^2 + a_2 \exp\left(-\frac{T}{2\tau_s}\right) - g\ell\vartheta. \quad (27)$$

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

$$\frac{T}{\tau_s} = 2 \ln \left[\frac{2a_1}{a_2 + \sqrt{a_2^2 - 4a_1g\ell\vartheta}} \right]. \quad (28)$$

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 $x = \sqrt{a_2^2 - 4a_1g\ell\vartheta}$ for

brevity, the derivatives of Eqn. (28) are

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) \quad (29)$$

$$= 2\tau_s \left[\frac{1}{a_1} + \frac{2g_\ell \vartheta}{(a_2 + x)x} \right] \exp\left(\frac{t_i}{\tau_s}\right) - \frac{2\tau_s}{x} \exp\left(\frac{t_i}{2\tau_s}\right),$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) \quad (30)$$

$$= 2w_i \left[\frac{1}{a_1} + \frac{2g_\ell \vartheta}{(a_2 + x)x} \right] \exp\left(\frac{t_i}{\tau_s}\right) - \frac{w_i}{x} \exp\left(\frac{t_i}{2\tau_s}\right).$$

Again, inserting the output spike time yields

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}, T) \quad (31)$$

$$= \frac{2\tau_s}{a_1} \left[1 + \frac{g_\ell \vartheta}{x} \exp\left(\frac{T}{2\tau_s}\right) \right] \exp\left(\frac{t_i}{\tau_s}\right) - \frac{2\tau_s}{x} \exp\left(\frac{t_i}{2\tau_s}\right),$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}, T) \quad (32)$$

$$= \frac{2w_i}{a_1} \left[1 + \frac{g_\ell \vartheta}{x} \exp\left(\frac{T}{2\tau_s}\right) \right] \exp\left(\frac{t_i}{\tau_s}\right) - \frac{w_i}{x} \exp\left(\frac{t_i}{2\tau_s}\right).$$

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

$$\Delta w_{ni}^{(N)} \propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ni}^{(N)}} = -\frac{\partial t_n^{(N)}}{\partial w_{ni}^{(N)}} \frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_n^{(N)}}. \quad (33)$$

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

$$\Delta w_{ki}^{(l)} \propto -\frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial w_{ki}^{(l)}} = -\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}} \delta_k^{(l)}, \quad (34)$$

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

$$\delta_k^{(l)} := \frac{\partial L[\mathbf{t}^{(N)}, n^*]}{\partial t_k^{(l)}} = \sum_j \frac{\partial t_j^{(l+1)}}{\partial t_k^{(l)}} \delta_j^{(l+1)}. \quad (35)$$

In the following we treat the $\tau_m = \tau_s$ case but the calculations can be performed analogously for the other cases. Rewriting Eqns. (24) and (25) in a layer-wise setting, the derivatives of the spike time for a neuron k in arbitrary layer

l are

$$\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) \quad (36)$$

$$= -\frac{1}{a_1} \exp\left(\frac{t_i^{(l-1)}}{\tau_s}\right) \frac{1}{\mathcal{W}(z) + 1} (t_k^{(l)} - t_i^{(l-1)}),$$

$$\frac{\partial t_k^{(l)}}{\partial t_i^{(l-1)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) \quad (37)$$

$$= -\frac{1}{a_1} \exp\left(\frac{t_i^{(l-1)}}{\tau_s}\right) \frac{1}{\mathcal{W}(z) + 1} \frac{w_{ki}^{(l)}}{\tau_s} (t_k^{(l)} - t_i^{(l-1)} - \tau_s).$$

Inserting Eqns. (35) to (37) into Eqns. (33) and (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:

$$\delta^{(N)} = \frac{\partial L}{\partial \mathbf{t}^{(N)}}, \quad (38)$$

$$\delta^{(l-1)} = \left(\widehat{\mathbf{B}}^{(l)} - \mathbf{1} \right) \odot \boldsymbol{\rho}^{(l-1)} \odot \left(\mathbf{w}^{(l),T} \delta^{(l)} \right), \quad (39)$$

$$\Delta \mathbf{w}^{(l)} = -\eta \tau_s \left(\delta^{(l)} \boldsymbol{\rho}^{(l-1),T} \right) \odot \widehat{\mathbf{B}}^{(l)}, \quad (40)$$

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

$$\rho_i^{(l)} = -\frac{1}{a_1} \exp\left(\frac{t_i^{(l)}}{\tau_s}\right) \frac{1}{\mathcal{W}(z) + 1}, \quad (41)$$

$$\widehat{B}_{ki}^{(l)} = \frac{t_k^{(l)} - t_i^{(l-1)}}{\tau_s}. \quad (42)$$

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 configured; these circuits can be restricted to LIF dynamics as required by our training framework [79]. Both the membrane and synaptic time constants were calibrated to 6 μ s.

Each neuron circuit is connected to one of four synapse matrices on the chip, and integrates stimuli from its col-

umn of 256 CuBa synapses [59]. 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. 4h therefore originates from a different repetition of the experiment.

The ASIC is controlled by a layered software stack [80] 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` [81]. The network module implements layers of LIF neurons whose spike times are calculated according to Eqn. (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. (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 rep-

resent 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 the Adam optimizer [82] and learning rate scheduling (the parameters can be found in Tables A and B).

To assist learning we employ several regularization techniques. The term $+\alpha \left[\exp \left(t_n^{(N)} / \beta \tau_s \right) - 1 \right]$ with scaling parameters $\alpha, \beta \in \mathbb{R}^+$, can be added to the loss in Eqn. (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. (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 $\tau_{\text{ref}} > \tau_s$ 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

Data available on request from the authors.

Code availability

Code of the Yin-Yang data set [65] available at https://github.com/lkriener/yin_yang_data_set, other code available on request from the authors.

Table A: Neuron, network and training parameters used to produce the results in Figs. 2 and 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	$[0.9\tau_s, 0.9\tau_s]$	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	0.15
t_{late}	2.0	2.0
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 B: Network and training parameters for training on BrainScaleS-2 used to produce the results in Fig. 4. In contrast to Table A, 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 τ_s , 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	0.15
t_{late}	2.0	2.0 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.

References

1. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Image classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097–1105 (2012).
2. Silver, D. et al. Mastering the game of go without human knowledge. *Nature* **550**, 354 (2017).
3. Brown, T. B. et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
4. Brooks, R., Hassabis, D., Bray, D. & Shashua, A. Is the brain a good model for machine intelligence? *Nature* **482**, 462 (2012).
5. Ng, A. What artificial intelligence can and can't do right now. *Harvard Business Review* **9** (2016).
6. Hassabis, D., Kumaran, D., Summerfield, C. & Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **95**, 245–258 (2017).
7. Sejnowski, T. J. The deep learning revolution. *MIT Press* (2018).
8. Richards, B. A. et al. A deep learning framework for neuroscience. *Nature Neuroscience* **22**, 1761–1770 (2019).
9. Pfeiffer, M. & Pfeil, T. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience* **12** (2018).
10. Gerstner, W. What is different with spiking neurons? *Plausible neural networks for biological modelling*, 23–48 (2001).
11. Izhikevich, E. M. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks* **15**, 1063–1070 (2004).
12. Gerstner, W. Spiking neurons. *MIT Press* (1998).
13. Maass, W. Searching for principles of brain computation. *Current Opinion in Behavioral Sciences* **11**, 81–92 (2016).
14. Davies, M. Benchmarks for progress in neuromorphic computing. *Nature Machine Intelligence* **1**, 386–388 (2019).
15. Linnainmaa, S. 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*, 6–7 (1970).
16. Werbos, P. J. Applications of advances in nonlinear sensitivity analysis. *System modeling and optimization*, 762–770 (1982).

17. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature*, 533–536 (1986).
18. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. *Neural Networks* (2018).
19. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948* (2019).
20. Güttig, R. & Sompolinsky, H. The tempotron: a neuron that learns spike timing-based decisions. *Nature Neuroscience* **9**, 420 (2006).
21. Cao, Y., Chen, Y. & Khosla, D. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision* **113**, 54–66 (2015).
22. Diehl, P. U., Zarella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 1–8 (2016).
23. Schmitt, S. 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)*, 2227–2234 (2017).
24. Wu, J. et al. Deep Spiking Neural Network with Spike Count based Learning Rule. *arXiv preprint arXiv:1902.05705* (2019).
25. Thakur, C. S. T. et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in Neuroscience* **12**, 891 (2018).
26. Mead, C. Neuromorphic electronic systems. *Proceedings of the IEEE* **78**, 1629–1636 (1990).
27. Roy, K., Jaiswal, A. & Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature* **575**, 607–617 (2019).
28. Petrovici, M. A., Bill, J., Bytschok, I., Schemmel, J. & Meier, K. Stochastic inference with deterministic spiking neurons. *arXiv preprint arXiv:1311.3211* (2013).
29. Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K. & Cauwenberghs, G. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience* **7**, 272 (2014).
30. Petrovici, M. A., Bill, J., Bytschok, I., Schemmel, J. & Meier, K. Stochastic inference with spiking neurons in the high-conductance state. *Physical Review E* **94**, 042312 (2016).
31. Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M. & Cauwenberghs, G. Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in Neuroscience* **10**, 241 (2016).
32. Leng, L. et al. Spiking neurons with short-term synaptic plasticity form superior generative networks. *Scientific Reports* **8**, 1–11 (2018).
33. Kungl, A. F. et al. Accelerated physical emulation of Bayesian inference in spiking neural networks. *Frontiers in Neuroscience* **13**, 1201 (2019).
34. Dold, D. et al. Stochasticity from function—Why the Bayesian brain may need no noise. *Neural Networks* **119**, 200–213 (2019).
35. Jordan, J. et al. Deterministic networks for probabilistic computing. *Scientific Reports* **9**, 1–17 (2019).
36. Hunsberger, E. & Eliasmith, C. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141* (2016).
37. Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J. & Masquelier, T. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* **99**, 56–67 (2018).
38. Illing, B., Gerstner, W. & Brea, J. Biologically plausible deep learning—but how far can we go with shallow networks? *Neural Networks* (2019).
39. Bohte, S. M., Kok, J. N. & La Poutré, J. A. Spike-Prop: backpropagation for networks of spiking neurons. *ESANN*, 419–424 (2000).
40. Zenke, F. & Ganguli, S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation* **30**, 1514–1541 (2018).
41. Huh, D. & Sejnowski, T. J. Gradient Descent for Spiking Neural Networks. *Advances in Neural Information Processing Systems 31*, 1433–1443 (2018).
42. Thorpe, S., Delorme, A. & Van Rullen, R. Spike-based strategies for rapid processing. *Neural Networks* **14**, 715–725 (2001).
43. Thorpe, S., Fize, D. & Marlot, C. Speed of processing in the human visual system. *Nature* **381**, 520 (1996).
44. Johansson, R. S. & Birznieks, I. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature Neuroscience* **7**, 170 (2004).
45. Gollisch, T. & Meister, M. Rapid neural coding in the retina with relative spike latencies. *Science* **319**, 1108–1111 (2008).

46. Schemmel, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 1947–1950 (2010).
47. Akopyan, F. 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**, 1537–1557 (2015).
48. Billaudelle, S. et al. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. *arXiv preprint arXiv:1912.12980* (2019).
49. Davies, M. et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).
50. Mayr, C., Hoepfner, S. & Furber, S. SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning. *arXiv preprint arXiv:1911.02385* (2019).
51. Pei, J. et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* **572**, 106–111 (2019).
52. Moradi, S., Qiao, N., Stefanini, F. & Indiveri, G. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE transactions on biomedical circuits and systems* **12**, 106–122 (2017).
53. Mostafa, H. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems* **29**, 3227–3235 (2017).
54. Kheradpisheh, S. R. & Masquelier, T. S4NN: temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems* **30**, 2050027 (2020).
55. Rauch, A., La Camera, G., Luscher, H.-R., Senn, W. & Fusi, S. Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo-like input currents. *Journal of Neurophysiology* **90**, 1598–1612 (2003).
56. Gerstner, W. & Naud, R. How good are neuron models? *Science* **326**, 379–380 (2009).
57. Teeter, C. et al. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature Communications* **9**, 709 (2018).
58. Göltz, J. *Training Deep Networks with Time-to-First-Spike Coding on the BrainScaleS Wafer-Scale System* Master’s thesis (Universität Heidelberg, Apr. 2019). <http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3909>.
59. Friedmann, S. et al. Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System. *IEEE Transactions on Biomedical Circuits and Systems* **11**, 128–142 (2017).
60. Prodromakis, T. & Toumazou, C. A review on memristive devices and applications. *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, 934–937 (2010).
61. Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V. & Modha, D. S. Backpropagation for energy-efficient neuromorphic computing. *Advances in Neural Information Processing Systems*, 1117–1125 (2015).
62. Van De Burgt, Y., Melianas, A., Keene, S. T., Malliaras, G. & Salleo, A. Organic electronics for neuromorphic computing. *Nature Electronics* **1**, 386–397 (2018).
63. Wunderlich, T. et al. Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in Neuroscience* **13**, 260 (2019).
64. Feldmann, J., Youngblood, N., Wright, C., Bhaskaran, H & Pernice, W. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature* **569**, 208 (2019).
65. Kriener, L., Göltz, J. & Petrovici, M. A. The Yin-Yang dataset. *arXiv preprint arXiv:2102.08211* (2021).
66. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**, 2278–2324 (1998).
67. Schemmel, J., Billaudelle, S., Dauer, P. & Weis, J. Accelerated Analog Neuromorphic Computing. *arXiv preprint arXiv:2003.11996* (2020).
68. Comsa, I. M. et al. Temporal coding in spiking neural networks with alpha synaptic function. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8529–8533 (2020).
69. Tavanaei, A., Kirby, Z. & Maida, A. S. Training spiking convnets by stdp and gradient descent. *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2018).

70. Stromatias, E. et al. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2015).
71. Chen, G. K., Kumar, R., Sumbul, H. E., Knag, P. C. & Krishnamurthy, R. K. 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**, 992–1002 (2018).
72. Aamir, S. A. et al. An Accelerated LIF Neuronal Network Array for a Large-Scale Mixed-Signal Neuromorphic Architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers* **65**, 4299–4312 (Dec. 2018).
73. Petrovici, M. A. et al. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PloS one* **9**, e108590 (2014).
74. Cramer, B. et al. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv preprint arXiv:2006.07239* (2020).
75. Petrovici, M. A. Form versus function: theory and models for neuronal substrates. *Springer* (2016).
76. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* **18**, 6869–6898 (2017).
77. Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv* *10.1101/2020.03.30.015511* (2020).
78. Sacramento, J. a., Ponte Costa, R., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in Neural Information Processing Systems* **31** (2018).
79. Aamir, S. A. et al. A Mixed-Signal Structured AdEx Neuron for Accelerated Neuromorphic Cores. *IEEE Transactions on Biomedical Circuits and Systems* **12**, 1027–1037 (Oct. 2018).
80. Müller, E. et al. Extending BrainScaleS OS for BrainScaleS-2. *arXiv preprint arXiv:2003.13750* (2020).
81. Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems* *32*, 8024–8035 (2019).
82. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

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. 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 under grant agreements 604102, 720270, 785907, 945539 (HBP) and the Manfred Stärk Foundation.

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.

Supplementary Information

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

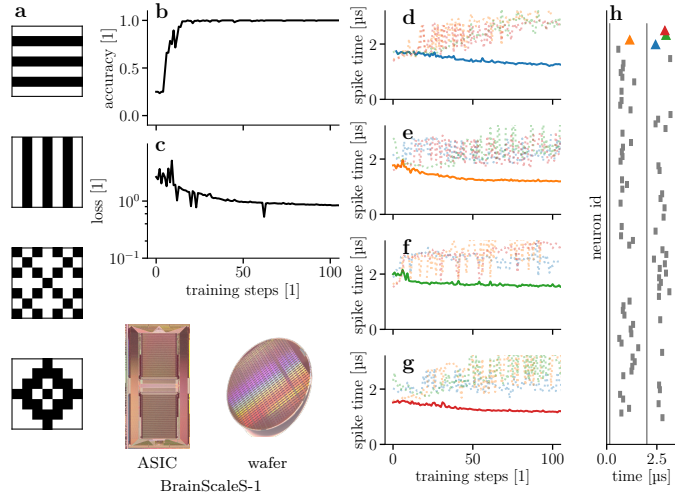


Figure SI.A1: Training a spiking network on the wafer-scale BrainScaleS-1 system. (a) Simple data set consisting of 4 classes with 7×7 input pixels. Accuracy (b) and loss (c) during training of the four pattern data set. (d-g) Evolution of the spike times in the label layer for the four different patterns. 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.

To demonstrate the applicability of our approach to different neuromorphic substrates, we also tested it on the BrainScaleS-1 system [1]. 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 current-based (CuBa) neurons. Furthermore, due to the different fabrication technology and design choices [in particular, the floating-gate parameter memory, see 1–3], the parameter variability and spike time jitter are significantly higher than on BrainScaleS-2 [4].

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 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 A. 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.

Table SI.B1: Additional simulation runs on the MNIST data set. The values given as the baseline are taken from Table 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 A and the simulations for the input resolution of 16×16 used the hyperparameters given in Table B.

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.

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. 4 and 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 $\tau_s = \tau_m = 1$ 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 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.

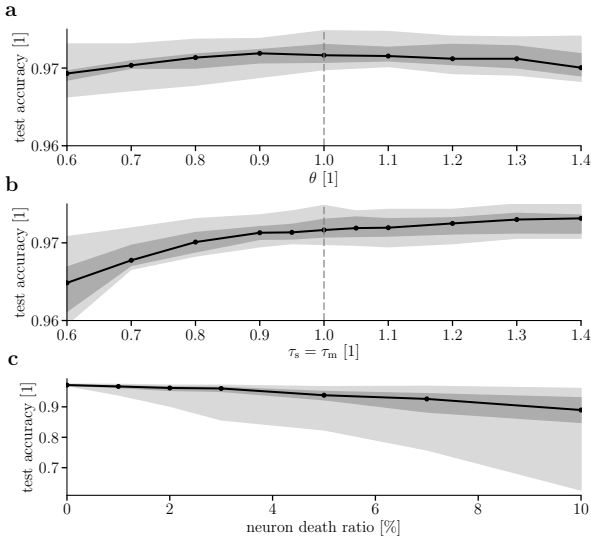


Figure SI.C1: Robustness to variations not present during training. All panels show median (black), quartiles (dark gray), as well as the entire range between minimum and maximum (light gray) in the shaded regions. **(a)** Dependence of test accuracy for evaluation for 10 trained networks with shifted 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).

SI.D Simplification of the learning rule

The learning rule for $\tau_m = \tau_s$ 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

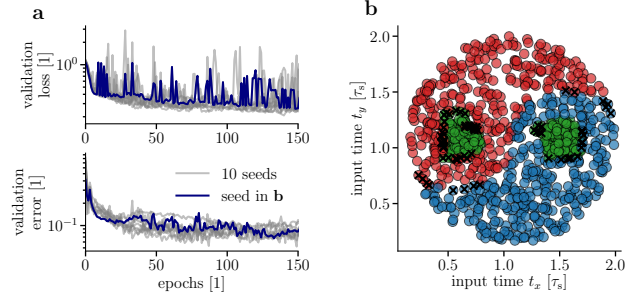


Figure SI.D1: Training on the Yin-Yang data set with a simplified learning rule. We approximated the learning rule to have less complex updates (Eqns. (SI.D1) and (SI.D2)). **(a)** shows the training process of 150 epochs. We reach a test accuracy of $(91.7 \pm 1.4)\%$ and training accuracy $(91.7 \pm 1.2)\%$ averaged over 10 seeds. **(b)** shows the classification as in Fig. 2 after training for the highlighted training in (a).

chips is limited, we investigate the possibility of approximating the derivatives Eqns. (4) and (5) by replacing the exponential functions as well as \mathcal{W} by a constant λ^1 :

$$\frac{\partial t_k}{\partial w_{ki}} = -\lambda (t_k - t_i), \quad (\text{SI.D1})$$

$$\frac{\partial t_k}{\partial t_i} = -\lambda \frac{w_{ki}}{\tau_s} (t_k - t_i - \tau_s). \quad (\text{SI.D2})$$

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 $\tau_m = \tau_s$ 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 $(91.7 \pm 1.4)\%$, this represents a small but noticeable drop in accuracy compared to the full learning rule (Table 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 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 postsynaptic potential (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

¹This effectively leads to ρ being a constant in Eqns. (39) and (40).

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 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.

SI.E.1 BrainScaleS-2

Power breakdown The full BrainScaleS-2 chip consumed a total of 175 mW measured during runtime with the INA219 chip [5]. 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 the 10 000 images of the test set.

This results in a classification every 48 μ s or equivalently a classification rate of 20 800 images per second.

Considering the relevant hardware time constants of $\tau_s \approx \tau_m \approx \dots 6 \mu$ s and the typical time to solution of around $1 \tau_s$ to $1.5 \tau_s$, a classification duration per sample of 48 μ s 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. [6] 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. [6] shorten the classification time per image to 11.8 μ s. 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.

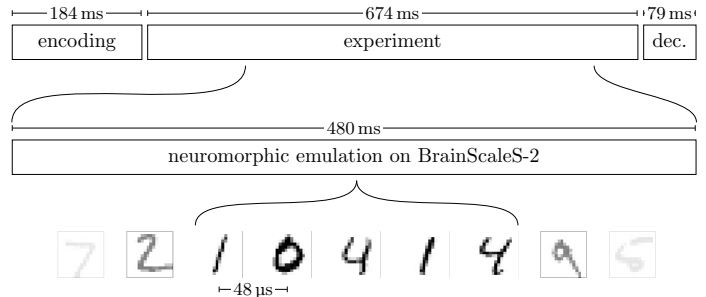


Figure SI.E1: Breakdown of the execution time for inference on the MNIST test set. The total time of about one second consists of an encoding, an experiment and a decoding phase. The encoding phase includes the translation of PyTorch tensors into spike trains and the encoding of the spike trains into 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.

SI.E.2 GPU

For comparison to conventional computing hardware we add power and classification speed measurements on a Nvidia Tesla P100 GPU to Table 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 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 μ s. This amounts to an energy-per-classification value of 852 μ J.

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 μ J.

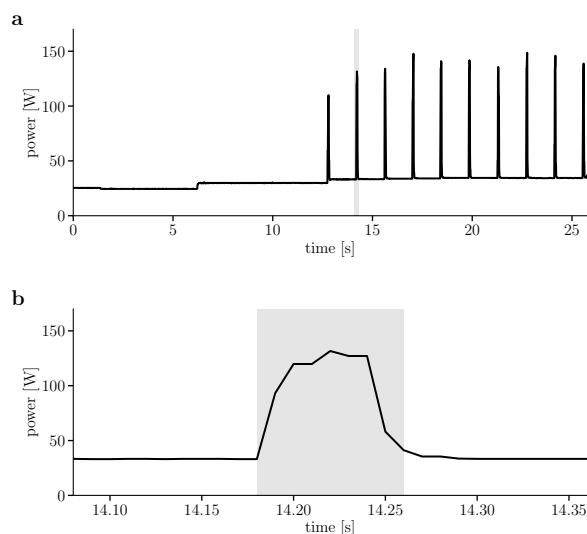


Figure SI.E2: Power consumption of Nvidia Tesla P100 GPU during classification of MNIST test data. (a) Power consumption of a standard PyTorch network for MNIST classification while running inference on the test data set for 10 times. **(b)** Zoom on a peak in the power consumption. The shaded area corresponds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

SI.F Extended literature comparison

In Table SI.F1 we provide a more comprehensive overview of neuromorphic classifiers, including references which lack energy and/or time measurements .

References

1. Schemmel, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 1947–1950 (2010).
2. Srowig, A. et al. Analog floating gate memory in a 0.18 μ m single-poly CMOS process. *Internal FACETS Documentation* (2007).
3. Koke, C. Device Variability in Synapses of Neuromorphic Circuits. *PhD thesis Heidelberg University* (2017).
4. Schmitt, S. 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)*, 2227–2234 (2017).
5. *INA219* Rev. G. Texas Instruments (Dec. 2015). <https://www.ti.com/lit/ds/symlink/ina219.pdf>.
6. Cramer, B. et al. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. *arXiv preprint arXiv:2006.07239* (2020).

Table SI.F1: 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 %	[7], 2015
True North ¹	digital	rate	CNN	0.27 μ J	1000	92.7 %	[8], 2015
True North ¹	digital	rate	CNN	108 μ J	1000	99.4 %	[8], 2015
FPGA (nLIF neurons) ²	digital	temporal	784-600-10	-	-	96.8 %	[9], 2017
unnamed (Intel) ³	digital	temporal	236-20	17.1 μ J	6250	89.0 %	[10], 2018
unnamed (Intel) ⁴	digital	temporal	784-1024-512-10	112.4 μ J	-	98.2 %	[10], 2018
unnamed (Intel) ⁴	digital	temporal	784-1024-512-10	1.7 μ J	-	97.9 %	[10], 2018
Loihi ⁵	digital	temporal	1920-10	-	-	96.4 %	[11], 2018
SPOON ⁶	digital	temporal	CNN	0.3 μ J	8547	97.5 %	[12], 2020
BrainScaleS-2	mixed	temporal	256-246-10	8.4 μ J	20 800	96.9 %	this work

¹ In [8] 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 Images preprocessed with $4 \times 5 \times 5$ Gabor filters and 3×3 pooling.

⁴ No speed measurements reported.

⁵ No energy or speed measurements reported. Images were preprocessed with an algorithm described as "using scan-line encoders".

⁶ Reported energy values are pre-silicon simulations.

7. Stomatias, E. et al. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2015). Learning with Spike-Based Retinas. *2020 IEEE International Symposium on Circuits and Systems (IS-CAS)*, 1–5 (2020).
8. Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V. & Modha, D. S. Backpropagation for energy-efficient neuromorphic computing. *Advances in Neural Information Processing Systems*, 1117–1125 (2015).
9. Mostafa, H., Pedroni, B. U., Sheik, S. & Cauwenberghs, G. Fast classification using sparsely active spiking networks. *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–4 (May 2017).
10. Chen, G. K., Kumar, R., Sumbul, H. E., Knag, P. C. & Krishnamurthy, R. K. 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**, 992–1002 (2018).
11. Lin, C.-K. et al. Programming spiking neural networks on intel's loihi. *Computer* **51**, 52–61 (2018).
12. Frenkel, C., Legat, J.-D. & Bol, D. A 28-nm Convolutional Neuromorphic Processor Enabling Online