nature machine intelligence

ARTICLES

https://doi.org/10.1038/s42256-021-00388-x

Check for updates

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

J. Göltz[1,2,4] ✉, L. Kriener[2,4] ✉, A. Baumbach[1], S. Billaudelle[1], O. Breitwieser[1], B. Cramer[1], D. Dold[1,3], A. F. Kungl[1], W. Senn[2], J. Schemmel[1], K. Meier[1] and M. A. Petrovici[1,2] ✉

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.

In recent years, the machine learning landscape has been dominated by deep learning methods. Among the benchmark problems they have managed to crack, some remained elusive for a long time[1–3]. It is thus not an exaggeration to say that deep learning dominates our understanding of 'artificial intelligence'[4–8].

Compared to the abstract neural networks used in deep learning, the more biological archetypes—spiking neural networks—still lag behind in terms of performance and scalability[9]. The reasons for this difference in success are numerous; for example, unlike abstract neurons, even an individual biological neuron represents a complex system, with finite response times, membrane dynamics and spike-based communication[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], has remained incompatible with spiking neural networks until only very recently[18,19].

Despite these challenges, spiking neural networks promise to present 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 specialized neuromorphic devices potentially more powerful—at least in principle—than the 'conventional', simple machine learning models currently used on von Neumann machines, even though this potential still remains mostly unexploited[9].

Many attempts have been made to reconcile spiking neural networks with their abstract counterparts in terms of functionality, for example, by 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 the 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 refs. [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 analogue 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 behaviour[55–57]. An early version of this work was presented in ref. [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 the afferent and efferent spike times of all neurons, it lends itself to emulation

[1]Kirchhoff-Institute for Physics, Heidelberg University, Heidelberg, Germany. [2]Department of Physiology, University of Bern, Bern, Switzerland. [3]Siemens AI lab, Siemens AG Technology, Munich, Germany. [4]These authors contributed equally: J. Göltz, L. Kriener. ✉e-mail: julian.goeltz@kip.uni-heidelberg.de; laura.kriener@unibe.ch; mihai.petrovici@unibe.ch

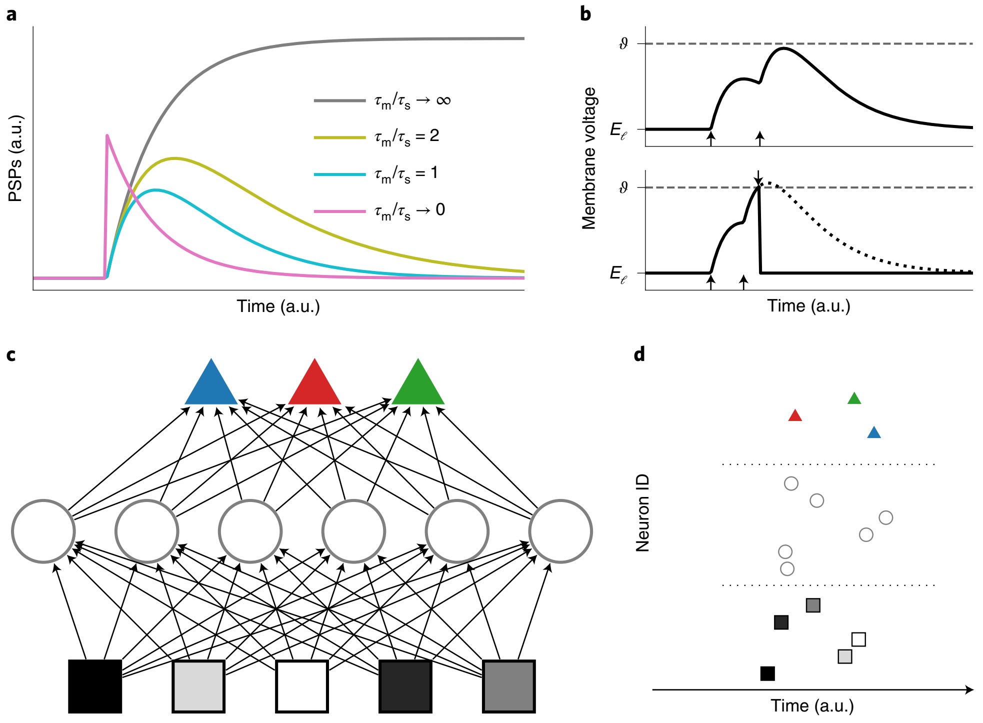

Fig. 1 | Time-to-first-spike coding and learning. **a**, Postsynaptic potential (PSP) shapes for different ratios of time constants τ_s and τ_m for single neurons. The finiteness of time constants causes the neuron to gradually forget prior input. **b**, One key challenge of this finite memory arises when small variations of the synaptic weights result in disappearing/appearing output spikes, which elicits a discontinuity in the function describing output spike timing. Plots for single neurons are shown in **a** and **b**. **c**,**d**, Application to feedforward hierarchical networks. **c**, Network structure. The geometric shape of the neurons represents a notation of their respective types (input, squares; hidden, circles; label, triangles). The shading of the input neurons (black, grey and white squares) represents the corresponding data, such as pixel brightness. The colour of the label neurons represents their respective class (blue, red, green triangles). **d**, 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 (red triangle). a.u., arbitrary units.

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 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 the effects of several types of substrate-induced distortion.

Results

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

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

with membrane capacitance C_m, leak conductance g_ℓ (from which the membrane time constant $\tau_m = C_m/g_\ell$ follows), weights w_i and spike times t_i of presynaptic neuron i, synaptic time constant τ_s and where θ is the Heaviside step function. The first sum runs over all presynaptic neurons and the second 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 modelled by clamping its membrane potential to a reset value ϱ: $u(t') = \varrho$ for $T \le t' \le T + \tau_{ref}$. For convenience and without loss of generality, we set the leak potential $E_\ell = 0$. Equation (1) can be solved analytically and yields subthreshold dynamics as described by equation (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 $\tau_m = \tau_s$) and 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 (equation (9)).

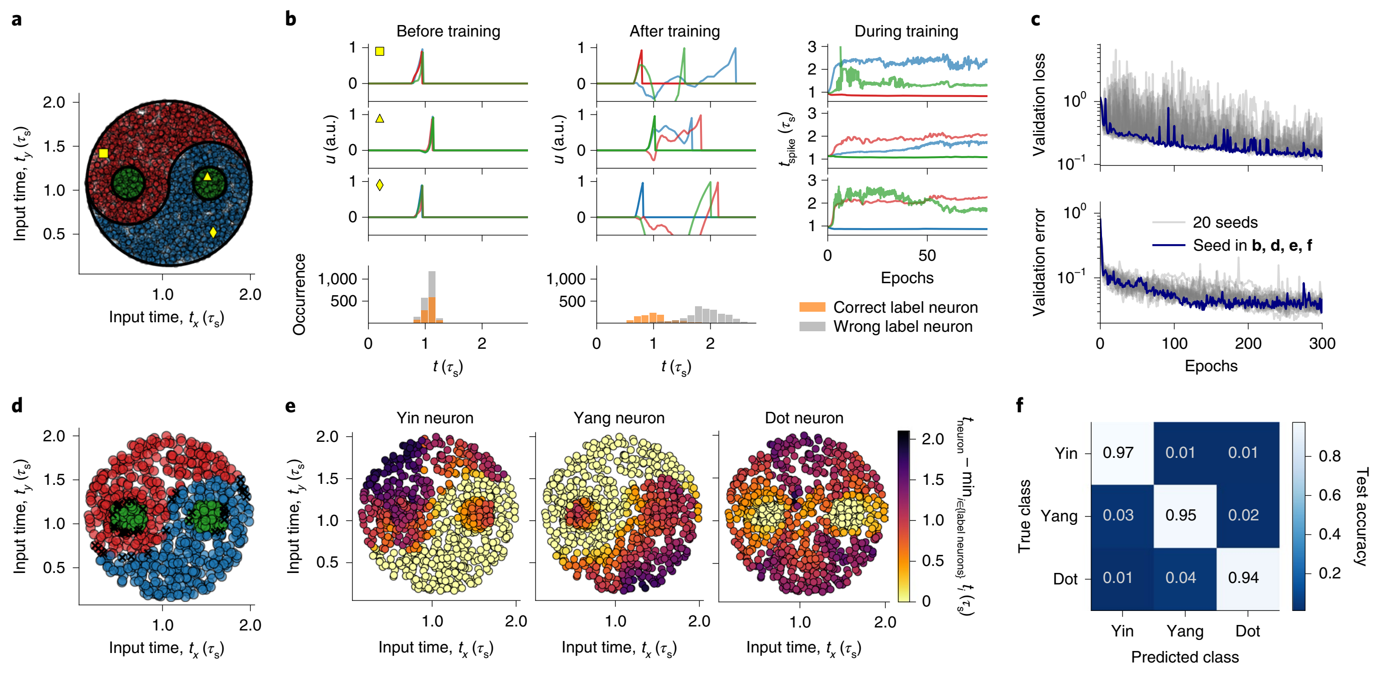

Fig. 2 | Classification of the Yin-Yang dataset. a, Illustration of the Yin-Yang dataset. The samples are separated into three classes, Yin (blue circles), Yang (red circles) and Dot (green circles). The yellow symbols (square, triangle and diamond) mark samples for which the training process is illustrated in **b**. 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. Units of τ_s indicate times that are measured in multiples of the synaptic time constant, τ_s. **b**, Training mechanism for three example 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 colour-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 equation (6) and error rate) over 300 epochs for 20 training runs with random initializations (grey). The run shown in **b** and **d**–**f** is plotted in blue. **d**, Classification result on the test set (1,000 samples). The colour 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, that is, 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.

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 the 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 ref. [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 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, for example, an input classification problem).

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 (Methods)

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

and

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

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

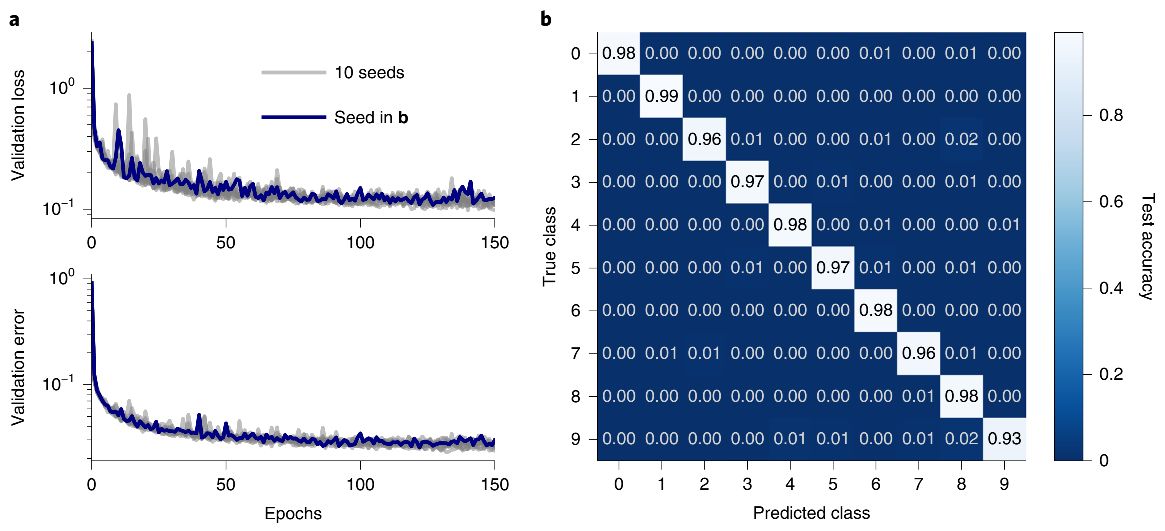

Fig. 3 | Classification of the MNIST dataset. a, Training progress of a network over 150 epochs for 10 different random initializations. The run drawn in blue is the one that produced the results in **b**. **b**, Confusion matrix for the test set after training.

$C=\{i|t_i<T\}$ (equations (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 (Methods). One inherent advantage of physical emulation is the reduction of this calculational burden.

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

Exact error backpropagation with spikes. Learning in SNNs requires the ability to relate efferent spiking to both afferent weights and spike times. For the output spike time of a neuron k with presynaptic partners i, the first relationship can be formally described by the derivative of the output spike time with respect to the presynaptic weights (equation (22)). Using certain properties of $\mathcal{W}$, we can find a simple expression that can also be made to depend on the output spike time t_k itself:

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

with a_1 and z representing functions of w_{ki} and t_i as defined in equations (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:

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

Together, equations (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 feedforward networks and their spiking activity. The input uses the same coding scheme as all other neurons: more prominent features are encoded by earlier spikes. The output of the network is defined by the identity of the label neuron that spikes first (Fig. 1d).

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

To apply the error backpropagation algorithm[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 fulfils the above requirements:

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

where $\mathbf{t}^{(N)}$ denotes the vector of label spike times $t_n^{(N)}$, n^* the index of the correct label and $\xi\in\mathbb{R}^+$ is a scaling parameter. This loss function represents a cross entropy between the true label distribution and the softmax-scaled label spike times produced by the network (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 the case of a non-spiking label neuron, we treat its spike time as $t_n^{(N)}=\infty$. In this case, however, equation (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 (Methods). In some scenarios, learning can be facilitated by the addition of a spike-time-dependent regularization term (Methods).

Gradient descent on the loss function equation (6) can now be easily performed by repeated application of the chain rule.

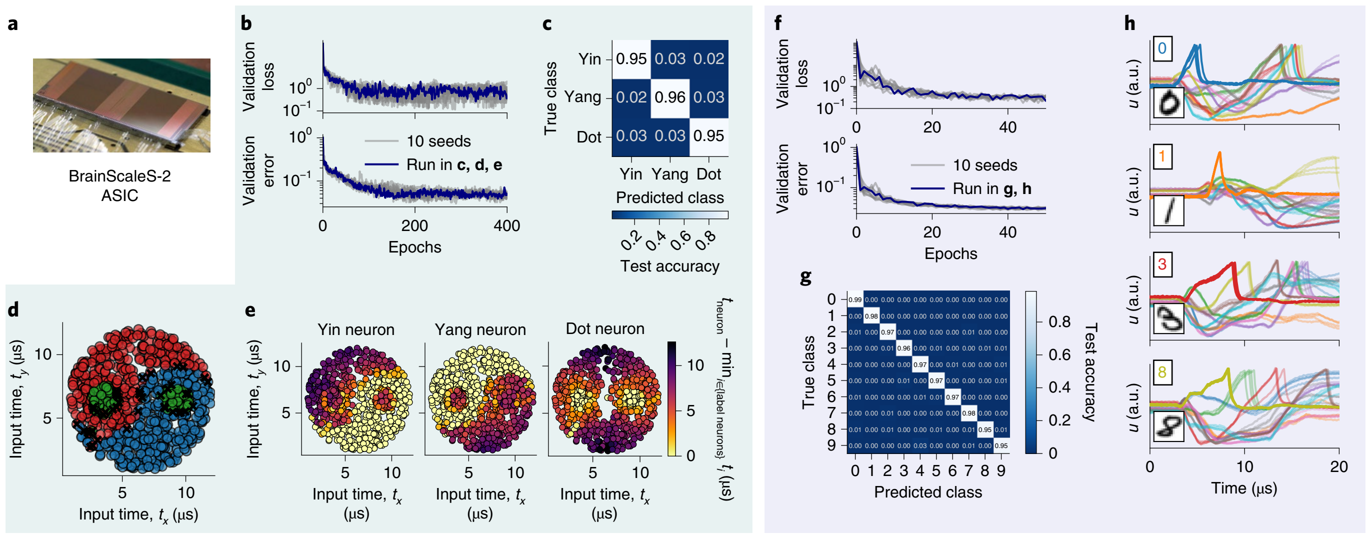

Fig. 4 | Classification on the BrainScaleS-2 neuromorphic platform. **a**, Photograph of a BrainScaleS-2 chip. **b**–**e**, Results for the Yin-Yang dataset. **b**, Training progress over 200 epochs for 11 different random initializations. The run drawn in blue also produced the results shown in **c**–**e**. **c**, Confusion matrix for the test set after training. **d**, Classification result on the test set. For each input sample the colour 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 dataset. **f**–**h**, Results for the MNIST dataset. **f**, Evolution of training over 50 epochs for 10 different random initializations. The run drawn in blue is the one that produced the results shown in **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 colour-coded voltage traces of four label neurons for one input that was presented repeatedly to the network (the insets show the input and its correct class). Each trace was recorded four times to highlight the trial-to-trial variations.

Using the exact derivatives, equations (4) and (5), this yields the synaptic plasticity rule

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

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

A compact formulation for hierarchical networks that highlights the backpropagation of errors can be found in equations (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 equation (2) or by simulating (or emulating) the LIF dynamics (equation (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 Supplementary Section D we present early results using such an approach. Note that the simplification is only used in Supplementary Section D and all other results we report in the following were produced using the full analytical equations (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 datasets 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 dataset[65]. For higher-dimensional, discrete input, we used the MNIST dataset[66] as a small-scale image classification scenario.

The results in this section are based on equation (2) for calculating the spike times in the forward pass, and equation (40) for calculating weight updates. Details regarding implementation are provided in the Methods. For the hyperparameters of the discussed experiments, see Supplementary Tables F1 and F2.

Yin-Yang classification task. The first dataset 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 yin-yang motive, the dataset 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 labelled according to the respective area they occupy, that is, Yin, Yang or Dot. This augmented dataset was specifically designed to require latent variables for classification: a shallow non-spiking classifier reaches (64.3 ± 0.2)% test accuracy, an ANN with one hidden layer of size 120 typically around (98.7 ± 0.3)%. Because of this large gap, our Yin-Yang dataset 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 (Fig. 1 and Methods), they were joined with a bias spike at fixed time, and these five spikes served as input to a network with 120 hidden and three 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. the 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 ± 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

Table 1 | Summary of the presented results

Dataset	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

Accuracies are given as mean and standard deviation. Results are distinguished between software simulations (SW) and hardware emulations (HW). For comparison, on the Yin-Yang dataset 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 dataset 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.

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).

MNIST classification task. To study the scalability of our approach to larger and more high-dimensional datasets, we applied it to the classification of MNIST handwritten digits[66]. Figure 3 shows training results for networks with 784-350-10 neurons (input-hidden-label layer size), 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 ± 0.1)%. Similar results are also achieved for deeper architectures with multiple hidden layers (Supplementary Table B1 provides additional simulation runs with different network architectures).

For reference, we consider several other results obtained with spiking-time coding. In ref. [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 and Masquelier[54] report 97.4% using 400 hidden neurons. In ref. [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 straightforward 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 1,000 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, see Supplementary Section 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 analogue circuit emulating LIF neuronal dynamics with an acceleration factor of 10^3 to 10^4 compared to biological timescales.

Owing 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[70], 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[33,34,63,71].

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 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 set-up are available in the Methods.

We used an in-the-loop training approach[23,33,72], where inference runs emulated on the neuromorphic substrate were interleaved with host-based weight update calculations. For emulating the forward pass, the spike times for each sample in a mini-batch were joined sequentially into one long spike train and then injected into the neuromorphic system via a field-programmable gate array (FPGA). The latter was also used to record the spikes emitted by the hidden and label layers.

Figure 4a–d shows the results of training a spiking network with 120 hidden neurons on BrainScaleS-2 on the Yin-Yang dataset. The system quickly learned to discriminate between the presented patterns, with an average test accuracy of (95.0 ± 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. Fig. 4h). Considering that one of the specific challenges built into the Yin-Yang dataset 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 dataset using the BrainScaleS-2 system, we emulated and trained a network of size 256-246-10 (Fig. 4f–h). Owing 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 ± 0.1)%; similarly to the Yin-Yang dataset, this is only slightly

Table 2 | Comparison of pattern recognition models on the MNIST dataset emulated on neuromorphic back-ends, sorted by classification speed

Platform	Type	Technology	Coding	Input resolution	Network size/ structure	Data augmentation/ regularization	Energy per classification	Classifications per second[a]	Test accuracy (%)	Ref. (year)
Nvidia Tesla P100	Digital	14 nm	ANN	28×28	CNN[b]	Dropout	852 μJ	125,000	99.2	Supplementary Section SI.E.2
SpiNNaker	Digital	130 nm	Rate	28×28	784-600-500-10	Noisy input encoding	3.3 mJ	91	95.0	[82] (2015)
True North	Digital	28 nm	Rate	28×28	CNN	Noisy input encoding	0.27 μJ	1,000	92.7	[61] (2015)
True North	Digital	28 nm	Rate	28×28	CNN	Noisy input encoding	108 μJ	1,000	99.4	[61] (2015)
Loihi	Digital	14 nm	Bin. rate	(20×20)[c]	400-400-10	Not available	2.5 μJ	5,917	96.2	[83] (2021)
Unnamed (Intel)	Digital	10 nm	Temporal	(28×28)[d]	236-20	Stochastic spike loss	1.0 μJ	6,250	88.0	[84] (2018)
BrainScaleS-2	Mixed	65 nm	Temporal	16×16	256-246-10	Input noise	8.4 μJ	20,800	96.9	This work; Supplementary Section SI.E.1

[a]Note that some platforms achieve a high number of classifications per second simply by processing a large number of samples in parallel, while other platforms rely on the sequential (but fast) processing of individual samples. [b]Standard architecture given as an example in the PyTorch repository, for details see Supplementary Section SI.E.2. [c]Four (empty) pixels on each margin are cropped to yield the 20×20 centre from the 28×28 image. [d]The 28×28 image is preprocessed using 5×5 Gabor filters and 3×3 pooling before being sent into the chip. For reference, an ANN running on a graphics processing unit is included in the top row. Note that we include only references that present measurements for both energy and throughput in addition to accuracy. An extended table containing results with partial or estimated measurements is provided as Supplementary Table F3.

lower than in software simulations of equally sized networks (Table 1). 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 μs and is clearly separable from all other label neurons.

Because of its short intrinsic time constants and overall energy efficiency, the BrainScaleS-2 system enables very fast and energy-efficient acquisition of classification results. Classification of the 10,000 MNIST test samples takes a total of 0.937 s, including data transmission, emulation of dynamics and return of the classification results. The total time on the BrainScaleS-2 chip was 480 ms (a detailed breakdown of the execution time is shown in Supplementary Section E). The power consumption of the chip, measured during runtime, including all chip components needed for spike generation and processing (that is, excluding the host and FPGA) amounted to 175 mW. For measurement details and scalability considerations we refer to Supplementary Section E. This results in an average energy consumption of 8.4 μJ per classification. Table 2 provides a comparison to other neuromorphic platforms.

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, is provided in Supplementary Table F3.

Our current experimental set-up leaves room for substantial optimization. For an estimation of possible improvements and their potential effect on classification rate and energy consumption, see Supplementary Section E and ref. [72]. 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 three.

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 (that is, spike times, as opposed to, for example, 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 comprehensive study of the robustness of our approach.

Most physical neuronal substrates have several forms of variability in common (chapter 5 in ref. [73]). In both digital and mixed-signal systems, synaptic weights are typically limited in both range and resolution. Additionally, the parameters of analogue 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 equation (4). The output spike time given in equation (2) depends only on neuron parameters, presynaptic spike times and weights, so its derivatives share the same dependencies (equations (22) and (23)). With some manipulations, the equation for the actual output spike time can be inserted (equations (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 substantially more stable, which is confirmed below.

Using dimensionless weight units (scaled by the inverse threshold), we observe that an upper weight limit of ~3 is sufficient for achieving peak performance (Fig. 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 (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[74]. Our model shows approximately constant performance for weight

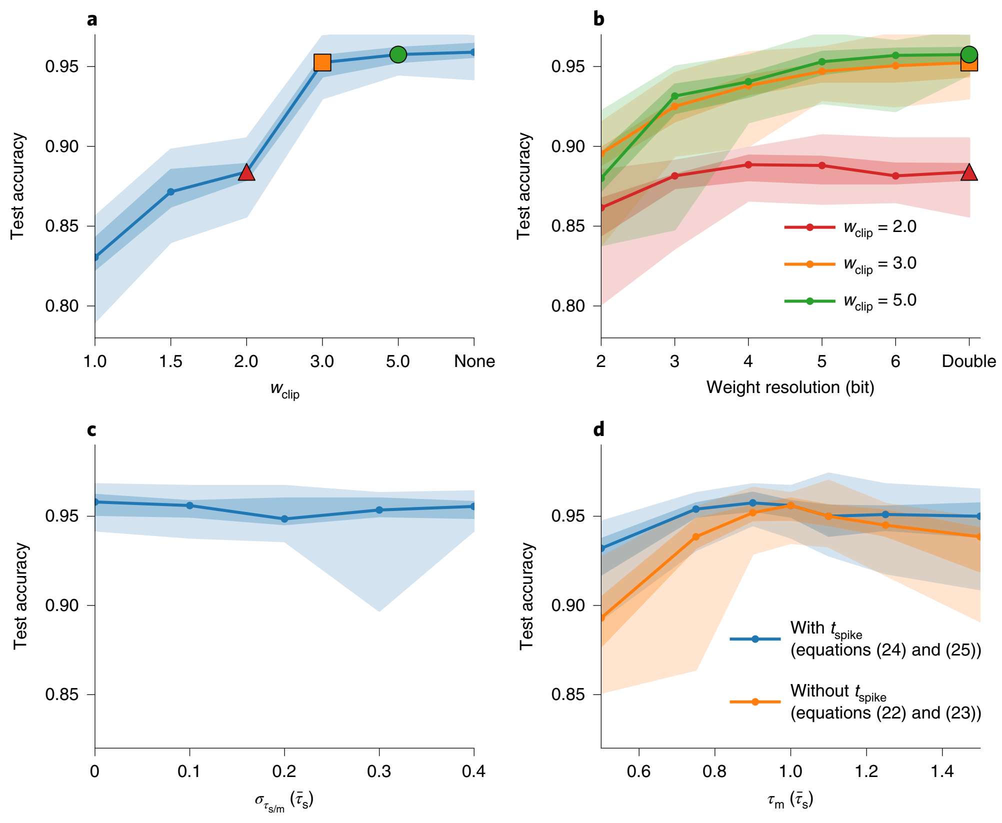

Fig. 5 | Effects of substrate imperfections. Modelled constraints were added artificially into simulated networks. All panels show the median, quartiles, minimum and maximum of the final test accuracy on the Yin-Yang dataset for 20 different initializations. **a**, Limited weight range. The weights were 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 **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 set-up where the interval $[-w_{clip}, w_{clip}]$ is discretized into $2 \times 2^n - 1$ 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 normal 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 'naïve' 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.

resolutions down to 5 bit, followed by gradual degradation below (Fig. 5b).

Interestingly, adding variability to the synapse and membrane time constants has no discernible effects (Fig. 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 'naïve' gradient descent without this information is shown in Fig. 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 Section C we investigate trained networks regarding their robustness to adverse effects that appear only after training, such as temperature-induced parameter variations or inactivation of neurons. Our simulations show that trained networks can cope with such effects, suggesting that our training algorithm develops network structures robust even to distortions not present during training.

Finally, we note that all of the effects addressed above also have biological correlates. Although 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.

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 feed-forward topologies, it yields a spike-based form of error backpropagation. In this Article, 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.

Although 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, for example, 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 (refs. [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 substantially 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 ~10 µs after receiving the first spike. Including relaxation between patterns and communication, the complete MNIST test set with 10,000 samples is classified in less than 1 s with an energy consumption of ~8.4 µJ per classification, which compares favourably 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, for example, graphics processing units, 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 substantially 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 postsynaptic spikes, as in classical spike-timing-dependent plasticity, and an error term, which could be propagated by mechanisms such as those proposed in, for example, refs. [75,76]. This raises the intriguing possibility for our framework to help explain learning in biological substrates as well.

Because, 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, 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 Article, starting with the learning rule for $\tau_\mathrm{m} \to \infty$, then $\tau_\mathrm{m} = \tau_\mathrm{s}$, equation (2) and finally $\tau_\mathrm{m} = 2\tau_\mathrm{s}$, equation (3). The case $\tau_\mathrm{m} \to \infty$ has already been discussed in ref. [53] and was reproduced here for completeness and comparison. Owing to the symmetry in τ_m and τ_s of the PSP (equation (14)), the $\tau_\mathrm{m} = 2\tau_\mathrm{s}$ case describes the $\tau_\mathrm{m} = \frac{1}{2}\tau_\mathrm{s}$ case as well.

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

$$u(T) = \vartheta, \tag{8}$$

has to be found, given LIF dynamics

$$u(t) = \frac{1}{C_\mathrm{m}} \frac{\tau_\mathrm{m}\tau_\mathrm{s}}{\tau_\mathrm{m} - \tau_\mathrm{s}} \sum_{\text{spikes } t_i} w_i \kappa(t - t_i), \tag{9}$$

$$\kappa(t) = \theta(t)\left[\exp\left(-\frac{t}{\tau_\mathrm{m}}\right) - \exp\left(-\frac{t}{\tau_\mathrm{s}}\right)\right], \tag{10}$$

with membrane time constant $\tau_\mathrm{m} = C_\mathrm{m}/g_\ell$ and the PSP kernel κ given by a difference of exponentials. Here we already assumed our TTFS use case in which each neuron only produces one relevant spike and the second sum in equation (1) reduces to a single term.

For convenience, we use the following definitions:

$$a_n := \sum_{i \in C} w_i \exp\left(\frac{t_i}{n\tau_\mathrm{s}}\right), \tag{11}$$

$$b := \sum_{i \in C} w_i \frac{t_i}{\tau_\mathrm{s}} \exp\left(\frac{t_i}{\tau_\mathrm{s}}\right), \tag{12}$$

with summation over the set of causal presynaptic spikes $C = \{i | t_i < T\}$.

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 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 $T = \infty$.

The nLIF learning rule for $\tau_\mathrm{m} \to \infty$. With this choice of τ_m, the first term in equation (10) becomes 1 and we recover the nLIF case discussed in ref. [53]. Given the existence of an output spike, in equation (8) the spike time T appears only in one place and simple reordering yields

$$\frac{T}{\tau_\mathrm{s}} = \ln\left[\frac{a_1}{a_\infty - \vartheta C_\mathrm{m}/\tau_\mathrm{s}}\right], \tag{13}$$

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

The learning rule for $\tau_\mathrm{m} = \tau_\mathrm{s}$. According to l'Hôpital's rule, in the limit $\tau_\mathrm{m} \to \tau_\mathrm{s}$, equation (9) becomes a sum over α-functions of the form

$$u(t) = \frac{1}{C_\mathrm{m}} \sum_i w_i \theta(t - t_i)(t - t_i) \exp\left(-\frac{t - t_i}{\tau_\mathrm{s}}\right). \tag{14}$$

Using these voltage dynamics for the equation of the spike time equation (8), together with the definitions of equations (11) and (12) and $\tau_\mathrm{m} = C_\mathrm{m}/g_\ell$, we get the equation

$$0 = g_\ell \vartheta \exp\left(\frac{T}{\tau_\mathrm{s}}\right) + \underbrace{b - a_1 \frac{T}{\tau_\mathrm{s}}}_{=:\, y}. \tag{15}$$

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

$$h \exp(h) = z \tag{16}$$

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

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

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

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 (Supplementary Fig. F1a,b). Correspondingly, the Lambert W function (Supplementary Fig. F1c,d) has two real branches (in addition to infinite imaginary ones), and we need to choose the branch that returns the earlier solution. In case the voltage is only tangent to the threshold at its maximum, the Lambert W function only has one solution.

For choosing the branch in the other cases we need to look at h from the definition, that is

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

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 $T \le t_i + \tau_s$ and therefore

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

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

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

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) = \frac{\tau_s}{a_1}\exp\left(\frac{t_i}{\tau_s}\right)\left[z\mathcal{W}'(z) + \left(\frac{t_i}{\tau_s} - \frac{b}{a_1}\right)\left(1 - z\mathcal{W}'(z)\right)\right], \quad (22)$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) = \frac{w_i}{a_1}\exp\left(\frac{t_i}{\tau_s}\right)\left[1 + \left(\frac{t_i}{\tau_s} - \frac{b}{a_1}\right)\left(1 - z\mathcal{W}'(z)\right)\right]. \quad (23)$$

A crucial step is to reinsert the definition of the spike time where 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 equation (16) with $h = \mathcal{W}(z)$ with respect to z. With this equation one can calculate the derivative of equation (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 equations (4) and (5) shown in the main text.

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

$$\begin{aligned} g_\ell \vartheta &= \exp\left(-\tfrac{T}{\tau_m}\right)\sum_{i\in C} w_i \exp\left(\tfrac{t_i}{\tau_m}\right) \\ &\quad - \exp\left(-\tfrac{T}{\tau_s}\right)\sum_{i\in C} w_i \exp\left(\tfrac{t_i}{\tau_s}\right). \end{aligned} \quad (26)$$

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

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

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 $x = \sqrt{a_2^2 - 4a_1 g_\ell \vartheta}$ for brevity, the derivatives of equation (28) are

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}) = 2\tau_s\left[\frac{1}{a_1} + \frac{2g_\ell\vartheta}{(a_2 + x)x}\right]\exp\left(\frac{t_i}{\tau_s}\right) - \frac{2\tau_s}{x}\exp\left(\frac{t_i}{2\tau_s}\right), \quad (29)$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}) = 2w_i\left[\frac{1}{a_1} + \frac{2g_\ell\vartheta}{(a_2 + x)x}\right]\exp\left(\frac{t_i}{\tau_s}\right) - \frac{w_i}{x}\exp\left(\frac{t_i}{2\tau_s}\right). \quad (30)$$

Again, inserting the output spike time yields

$$\frac{\partial T}{\partial w_i}(\mathbf{w}, \mathbf{t}, T) = \frac{2\tau_s}{a_1}\left[1 + \frac{g_\ell\vartheta}{x}\exp\left(\frac{T}{2\tau_s}\right)\right]\exp\left(\frac{t_i}{\tau_s}\right) - \frac{2\tau_s}{x}\exp\left(\frac{t_i}{2\tau_s}\right), \quad (31)$$

$$\frac{\partial T}{\partial t_i}(\mathbf{w}, \mathbf{t}, T) = \frac{2w_i}{a_1}\left[1 + \frac{g_\ell\vartheta}{x}\exp\left(\frac{T}{2\tau_s}\right)\right]\exp\left(\frac{t_i}{\tau_s}\right) - \frac{w_i}{x}\exp\left(\frac{t_i}{2\tau_s}\right). \quad (32)$$

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 equations (24) and (25) in a layer-wise setting, the derivatives of the spike time for a neuron k in arbitrary layer l are

$$\frac{\partial t_k^{(l)}}{\partial w_{ki}^{(l)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) = -\frac{1}{a_1}\exp\left(\frac{t_i^{(l-1)}}{\tau_s}\right)\frac{1}{\mathcal{W}(z)+1}\left(t_k^{(l)} - t_i^{(l-1)}\right), \quad (36)$$

$$\frac{\partial t_k^{(l)}}{\partial t_i^{(l-1)}}(\mathbf{w}, \mathbf{t}^{(l-1)}, \mathbf{t}^{(l)}) = -\frac{1}{a_1}\exp\left(\frac{t_i^{(l-1)}}{\tau_s}\right)\frac{1}{\mathcal{W}(z)+1}\frac{w_{ki}^{(l)}}{\tau_s}\left(t_k^{(l)} - t_i^{(l-1)} - \tau_s\right). \quad (37)$$

Inserting equations (35) to (37) into equations (33) and (34) yields a synaptic learning rule that implements exact error backpropagation on spike times.

This learning rule can be rewritten to resemble the standard error backpropagation algorithm for ANNs:

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

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

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

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

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

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

BrainScaleS-2. The application-specific integrated circuit (ASIC) is built around an analogue neuromorphic core that 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 analogue circuits, this emulation takes place at 1,000-fold accelerated timescales 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[77]. Both the membrane and synaptic time constants were calibrated to 6 μs.

Each neuron circuit is connected to one of four synapse matrices on the chip, and integrates stimuli from its column of 256 CuBa synapses[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 and 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 an FPGA to handle 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 also 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[78] that exposes the necessary interfaces to a high-level user via Python bindings. These were used in our framework, which is described in the following.

Simulation software. Our experiments were performed using custom modules for the deep learning library PyTorch[79]. The network module implements layers of LIF neurons whose spike times are calculated according to equation (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 (equation (40)) are realized in a custom backward-pass module for the network.

Training and regularization methods. To train a given dataset using our learning framework, the input data have 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 dataset requires a bias to be solvable, our framework allows its addition. These bias spikes essentially represent additional input spikes for a layer, which have the same spike time for any input. The weights from the neurons to these 'bias sources' are learned in the same way as all the other synaptic weights. For the Yin-Yang dataset, 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 dataset has a much higher input dimensionality and the spikes are more distributed over the input time interval. Therefore, the activity provided to the hidden layer at any point in time is high, even without additional bias.

Implementing our learning algorithm as custom PyTorch modules allows us to use the training architecture provided by the library. The simulations were performed using mini-batch training in combination with with the Adam optimizer[80] and learning rate scheduling (the parameters are provided in Supplementary Tables F1 and F2).

To assist learning we employ several regularization techniques. The term $+\alpha\left[\exp\left(t_{n^*}^{(N)}/\beta\tau_{\mathrm{s}}\right)-1\right]$ with scaling parameters $\alpha, \beta \in \mathbb{R}^+$, can be added to the loss in equation (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 dataset.

Weight updates Δw with absolute value larger than a given hyperparameter are set to zero to compensate divergence for vanishing denominator in equation (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 that 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 the 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 (1) can receive a set of input spikes and yield the output spike times of all neurons in the emulated network and (2) 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. Because the only information-carrying signal that is not reset upon firing is the synaptic current, which is forgotten on the timescale of τ_{s}, we found that, in practice, setting the refractory time $\tau_{\text{ref}} > \tau_{\mathrm{s}}$ leads to most neurons eliciting only one spike before the classification of a given input pattern.

For training the Yin-Yang dataset on BrainScaleS-2, having only five inputs proved insufficient due to the combination of limited weights and neuron variability. We therefore multiplexed each logical input into five physical spike sources, totalling 25 inputs spikes per pattern. Adding further copies of the inputs effectively increased the weights for each individual input. This method has the added benefit of averaging out some of the effects of the fixed-pattern noise on the input circuits as multiple of them are employed for the same task.

Data availability

We used the MNIST[66] and the Yin-Yang dataset[65]. For the latter, see https://github.com/lkriener/yin_yang_data_set.

Code availability

Code for the simulations[81] is available at https://github.com/JulianGoeltz/fastAndDeep.

Received: 17 November 2020; Accepted: 30 July 2021;
Published online: 17 September 2021

References

1. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* 1097–1105 (NIPS, 2012).
2. Silver, D. et al. Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017).
3. Brown, T. B. et al. Language models are few-shot learners. Preprint at https://arxiv.org/pdf/2005.14165.pdf (2020).
4. Brooks, R., Hassabis, D., Bray, D. & Shashua, A. Is the brain a good model for machine intelligence?. *Nature* **482**, 462–463 (2012).
5. Ng, A. What artificial intelligence can and can't do right now. *Harvard Business Review* (9 November 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. *Nat. Neurosci.* **22**, 1761–1770 (2019).
9. Pfeiffer, M. & Pfeil, T. Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci* **12**, 774 (2018).
10. Gerstner, W. What is different with spiking neurons? In *Plausible Neural Networks for Biological Modelling. Mathematical Modelling: Theory and Applications* Vol 13. (eds Mastebroek, H. A. K. & Vos, J. E.) 23–48 (Springer, 2001).
11. Izhikevich, E. M. Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* **15**, 1063–1070 (2004).
12. Gerstner, W. *Spiking Neurons* (MIT Press, 1998).
13. Maass, W. Searching for principles of brain computation. *Curr. Opin. Behav. Sci.* **11**, 81–92 (2016).
14. Davies, M. Benchmarks for progress in neuromorphic computing. *Nat. Mach. Intell.* **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. In *System Modeling and Optimization. Lecture Notes in Control and Information Sciences* Vol. 38 (eds Drenick, R. F. & Kozin, F.) 762–770 (Springer, 1982).

17. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
18. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. *Neural Netw* **111**, 47–63 (2018).
19. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process.* **36**, 51–63 (2019).
20. Gütig, R. & Sompolinsky, H. The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* **9**, 420–428 (2006).
21. Cao, Y., Chen, Y. & Khosla, D. Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* **113**, 54–66 (2015).
22. Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *Proc. 2016 IEEE International Conference on Rebooting Computing* (*ICRC*) 1–8 (IEEE, 2016).
23. Schmitt, S. et al. Neuromorphic hardware in the loop: training a deep spiking network on the BrainScaleS wafer-scale system. In *Proc. 2017 International Joint Conference on Neural Networks* (*IJCNN*) 2227–2234 (2017).
24. Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., & Li, H. Deep spiking neural network with spike count based learning rule. In *International Joint Conference on Neural Networks* 1–6 (IEEE, 2019).
25. Thakur, C. S. T. et al. Large-scale neuromorphic spiking array processors: a quest to mimic the brain. *Front. Neurosci.* **12**, 891 (2018).
26. Mead, C. Neuromorphic electronic systems. *Proc. 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. Preprint at https://arxiv.org/pdf/1311.3211.pdf (2013).
29. Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K. & Cauwenberghs, G. Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* **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. *Phys. Rev. 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. *Front. Neurosci.* **10**, 241 (2016).
32. Leng, L. et al. Spiking neurons with short-term synaptic plasticity form superior generative networks. *Sci. Rep.* **8**, 10651 (2018).
33. Kungl, A. F. et al. Accelerated physical emulation of Bayesian inference in spiking neural networks. *Front. Neurosci.* **13**, 1201 (2019).
34. Dold, D. et al. Stochasticity from function-why the Bayesian brain may need no noise. *Neural Netw.* **119**, 200–213 (2019).
35. Jordan, J. et al. Deterministic networks for probabilistic computing. *Sci. Rep.* **9**, 18303 (2019).
36. Hunsberger, E. & Eliasmith, C. Training spiking deep networks for neuromorphic hardware. Preprint at https://arxiv.org/pdf/1611.05141.pdf (2016).
37. Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J. & Masquelier, T. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* **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 Netw* **118**, 90–101 (2019).
39. Bohte, S. M., Kok, J. N. & La Poutré, J. A. Spikeprop: backpropagation for networks of spiking neurons. In *8th European Symposium on Artificial Neural Networks* 419–424 (2000).
40. Zenke, F. & Ganguli, S. Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* **30**, 1514–1541 (2018).
41. Huh, D. & Sejnowski, T. J. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems* Vol. 31, 1433–1443 (NIPS, 2018).
42. Thorpe, S., Delorme, A. & Van Rullen, R. Spike-based strategies for rapid processing. *Neural Netw.* **14**, 715–725 (2001).
43. Thorpe, S., Fize, D. & Marlot, C. Speed of processing in the human visual system. *Nature* **381**, 520–522 (1996).
44. Johansson, R. S. & Birznieks, I. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nat. Neurosci.* **7**, 170–177 (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. In *Proc. 2010 IEEE International Symposium on Circuits and Systems* 1947–1950 (IEEE, 2010).
47. Akopyan, F. et al. TrueNorth: design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integrated Circuits Syst.* **34**, 1537–1557 (2015).
48. Billaudelle, S. et al. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. In *IEEE International Symposium on Circuits and Systems* 1–5 (IEEE, 2020).
49. Davies, M. et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**, 82–99 (2018).
50. Mayr, C., Höppner, S., & Furber, S. SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018* 277–280 (IOS Press, 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 Trans. Biomed. Circuits Syst.* **12**, 106–122 (2017).
53. Mostafa, H. Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **29**, 3227–3235 (2017).
54. Kheradpisheh, S. R. & Masquelier, T. S4NN: temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* **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. *J. Neurophysiol.* **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. *Nat. Commun.* **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 (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 Trans. Biomed. Circuits Syst.* **11**, 128–142 (2017).
60. Prodromakis, T. & Toumazou, C. A review on memristive devices and applications. In *Proc. 2010 17th IEEE International Conference on Electronics, Circuits and Systems* 934–937 (IEEE, 2010).
61. Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V. & Modha, D. S. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems* 1117–1125 (NIPS, 2015).
62. van De Burgt, Y., Melianas, A., Keene, S. T., Malliaras, G. & Salleo, A. Organic electronics for neuromorphic computing. *Nat. Electron.* **1**, 386–397 (2018).
63. Wunderlich, T. et al. Demonstrating advantages of neuromorphic computation: a pilot study. *Front. Neurosci.* **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**, 208–214 (2019).
65. Kriener, L., Göltz, J. & Petrovici, M. A. The yin-yang dataset. Preprint at https://arxiv.org/pdf/2102.08211.pdf (2021).
66. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
67. Schemmel, J., Billaudelle, S., Dauer, P. & Weis, J. Accelerated analog neuromorphic computing. Preprint at https://arxiv.org/pdf/2003.11996.pdf (2020).
68. Comsa, I. M. et al. Temporal coding in spiking neural networks with alpha synaptic function. In *Proc. 2020 IEEE International Conference on Acoustics, Speech and Signal Processing* (*ICASSP*) 8529–8533 (IEEE, 2020).
69. Tavanaei, A., Kirby, Z. & Maida, A. S. Training spiking ConvNets by STDP and gradient descent. In *Proc. 2018 International Joint Conference on Neural Networks* (*IJCNN*) 1–8 (IEEE, 2018).
70. Aamir, S. A. et al. An accelerated LIF neuronal network array for a large-scale mixed-signal neuromorphic architecture. *IEEE Trans. Circuits Syst. I Regular Papers* **65**, 4299–4312 (2018).
71. Petrovici, M. A. et al. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PLoS ONE* **9**, e108590 (2014).
72. Cramer, B. et al. Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate. Preprint at https://arxiv.org/pdf/2006.07239.pdf (2020).
73. Petrovici, M. A. *Form Versus Function: Theory and Models for Neuronal Substrates* (Springer, 2016).
74. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* **18**, 6869–6898 (2017).
75. Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. Preprint at *bioRxiv* https://doi.org/10.1101/2020.03.30.015511 (2020).
76. Sacramento, J., Ponte Costa, R., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems* Vol. 31, 8721–8732 (NIPS, 2018).
77. Aamir, S. A. et al. A mixed-signal structured AdEx neuron for accelerated neuromorphic cores. *IEEE Trans. Biomed. Circuits Syst.* **12**, 1027–1037 (2018).
78. Müller, E. et al. Extending BrainScaleS OS for BrainscaleS-2. Preprint at https://arxiv.org/pdf/2003.13750.pdf (2020).

79. Paszke, A. et al. PyTorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* Vol. 32, 8024–8035 (NIPS, 2019).
80. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. Preprint at https://arxiv.org/pdf/1412.6980.pdf (2014).
81. Göltz, J. et al. Fast and energy-efficient neuromorphic deep learning with first-spike times (Zenodo, 2021); https://doi.org/10.5281/zenodo.5115007
82. Stromatias, E. et al. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on SpiNNaker. In *Proc. 2015 International Joint Conference on Neural Networks* (*IJCNN*) 1–8 (2015).
83. Renner, A., Sheldon, F., Zlotnik, A., Tao, L. & Sornborger, A. The backpropagation algorithm implemented on spiking neuromorphic hardware. Preprint at https://arxiv.org/pdf/2106.07030.pdf (2021).
84. 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 J. Solid State Circuits* **54**, 992–1002 (2018).

Acknowledgements
We thank J. Jordan and N. Gürtler for valuable discussions, S. Schmitt for assistance with BrainScaleS-1, V. Karasenko, P. Spilger and Y. Stradmann for taming physics, as well as M. Davies and Intel for their ongoing support (L.K., W.S., M.A.P.). 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 grant agreement no. 800858. Some simulations were performed on the bwForCluster NEMO, supported by the state of Baden–Württemberg through bwHPC and the German Research Foundation (DFG) through grant no. INST 39/963-1 FUGG. We gratefully acknowledge funding from the European Union for the Human Brain Project under grant agreements 604102 (J.S., K.M., M.A.P.), 720270 (S.B., O.B., B.C., J.S., K.M., M.A.P.), 785907 (S.B., O.B., B.C., W.S., J.S., K.M., M.A.P.), 945539 (L.K., A.B., S.B., O.B., B.C., W.S., J.S., M.A.P.) and the Manfred Stärk Foundation (J.G., A.B., D.D., A.F.K., K.M., M.A.P.).

Author contributions
J.G., A.B. and M.A.P. designed the conceptual and experimental approach. J.G. derived the theory, implemented the algorithm and performed the hardware experiments. L.K. embedded the algorithm into a comprehensive training framework and performed the simulation experiments. A.B. and O.B. offered substantial software support. S.B., B.C., J.G. and A.F.K. provided low-level software for interfacing with the hardware. J.G., L.K., D.D., S.B. and M.A.P. wrote the manuscript.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-021-00388-x.

Correspondence and requests for materials should be addressed to J. Göltz, L. Kriener or M. A. Petrovici.

Peer review information *Nature Machine Intelligence* thanks the anonymous reviewers for their contribution to the peer review of this work.