nature communications

Article

https://doi.org/10.1038/s41467-024-53827-9

The backpropagation algorithm implemented on spiking neuromorphic hardware

Received: 11 April 2024

Accepted: 22 October 2024

Published online: 08 November 2024

Check for updates

Alpha Renner [1,2], Forrest Sheldon [3,4], Anatoly Zlotnik [5], Louis Tao [6,7] & Andrew Sornborger [8] ✉

The capabilities of natural neural systems have inspired both new generations of machine learning algorithms as well as neuromorphic, very large-scale integrated circuits capable of fast, low-power information processing. However, it has been argued that most modern machine learning algorithms are not neurophysiologically plausible. In particular, the workhorse of modern deep learning, the backpropagation algorithm, has proven difficult to translate to neuromorphic hardware. This study presents a neuromorphic, spiking backpropagation algorithm based on synfire-gated dynamical information coordination and processing implemented on Intel's Loihi neuromorphic research processor. We demonstrate a proof-of-principle three-layer circuit that learns to classify digits and clothing items from the MNIST and Fashion MNIST datasets. To our knowledge, this is the first work to show a Spiking Neural Network implementation of the exact backpropagation algorithm that is fully on-chip without a computer in the loop. It is competitive in accuracy with off-chip trained SNNs and achieves an energy-delay product suitable for edge computing. This implementation shows a path for using in-memory, massively parallel neuromorphic processors for low-power, low-latency implementation of modern deep learning applications.

Spike-based learning in plastic neuronal networks plays increasingly key roles in both theoretical neuroscience and neuromorphic computing. The brain learns in part by modifying the synaptic strengths between neurons and neuronal populations. Classically, backpropagation (BP)[1–3] has been essential for supervised learning in artificial neural networks (ANNs). Although the question of whether or not BP operates in the brain is still an outstanding issue[4], BP does solve the problem of how a global objective function can be related to a local synaptic modification in a network. It seems clear, however, that if BP is implemented in the brain, or if one wishes to implement BP in a neuromorphic circuit, some amount of dynamical information coordination is necessary to propagate the correct information to the proper location such that appropriate local synaptic modification may take place to enable learning[5].

There is growing interest in reformulating classical algorithms for learning, optimization, and control using event-based

[1]Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich 8057, Switzerland. [2]Forschungszentrum Jülich, Jülich 52428, Germany. [3]Physics of Condensed Matter & Complex Systems (T-4), Los Alamos National Laboratory, Los Alamos, NM 87545, USA. [4]London Institute for Mathematical Sciences, Royal Institution, London W1S 4BS, UK. [5]Applied Mathematics & Plasma Physics (T-5), Los Alamos National Laboratory, Los Alamos, NM 87545, USA. [6]Center for Bioinformatics, National Laboratory of Protein Engineering and Plant Genetic Engineering, School of Life Sciences, Peking University, Beijing 100871, China. [7]Center for Quantitative Biology, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, China. [8]Information Sciences (CCS-3), Los Alamos National Laboratory, Los Alamos, NM 87545, USA. ✉e-mail: sornborg@lanl.gov

information-processing mechanisms. Such spiking neural networks (SNNs) are inspired by the function of biological neural systems[6], i.e., neuromorphic computing[7]. The trend is driven by the advent of flexible computing architectures such as Intel's neuromorphic research processor, codenamed Loihi, that enable experimentation with such algorithms in hardware[8].

There is particular interest in deep learning, a central tool in modern machine learning. Deep learning relies on a layered, feed-forward network similar to the early layers of the visual cortex, with threshold nonlinearities at each layer that resemble mean-field approximations of neuronal integrate-and-fire models. While feed-forward architectures are readily translated to neuromorphic hardware[9–11], the far more computationally intensive training of these networks 'on-chip' has proven elusive as the structure of back-propagation makes the algorithm notoriously difficult to implement in a neural circuit[12,13]. Interest in a feasible neural implementation of backpropagation has gained renewed momentum with the advent of neuromorphic computational architectures that feature local synaptic plasticity[8,14–16]. Because of the well-known difficulties, neuromorphic systems have relied to date almost entirely on conventional off-chip learning and used on-chip computing only for inference[9–11]. Developing learning systems whose function is realized exclusively using neuromorphic mechanisms has been a long-standing challenge.

Backpropagation has been claimed to be biologically implausible or difficult to implement on spiking chips because of several issues:

(a) Weight transport—usually, synapses in biology and on neuromorphic hardware cannot be used bidirectionally; therefore, separate synapses for the forward and backward pass are employed. However, correct credit assignment, i.e., knowing how a weight change affects the error, requires feedback weights to be the same as feedforward weights[17,18];
(b) Backwards computation—forward and backward passes implement different computations[18]. The forward pass requires only weighted summation of the inputs, while the backward pass operates in the opposite direction and additionally takes into account the derivative of the activation function;
(c) Gradient storage—error gradients must be computed and stored separately from activations;
(d) Differentiability—for spiking networks, the issue of non-differentiability of spikes has been discussed, and solutions have been proposed[19–24]; and
(e) Hardware constraints—for the case of neuromorphic hardware, there are often constraints on plasticity mechanisms, which allow for adaptation of synaptic weights. On some hardware, no plasticity is offered at all, while in some cases, only specific spike-timing-dependent plasticity (STDP) rules are allowed. Additionally, in almost all available neuromorphic architectures, information must be local, i.e., information is only shared between synaptically connected neurons, particularly to facilitate parallelization.

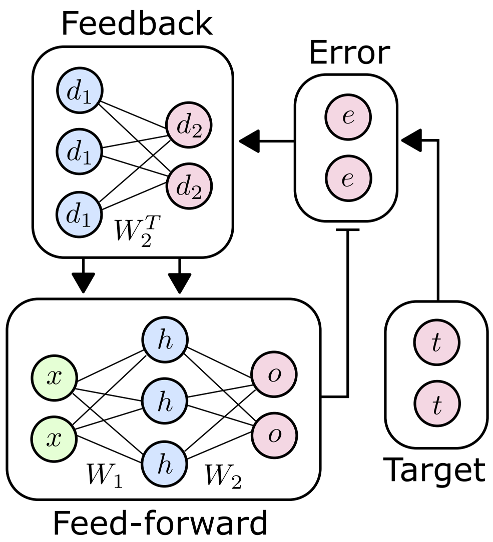

Fig. 1 | Overview of conceptual circuit architecture. Feedforward activations of input (**x**), hidden (**h**) and output (**o**) layers are calculated by a feedforward module. Errors (**e** = **t** − **o**) are calculated from the output and the training signal (**t**). Errors are backpropagated through a feedback module with the same weights W_2 for synapses between **h** and **o**, but in the opposite direction (mathematically expressed as the transpose, W_2^T). Local gradients ($\mathbf{d}_1$, $\mathbf{d}_2$) are gated back into the feedforward circuit at appropriate times to accomplish potentiation or depression of appropriate weights.

The most commonly used approach to avoiding the above issues is to use neuromorphic hardware only for inference using fixed weights obtained by training of an identical network offline and off-chip[9,10,25–29]. This approach has recently achieved state-of-the-art performance[11]. Hardware-independent ANN-SNN conversion approaches can compete with quantized neural networks also for large architectures and datasets[30,31]. However, off-chip, offline training does not make use of neuromorphic hardware's full potential and therefore consumes excessive power. Moreover, to function in most field applications, an inference algorithm should be able to learn adaptively after deployment, e.g., to adjust to a particular speaker in speech recognition, which would enable better autonomy and privacy of edge computing devices. So far, only last layer training, without back-propagation and using variants of the delta rule, has been achieved on spiking hardware[32–38]. Other on-chip learning approaches use alternatives to backpropagation[39,40], bio-inspired non-gradient based methods[41], or hybrid systems with a conventional computer in the loop[42–44]. Several promising alternative approaches for actual on-chip spiking backpropagation have been proposed recently[45–48], but have not yet been implemented in hardware.

To avoid the backward computation issue (b) and because neuromorphic synapses are not bidirectional, a separate feedback network for the backpropagation of errors has been proposed[49–51] (see Fig. 1). This leads to the weight transport problem (a), which has been solved by using symmetric learning rules to maintain weight symmetry[50,52,53] or with the Kolen-Pollack algorithm[53–55], which leads to symmetric weights automatically. It has also been found that weights do not have to be perfectly symmetric because backpropagation can still be effective with random feedback weights (random feedback alignment)[56]. However, symmetry in the sign between forward and backward weights matters[54,57].

The backward computation issue (b) and the gradient storage issue (c) have been addressed by approaches that separate the function of the neuron into different compartments and use structures that resemble neuronal dendrites for the computation of backward propagating errors[4,46,48,58,59]. The differentiability issue (d) has been circumvented by spiking rate-based approaches[10,26,60,61] that use the ReLU activation function as done in ANNs. The differentiability issue has also been addressed more generally using surrogate gradient methods[9,19,20,22,23,27,32,62] and methods that use biologically-inspired STDP and reward modulated STDP mechanisms[63–66]. For a review of SNN-based deep learning, see[67]. For a review of backpropagation in the brain, see[4].

In this study, we describe a hardware implementation of the exact backpropagation algorithm that addresses the issues described above using a set of mechanisms that have been developed and tested in simulation by the authors during the past decade, synthesized in our recent study[68] and simplified and adapted here to the features and constraints of the Loihi chip. These neuronal and network features include propagation of graded information in a circuit composed of neural populations using synfire-gated synfire chains (SGSCs)[69–72], control flow based on the interaction of synfire chains[70], and regulation of Hebbian learning using synfire-gating[73,74]. We simplify our previously proposed network architecture[68] and streamline its function. We demonstrate our approach using a proof-of-principle implementation on Loihi[8] and examine the performance of the algorithm for learning and inference of the MNIST[75] and Fashion MNIST[76] test data sets. The nBP implementation is competitive in clock time, sparsity, and power consumption with comparable state-of-the-art algorithms for the same tasks.

Perhaps most importantly, this forms a missing baseline in the field against which novel neuromorphic training algorithms can be assessed. With the proliferation of machine learning applications in the public sphere, there is a need for more efficient training options. We collect many of the current attempts into tables to facilitate further comparisons and hope this will stimulate other work investigating what can currently be achieved with available neuromorphic hardware.

Results

The binarized nBP model

For the proof of principle implementation on the Loihi hardware, we simplify and extend our previous architecture[68] using several new algorithmic and hardware-specific mechanisms. Each unit of the neural network is implemented as a single spiking neuron, using the current-based leaky integrate-and-fire (CUBA) model (see Equations (19) and (20) in the Methods section) that is built into Loihi. The time constants of the CUBA model are set to 1 so that the neurons are memoryless. Rather than using rate coding, where spikes are counted over time, we consider neuron spikes at every algorithmic time step so we can regard our implementation as a binary neural network (binary in activation, not weights). The network's feedforward component is a classic multilayer perceptron (MLP) with three layers, a binary activation function, and discrete (8-bit) weights. However, our approach may be extended to deeper networks and different encodings. In the following equations, each lowercase letter corresponds to a Boolean vector representing a layer of spiking neurons on the chip (a spike corresponds to a 1). The inference (forward) pass through the network is computed as:

$$\mathbf{o} = f(W_2 f(W_1 \mathbf{x})), \tag{1}$$

$$f(\mathbf{x}) = H(\mathbf{x} - 0.5), \tag{2}$$

$$H(\mathbf{x}) = \begin{cases} 0, & \mathbf{x}<0, \\ 1, & \mathbf{x} \geq 0, \end{cases} \tag{3}$$

where W_i is the weight matrix of the respective layer, f is a binary activation function with a threshold of 0.5, and H denotes the Heaviside function. The forward pass thus occurs in 3 time steps as spikes are propagated through layers. The degree to which the feedforward network's output ($\mathbf{o}$) deviates from a target value ($\mathbf{t}$) is quantified by the squared error, $\mathbf{E} = \frac{1}{2} \parallel \mathbf{o} - \mathbf{t} \parallel^2$, which we would like to minimize. Performing backpropagation to achieve this requires the calculation of weight updates, which depend on the forward activations, and backward propagated local gradients $\mathbf{d}_l$, which represent the amount by which the loss changes when the activity of that neuron changes, as:

$$\mathbf{d}_2 = (\mathbf{o} - \mathbf{t}) \circ f'(W_2 \mathbf{h}), \tag{4}$$

$$\mathbf{d}_1 = \mathrm{sgn}(W_2^T \mathbf{d}_2) \circ f'(W_1 \mathbf{x}), \tag{5}$$

$$\frac{\partial \mathbf{E}}{\partial W_l} = \mathbf{d}_l (\mathbf{a}_{l-1})^T, \tag{6}$$

$$W_l^{\mathrm{new}} = W_l^{\mathrm{old}} - \eta \frac{\partial \mathbf{E}}{\partial W_l}, l = 1, 2. \tag{7}$$

Here, $\circ$ denotes a Hadamard product, i.e. the element-wise product of two vectors, ^{T} denotes the matrix transpose, sgn($\mathbf{x}$) is the sign function, and $\mathbf{a}_l$ denotes the activation of the lth layer, $f(W_l \mathbf{a}_{l-1})$, with $\mathbf{a}_0 = \mathbf{x}$, $\mathbf{a}_1 = \mathbf{h}$, $\mathbf{a}_2 = \mathbf{o}$. Here, η denotes the learning rate and is the only hyperparameter of the model apart from the weight initialization'. Denotes the derivative, but because f is a binary thresholding function (Heaviside), the derivative would be the Dirac delta function, which is zero everywhere apart from at the threshold. Therefore, we use a common method[77,78] and represent the thresholding function using a truncated (between 0 and 1) ReLU (Equation (8)) as a surrogate or straight-through estimator when back-propagating the error. The derivative of the surrogate is a box function (Equation (9)):

$$f_{\mathrm{surrogate}}(\mathbf{x}) = \min(\max(\mathbf{x}, 0), 1), \tag{8}$$

$$f'(\mathbf{x}) = H(\mathbf{x}) - H(\mathbf{x} - 1). \tag{9}$$

The three functions (Equations (2), (8) and (9)) are plotted in the inset in Fig. 2.

When performed for each target ($\mathbf{t}$) in the training set, the model may be considered a stochastic gradient descent algorithm with a fixed step size update for each weight in the direction of the gradient sign.

nBP on Neuromorphic Hardware

On the computational level, Equations (1)–(9) fully describe the model exactly as implemented on Loihi, except for the handling of bit precision constraints that affect integer discreteness and value limits and ranges. In the following, we describe how these equations are translated from the computational to the algorithmic neural circuit level, enabling implementation on neuromorphic hardware. Further details on the implementation can be found in the Methods section.

a. **Hebbian weight update** Equation (7) effectively results in the following weight update per single synapse from presynaptic index i in layer $l-1$ to postsynaptic index j in layer l:

$$\Delta w_{ij} = -\eta \cdot a_{l-1,i} \cdot d_{l,j}, \tag{10}$$

where η is the constant learning rate. To accomplish this update, we use a Hebbian learning rule[79] implementable on the on-chip microcode learning engine (for the exact implementation on Loihi, see Methods). Hebbian learning means that neurons that fire together, wire together, i.e., the weight update Δw is proportional to the product of the simultaneous activity of the presynaptic (source) and the postsynaptic (target) neurons. In our case, this means that the values of the two factors of Equation (10) have to be propagated simultaneously, in the same time step, to the pre- ($a_{l-1,i}$) and postsynaptic ($d_{l,j}$) neurons. In contrast, the pre- and postsynaptic neurons are not allowed to fire simultaneously at any other time. For this purpose, a mechanism to control the information flow through the network is needed.

b. **Gating controls the information flow** As an information control mechanism, we use synfire gating[69–71,80]. The gating chain, a closed chain of 12 neurons containing a single spike perpetually sent around the circle, is the backbone of this flow control mechanism. The gating chain controls information flow through the controlled network by selectively boosting layers to bring their neurons closer to the threshold, thereby making them receptive to input.

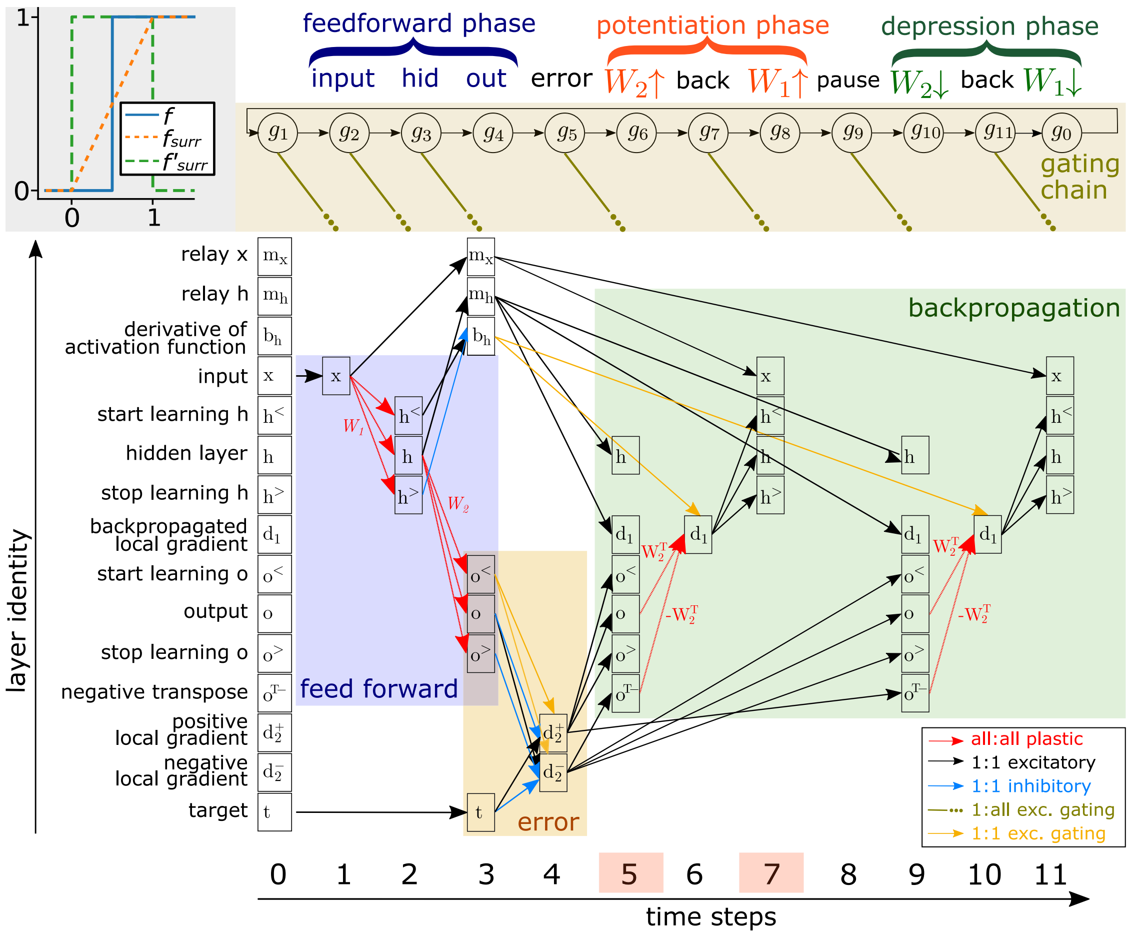

Fig. 2 | Functional connectivity of the 2-layer backpropagation circuit. Layers are only shown when gated `on,' and synapses are only shown when their target is gated on. Plastic connections are all-to-all (fully connected), i.e., all neurons are connected to all neurons in the next layer. The gating connections from the gating chain are one-to-all, and all other connections are one-to-one, which means that a firing pattern is copied directly to the following layer. The names of the neuron layers are given on the left margin so that the row corresponds to layer identity. The columns correspond to the time steps of the algorithm, which are the same as the time steps on Loihi. Time steps 5 and 7 are highlighted as in these steps, the sign of the weight update is inverted (positive), as $r = 1$ in Equation (23). Supplementary Table I shows the information contained in each layer in each respective time step and a detailed step-by-step explanation of the algorithm is given in the Supplementary Methods. The plot in the top left corner illustrates our approach to approximate the activation function f by a surrogate with the box function as derivative, $f_{\mathrm{surr}} = H(\mathbf{x})H(1-\mathbf{x})$, where f' is the rectified linear map (ReLU) (see Equations (2), (8) and (9)).

By connecting particular layers to the gating neuron that fires in the respective time steps, we lay out a path that the activity through the network is allowed to take. For example, to create the feedforward pass, the input layer **x** is connected to the first gating neuron and therefore gated 'on' in time step 1, the hidden layer **h** is connected to the second gating neuron and gated 'on' in time step 2, and the output layer **o** is connected to the third gating neuron and gated 'on' in time step 3. A schematic of this path of the activity can be found in Fig. 2. To speak in neuroscience terms, we are using synfire gating to design functional connectivity through the network anatomy shown in Supplementary Fig. 1. Using synfire gating, the local gradient $d_{l,j}$ is brought to the postsynaptic neuron at the same time as the activity $a_{l-1,i}$ is brought back to the presynaptic neuron effecting a weight update. In addition to bringing activity at the right time to the right place for Hebbian learning, the gating chain also makes it possible to calculate and back-propagate the local gradient.

c. **Local gradient calculation** For the local gradient calculation, according to Equation (5), the error $\mathbf{o} - \mathbf{t}$ and the box function derivative of the surrogate activation function (Equation (9)) are needed. Because there are no negative (signed) spikes, the local gradient is calculated and propagated back twice for a Hebbian weight update in two phases with different signs. The error $\mathbf{o} - \mathbf{t}$ is calculated in time step 4 in a layer that receives excitatory (positive) input from the output layer **o** and inhibitory (negative) input from the target layer **t**, and vice versa for $\mathbf{t} - \mathbf{o}$.

The box function $f'(\mathbf{x})$ (Equation (9)) has the role of initiating learning when the presynaptic neuron receives a non-negative input and of terminating learning when the input exceeds 1, which is why we call the two conditions 'start' and 'stop' learning (inspired by the nomenclature of ref. 81). This inherent feature of backpropagation avoids weight updates that do not affect the current output as the neuron is saturated with the current input due to the nonlinearity. This regulates learning by protecting trained weights mainly used for other inputs.

To implement these two terms of the box function (Equation (9)), we use two copies of the output layer that receive the same input ($W_2\mathbf{h}$) as the output layer. Using the above-described gating mechanism, one of the copies (start learning, $\mathbf{o}^<$) is brought exactly to its firing threshold when it receives the input, which means that it fires for any activity >0, and the input is not in the lower saturation region of the ReLU. The other copy (stop learning, $\mathbf{o}^>$) is brought further away from the threshold (to 0),

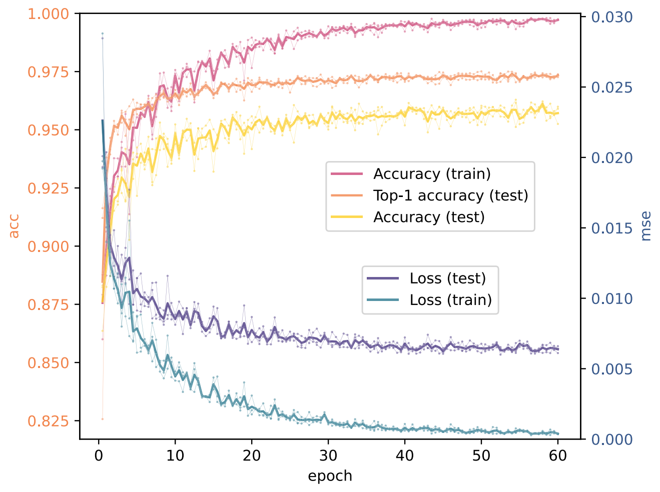

Fig. 3 | Accuracy (acc) and loss (mean squared error, mse) over epochs. The plot shows the mean over three training runs with random weight initialization and random dataset shuffles during training. Individual runs are shown with dots connected by a finer line. In addition to the accuracy and loss calculated using the last layer's spikes, the top-1 accuracy is calculated from the last layer's membrane potential. The accuracy calculation was performed off-chip using checkpoints stored from the on-chip trained network after each half training epoch. Note separate axis scaling for accuracy (left) and loss (right).

which means that if it fires, the upper saturation region of the ReLU has been reached, and learning should cease.

d. **Error backpropagation** Once the local gradient $\mathbf{d}_2$ is calculated as described in the previous paragraph, it is sent to the output layer to update W_2 in time steps 5 and 9. From there, the local gradient is propagated back through the weight matrices W_2^T and $-W_2^T$. These weight matrices are 'transposed' copies of W_2, i.e., they are connected in the opposite direction. Once propagated backward, the back-propagated error is combined with the 'start' and 'stop' learning conditions and then sent to the hidden layer in steps 7 and 11. All copies of the weight matrices receive the same update so that they maintain the same weight values.

Algorithm performance

Our implementation of the nBP algorithm on Loihi achieves an average inference accuracy of 95.7% after 60 epochs (best of 3 runs in any epoch: 96.3%) on the MNIST test data set, which is comparable with other shallow, stochastic gradient descent (SGD) trained MLP models without additional allowances. When multiple spikes were present in the output layer, the lowest indexed output was accepted. This occurred in ~2% of cases. The average top-1 accuracy read out from the last layer (before thresholding) is 97.3% (best of 3 runs: 97.5%). A more detailed breakdown of the accuracy is included in Supplementary Table III. The accuracy and loss over epochs are shown in Fig. 3.

In these experiments, the nBP model is distributed over 81 neuromorphic cores. Processing of a single sample takes 1.5 ms (0.17 ms for inference only) on the neuromorphic cores, including the time required to send the input spikes from the embedded processor. It consumes 0.6 mJ of energy on the neuromorphic cores (0.59 mJ of which is dynamic energy, i.e. energy used by the neural circuit in addition to the fixed background energy), resulting in an energy-delay product of 0.88 μJs. Table 1 and Supplementary Table IV compare our results with published performance metrics for other neuromorphic learning architectures also tested on MNIST. Supplementary Table II shows a breakdown of energy consumption and a comparison of different conditions and against a GPU. Switching off the learning engine after training reduces the dynamic energy per inference to 0.02 mJ, which reveals that the on-chip learning engine is responsible for most power consumption. Because the learning circuit is unnecessary for inference, we also tested a reduced architecture that performs inference within four time steps using the previously trained weights. This architecture uses 0.0025 mJ of dynamic energy and 0.17 ms per inference.

The nBP algorithm trains the network without explicit sparsity constraints, yet it exhibits sparsity because of its binary (spiking) nature. After applying the binary threshold of 0.5 to the MNIST images, one image is encoded using, on average, 100 spikes in the input layer, corresponding to a sparsity of 0.25 spikes per neuron per inference. This leads to a typical number of 110 spikes in the hidden layer (0.28 spikes per neuron per inference) and one spike in the output layer (0.1 spikes per neuron per inference). Furthermore, as expected, the sparsity of the backpropagation layers increases during training: The error-induced activity from the local gradient layer $\mathbf{d}_1$ starts with 0.7 spikes per neuron per sample (during the first 1000 samples) and is essentially switched off in the trained network as the error approaches 0.

Additionally we evaluated the algorithm on the more challenging drop-in replacement Fashion MNIST[76] and obtained an accuracy of 79% (81.8% top-1 before thresholding) with a network of size 784-400-10 with binarized images after 40 epochs. This is close to the accuracy (84.4%) of an ANN of this size trained with stochastic gradient descent on the same dataset (for details, see Methods).

Discussion

As we have demonstrated here, by using a well-defined set of neuronal and neural circuit mechanisms, it is possible to implement the backpropagation algorithm on contemporary neuromorphic hardware. Previously proposed methods to address the issues outlined in the Introduction were not on their own able to offer a straightforward path to implement a variant of the nBP algorithm on current hardware. In this study, we avoided or solved these previously encountered issues with neuromorphic backpropagation by combining known solutions with synfire-gated synfire chains (SGSC) as a dynamical information coordination scheme. The algorithm was evaluated on the MNIST and Fashion MNIST test data set on the Loihi VLSI hardware.

The five issues (a)–(e) listed in the Introduction were addressed using the following solutions: (a) The weight transport issue was avoided via the use of a deterministic, symmetric learning rule for the parts of the network that implement inference (feed-forward) and error propagation (feedback) as described by[50]. This approach is not biologically plausible because of a lack of developmental mechanisms to assure the equality of corresponding weights[53]. It would, however, without modifications to the architecture be feasible to employ weight decay as described by Kolen and Pollack[53] to achieve self-alignment of the backward weights to the forward weights or to use feedback alignment to approximately align the feedforward weights to random feedback weights[56]; (b) The backward computation issue was solved by using a separate error propagation network through which activation is routed using an SGSC; (c) The gradient storage issue was solved by routing activity through the inference and error propagation circuits within the network in separate stages, thereby preventing the mixing of inference and error information. Some alternatives would not require synfire gated routing but are more challenging to implement on hardware[46,48] as also described in a more comprehensive review[4]; (d) The differentiability issue was solved by representing the step activation function by a surrogate in the form of a (truncated) ReLU activation function with an easily implementable box function derivative; and (e) The hardware constraint issue was solved by the proposed mechanism's straightforward implementation on Loihi because it only requires standard integrate-and-fire neurons and Hebbian learning that is modulated by a single factor which is the same for all synapses.

Table 1 | Review of the MNIST Literature on neuromorphic hardware

Publication	Hardware	Learning Mode	Network Structure	Energy per Sample (mJ)	Latency per Sample (ms)	Test Accuracy (%)
On-chip backpropagation						
Renner et al. (2024) [This study]	Loihi	on-chip nBP	400-400-10[a]	0.59	1.5	96.3 (97.5[b])
On-chip single layer training or BP alternatives						
40 Shrestha et al. (2021)	Loihi	EM-STDP FA/DFA	CNN-CNN-100-10	8.4	20	94.7
39 Frenkel et al. (2020)	SPOON	DRTP	CNN-10	0.000366[c]	0.12	95.3
37 Park et al. (2019)	unnamed	mod. SD	784-200-200-10	0.000253[c]	0.01	98.1
102 Chen et al. (2018)	unnamed	S-STDP	236-20[d]	0.017	0.16	89
34 Frenkel et al. (2018)	ODIN	SDSP	256-10	0.000015	-	84.5
101 Lin et al. (2018)	Loihi	S-STDP	1920-10[d]	0.553	-	96.4
36 Buhler et al. (2017)	unnamed	LCA features	256-10	0.000050	0.001[c]	88
On-chip inference only						
Renner et al. (2024) [This study]	Loihi	inference	400-400-10[a]	0.0025	0.17	96.3 (97.5[b])
40 Shrestha et al. (2021)	Loihi	inference	CNN-CNN-100-10	2.47	10	94.7
39 Frenkel et al. (2020)	SPOON	inference	CNN-10	0.000313	0.12	97.5
85 Göltz et al. (2019)	BrainScaleS-2	inference	256-246-10	0.0084	0.048	96.9
101 Lin et al. (2018)	Loihi	inference	1920-10[d]	0.0128[e]	-	96.4
102 Chen et al. (2018)	unnamed	inference	784-1024-512-10	0.0017	-	97.9
106 Esser et al. (2015)	True North	inference	CNN (512 neurons)	0.00027	1	92.7
106 Esser et al. (2015)	True North	inference	CNN (3840 neurons)	0.108	1	99.4
107 Stromatias et al. (2015)	SpiNNaker	inference	784-500-500-10	3.3	11	95

The table includes three relevant classes of literature. Note that the energy-delay product may be computed from the Energy per Sample and Latency per Sample columns. For a table of simulated SNN learning algorithms, see Supplementary Table IV. Abbreviations: EM-STDP Error-modulated spike-timing dependent plasticity, DFA Direct feedback alignment, DRTP Direct random target projection, SD Segregated dendrites, SDSP Spike-driven synaptic plasticity, LCA Locally competitive algorithm.

[a]400 (20 × 20) corresponds to 784 (28 × 28) after cropping the empty (0-padded) image margin of 4 pixels. Including these pixels does not affect accuracy and has a minor effect on inference energy but roughly doubles energy for training, see Supplementary Table II.

[b]Value in parentheses is top-1 accuracy read out from last-layer activity (membrane potential) before thresholding.

[c]Calculated from given values.

[d]Off-chip preprocessing.

[e]Dynamic energy reported in the Supplementary Material of [82].

While neural network algorithms on GPUs usually use operations on dense activation vectors and weight matrices and, therefore, do not profit from sparsity, spiking neuromorphic hardware only performs an addition operation when a spike event occurs, i.e., adding the weight to the input current, as in Equations (19) and (20). This means that the power consumption directly depends on the number of spikes. Therefore, sparsity, which refers to the property of a vector to have mostly zero elements, is essential for neuromorphic algorithms[82,83], and it is also observed in biology[84]. Consequently, significant effort has been made to make SNNs sparse to overcome the classical rate-based approach based on counting spikes[83,85–87]. The binary encoding used here could be seen as a limit case of the quantized rate-based approach, allowing only 0 or 1 spike. Even without regularization to promote sparse activity, it yields very sparse activation vectors that are as sparse as most timing codes but only use a single time step. However, the achievable encoded information per spike is unquestionably lower. In a sense, we already employ spike timing to route spikes through the network because the location of a spike in time within the 12 time steps determines if and where it is sent and if the weights are potentiated or depressed. Using a timing code for activations could be enabled by having more than one Loihi time step per algorithm time step. Therefore, the use of SGSCs is not limited to this particular binary encoding, and in fact, SGSCs were initially designed for a population rate code.

Similarly, the routing method used in this work is not limited to backpropagation. It could serve as a general method to route information in SNNs where autonomous activity (without interference from outside the chip) is needed. That is, our proposed architecture can act in a similar way as or even in combination with neural state machines[88,89].

Although our implementation of nBP here was focused primarily on a particular hardware environment, we point out that the synfire-gated synfire chains and other network and neuronal structures that we employ could all potentially have relevance to the understanding of computation in neurophysiological systems. Many of the concepts we use, such as spike coincidence, were originally inspired by neurophysiological experiments[69,70,90]. Experimental studies have shown recurring sequences of stereotypical neuronal activation in several species and brain regions[91–93] and particularly replay in hippocampus[94]. Recent studies also hypothesize[95,96] and show[97] that a mechanism like gating by synfire chains may play a role in memory formation. Additional evidence[98] shows that large-scale cortical activity has a stereotypical, packet-like character that can convey information about the nature of a stimulus or be ongoing or spontaneous. This apparently algorithmically-related activity has a very similar form to the SGSC-controlled information flow found previously[69–72]. Interestingly, this type of sequential activation of populations is evoked by the nBP learning architecture, as seen in the raster plot in Supplementary Fig. 2.

Other algorithmic spiking features, such as the back-propagated local gradient layer activity decreasing from 0.7 spikes per neuron to 0 by the end of training, could be identified and used to generate qualitative and quantitative hypotheses concerning network activity in biological neural systems.

Although our accuracy is similar to early implementations of binary neural networks in software[78], subsequent approaches now reach 98.5%[99] and generally include binarized weights. However, networks that achieve such accuracy typically employ a convolutional structure or multiple larger hidden layers. Additional features such as dropout, softmax final layers, gain terms, and others could, in principle, be included in spiking hardware and may also account for this 3% gap. So, while we show that it is possible to implement backpropagation on neuromorphic hardware efficiently, several non-trivial steps are still required to make it usable in practical applications:

1. The algorithm needs to be scaled to deeper networks. While the present structure is, in principle, scalable to more layers without considerable adjustments, more investigation is needed to determine whether gradient information remains intact over many layers and to what extent additional features, such as alternatives to batch normalization, may need to be developed.
2. Generalization to convolutional networks is compelling, particularly for image processing applications. The Loihi hardware is advantageous in this setting because of its weight-sharing mechanisms.
3. Although our current implementation demonstrates on-chip learning, we train on the images in an offline fashion by iterating over the training set in epochs. Further research is required to develop truly continual learning mechanisms so that additional samples and classes can be learned without losing previously trained synaptic weights and without retraining on the whole dataset.Additionally, the proposed algorithmic methodology can inform hardware adjustments to promote efficiency for learning applications. Although it is highly efficient in terms of power usage, in particular for binary encoding, the Loihi hardware is not specifically designed for implementing standard deep learning models, but rather as general-purpose hardware for exploring different SNN applications[82].

This leads to a considerable computational overhead for functions not needed in the proposed model (e.g., neuronal dynamics) or that could have been realized more efficiently if integrated directly on the chip instead of using network mechanisms. The proposed model provides a potential framework to guide future hardware modifications to facilitate more efficient learning algorithm implementations. For example, in an upgraded version of the algorithm, relay neurons could be replaced by presynaptic (eligibility) traces to keep a memory of the presynaptic activity for the weight update. Furthermore, on the new version of the Loihi hardware, Loihi 2, graded spikes were introduced[100], which could allow an efficient implementation of the backpropagation algorithm also for non-binary neural activations.

To our knowledge, this work is the first to show an SNN implementation of the backpropagation algorithm that is fully on-chip without a computer in the loop. Other on-chip learning approaches so far either use feedback alignment[40], forward propagation of errors[39] or single layer training[32,34,36,101,102]. Compared to an equivalent implementation on a GPU, there is no loss in accuracy but about two orders of magnitude power savings in the case of small batch sizes, which are more realistic for edge computing settings. So, this implementation shows a path for using in-memory, massively parallel neuromorphic processors for low-power, low-latency implementation of modern deep learning applications. It can serve as a foundation and baseline for approaches that approximate backpropagation or implement it using refined, brain-inspired learning mechanisms, or bespoke circuitry. Due to its simplicity and explicit representation of all algorithmic elements, novel approaches should at least surpass the given performance and power metrics. The proposed network model offers opportunities as a building block that can, e.g., be integrated into larger SNN architectures that could profit from a trainable on-chip processing stage.

Methods

In this section, we describe our system on three different levels[103]. First, we describe the computational level by fully stating the equations that result in the intended computation. Second, we describe the spiking neural network (SNN) algorithm. Third, we describe the details of our hardware implementation necessary for exact reproducibility.

The binarized backpropagation model

Network Model. The inference (forward) pass through the network is computed as

$$\mathbf{o} = f(W_N f(W_{N-1}(\ldots f(W_1\mathbf{x})))), \tag{11}$$

where f is an element-wise nonlinearity and W_i is the weight matrix of the respective layer. The degree to which the network's output ($\mathbf{o}$) deviates from the target values ($\mathbf{t}$) is quantified by the squared error, $\mathbf{E} = \frac{1}{2}\parallel \mathbf{o} - \mathbf{t}\parallel^2$, which we aim to minimize. The weight updates for each layer are computed recursively by

$$\mathbf{d}_l = \begin{cases} (\mathbf{o} - \mathbf{t}) \circ f'(\mathbf{n}_l), & l = N \\ W_{l+1}^T \mathbf{d}_{l+1} \circ f'(\mathbf{n}_l), & l < N \end{cases}, \tag{12}$$

$$\frac{\partial \mathbf{E}}{\partial W_{l+1}} = \mathbf{d}_{l+1}(\mathbf{d}_l)^T, \tag{13}$$

$$W_l^{\text{new}} = W_l^{\text{old}} - \eta \frac{\partial \mathbf{E}}{\partial W_l}, \tag{14}$$

where $\mathbf{n}_l$ is the network activity at layer l (i.e., $\mathbf{n}_l = W_l f(W_{l-1}\mathbf{n}_{l-1})$). Here, $'$ denotes the derivative, $\circ$ denotes a Hadamard product, i.e. the element-wise product of two vectors, ^{T} denotes the matrix transpose, and $\mathbf{d}_l$ denotes $f(W_l\mathbf{d}_{l-1})$, with $\mathbf{d}_0 = \mathbf{x}$. The parameter η denotes the learning rate. These general Equations (12)–(14) are realized for two layers in our implementation as given by Equations (5)–(7) in the Results section. Below, we relate these equations to the neural Hebbian learning mechanism used in the neuromorphic implementation of nBP.

As the thresholding (Heaviside) activation function is not differentiable, we replace it with a surrogate[77,78] rectified linear map (ReLU) truncated between 0 and 1 (Equation (16)) in the part of the circuit that affects error backpropagation. The derivative f' of the surrogate in Equation (12) is then a box function, i.e.

$$f(\mathbf{x}) = H(\mathbf{x} - 0.5), \tag{15}$$

$$f_{\text{surrogate}}(\mathbf{x}) = \min(\max(\mathbf{x}, 0), 1), \tag{16}$$

$$f'(\mathbf{x}) = H(\mathbf{x}) - H(\mathbf{x} - 1), \tag{17}$$

where H denotes the Heaviside function:

$$H(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} < 0, \\ 1, & \mathbf{x} \geq 0. \end{cases} \tag{18}$$

The following section describes how we implement Equations (11)–(16) in a spiking neural network.

Weight initialization. Plastic weights are initialized by sampling from a Gaussian distribution with a mean of 0 and a standard deviation of $1/\sqrt{2/(N_{\text{fanin}} + N_{\text{fanout}})}$ (He initialization[104]). N_{fanin} denotes the number of neurons of the presynaptic layer and N_{fanout} the number of neurons of the postsynaptic layer.

Input data. The images of the MNIST dataset[75] were optionally cropped by a margin of 4 pixels on each side to remove pixels that are never active and avoid unused neurons and synapses on the chip.

The pixel values were thresholded with 0.5 to get a black-and-white image as input to the network. In the case of the 100 − 300 − 10 architecture, the input images were downsampled by a factor of 2. The dataset was presented in a different random order in each epoch.

Accuracy calculation. The reported accuracies are calculated on the MNIST or FMNIST test data set. A sample was counted as correct when the index of the spiking neuron of the output layer in the output phase (time step 3 in Fig. 2) is equal to the correct label index of the presented image. In <1% of cases, there was more than one spike in the output layer; in that case, the lowest spiking neuron index was compared.

The neuromorphic backpropagation algorithm

Spiking neuron model. For a generic spiking neuron element, we use the current-based linear leaky integrate-and-fire model (CUBA). This model is implemented on Intel's neuromorphic research processor, codenamed Loihi[8]. Time evolution in the CUBA model as implemented on Loihi is described by the discrete-time dynamics with $t \in \mathbb{N}$, and time increment $\Delta t \equiv 1$:

$$V_i(t+1) = V_i(t) - \frac{1}{\tau_V} V_i(t) + U_i(t) + I_{\text{const}}, \tag{19}$$

$$U_i(t+1) = U_i(t) - \frac{1}{\tau_U} U_i(t) + \sum_j w_{ij}\delta_j(t), \tag{20}$$

where i identifies the neuron.

The membrane potential $V(t)$ is reset to 0 upon exceeding the threshold V_{thr} and remains at its reset value 0 for a refractory period, T_{ref}. Upon reset, a spike is sent to all connecting synapses. Here, $U(t)$ represents a neuronal current, and δ represents a time-dependent spiking input. I_{const} is a constant input current.

Parameters and mapping. In our implementation of the backpropagation algorithm, we take $\tau_V = \tau_U = 1$, $T_{\text{ref}} = 0$, and $I_{\text{const}} = -8192$ (except in gating neurons, where $I_{\text{const}} = 0$). This leads to a memoryless point neuron that spikes whenever its input in the respective time step exceeds $V_{\text{thr}} = 1024$. This happens when the neuron receives synaptic input larger than $0.5 \cdot V_{\text{thr}}$, and in the same time step, a gating input overcomes the strong inhibition of the I_{const}, i.e., it is gated 'on.' This is how the Heaviside function in Equation (15) is implemented. For the other activation functions, a different gating input is applied.

There is a straightforward mapping between the weights and activations in the spiking neural network (SNN) described in this section and the corresponding artificial neural network (ANN) described in the previous subsection:

$$w_{\text{SNN}} = w_{\text{ANN}} \cdot V_{\text{thr}}, \tag{21}$$

$$a_{\text{SNN}} = a_{\text{ANN}} \cdot V_{\text{thr}}. \tag{22}$$

So, a value of $V_{\text{thr}} = 1024$ allows for a maximal ANN weight of 0.25 because the allowed weights on Loihi are the even integers from -256 – 254.

Feed-forward pass. The feedforward pass can be seen as an independent circuit module that consists of 3 layers: An input layer **x** with 400 (20 × 20) neurons that spikes according to the binarized MNIST dataset, a hidden layer **h** of 400 neurons, and an output layer **o** of 10 neurons. The three layers are sequentially gated 'on' by the gating chain so that activity travels from the input layer to the hidden layer through the plastic weight matrix W_1 and then from the hidden to the output layer through the plastic weight matrix W_2.

Learning rule. The plastic weights follow the standard Hebbian learning rule with a global third factor to control the sign. Note, however, that here, unlike other work with Hebbian learning rules, due to the particular activity routed to the layers, the learning rule implements a supervised mechanism (backpropagation). Here, we give the discrete update equation as implemented on Loihi:

$$\begin{aligned}\Delta w &= 4r(t) \circ x(t) \circ y(t) - 2x(t) \circ y(t)\\ &= (2r(t) - 1) \cdot 2x(t)y(t),\end{aligned} \tag{23}$$

$$\text{r}(t) = \begin{cases} 1, & \text{if } (t \bmod T) = 5, 7, \\ 0, & \text{otherwise}. \end{cases} \tag{24}$$

Above, x, y, and r represent the time series available at the synapse on the chip. The signals x and y are the pre- and postsynaptic neuron's spike trains, i.e., they are equal to 1 in time steps when the respective neuron fires and 0 otherwise. The signal r is a third factor provided to all synapses globally. It determines in which phase (potentiation or depression) the algorithm is in. T denotes the number of phases per trial, which is 12 in this case. So, r is 0 in all time steps apart from the 5th and 7th of each iteration, where the potentiation of the weights happens. This regularly occurring r signal could thus be generated autonomously using the gating chain. On Loihi, r is provided using a so-called "reinforcement channel". Note that the reinforcement channel can only provide a global modulatory signal that is the same for all synapses.

The above learning rule produces a positive weight update in time steps in which all three factors are 1, i.e., when both pre- and postsynaptic neurons fire and the reinforcement channel is active. It produces a negative update when only the pre- and post-synaptic neurons fire, and the weight stays the same in all other cases.

To achieve the correct weight update according to the backpropagation algorithm (see Equation (10)), the spiking network has to be designed in a way that the presynaptic activity $a_{l-1,i}$ and the local gradient $d_{l,j}$ are present in neighboring neurons at the same time step. Furthermore, the sign of the local gradient has to determine if the simultaneous activity happens in a time step with active third factor r or not.

This requires control over the flow of spikes in the network, which is achieved by a mechanism called synfire gating[73,74], which we adapt and simplify here.

Gating chain. Gating neurons are a separate structure within the backpropagation circuit and are connected in a ring. That is, the gating chain is a circular chain of neurons that, once activated, controls the timing of information processing in the rest of the network. This allows information routing throughout the network to be autonomous to realize the benefits of neuromorphic hardware without the need for control by a classical sequential processor. Specifically, the neurons in the gating chain are connected to the relevant layers, allowing them to control when and where information is propagated. All layers are inhibited far away from their firing threshold by default and can only transmit information, i.e., generate spikes, if their inhibition is neutralized via activation by the gating chain. Because a neuron only fires if it is gated 'on' AND gets sufficient input, such gating corresponds to a logical AND or coincidence detection with the input.

In our implementation, the gating chain consists of 12 neurons corresponding to the 12 algorithmic (Loihi) time steps needed to process one sample. Each neuron is connected to all layers that must be active in each respective time step. The network layers are connected in the manner shown in Supplementary Fig. 1, but the circuit structure, which is controlled by the gating chain, results in a functionally connected network as presented in Fig. 2, where the layers are shown according to the timing of when they become active during one iteration.

The weight of the standard gating synapses (from one of the gating neurons to each neuron in a target layer) is $w_{\mathrm{gate}} = -I_{\mathrm{const}} + 0.5V_{\mathrm{thr}}$, i.e. each neuron that is gated 'on' is brought to half of its firing threshold, which effectively implements Equation (15). In four cases, i.e., for the synapses to the start learning layers in time step 2 ($\mathbf{h}^{<}$) and 3 ($\mathbf{o}^{<}$) and to the backpropagated local gradient layer $\mathbf{d}_1$ in time steps 6 and 10, the gating weight is $w_{\mathrm{gate}} = -I_{\mathrm{const}} + V_{\mathrm{thr}}$. In two cases, i.e., for the synapses to the stop learning layer in time step 2 ($\mathbf{h}^{>}$) and 3 ($\mathbf{o}^{>}$), the gating weight is $w_{\mathrm{gate}} = -I_{\mathrm{const}}$. These different gating inputs lead to step activation functions with different thresholds, as required for the computations explained in point 3 in the next paragraph.

Backpropagation network modules. In the previous sections, we have explained how the weight update happens and how to bring the relevant values ($\mathbf{d}_{l-1}$ and $\mathbf{d}_l$ according to Equation (10)) to the correct place at the correct time. In this section, we discuss how these values are actually calculated. The signal $\mathbf{d}_{l-1}$, which is the layer activity from the feedforward pass, does not need to be calculated but only remembered. This is done using a relay layer with synaptic delays, as explained in point 1 below. The signal $\mathbf{d}_2$, the last layer local gradient, consists of 2 parts according to Equation (5). The difference between the output and the target $\mathbf{o} - \mathbf{t}$ (see point 2) and the box function f' must be calculated. We factor the latter into two terms, a start and stop learning signal (see point 3). The signal $\mathbf{d}_1$, the backpropagated local gradient, also consists of 2 parts, according to Equation (6). In addition to another 'start' and 'stop' learning signal, we need $\mathrm{sgn}(W_2^T\mathbf{d}_2)$, whose computation is explained in point 4.

In the following equation, the weight update is annotated with the number of the paragraph in which the calculating module is described:

$$\Delta W_2 = \eta\,\overbrace{(\mathbf{o}-\mathbf{t})}^{2.} \circ \overbrace{f'(W_2\mathbf{h})}^{3.}\,\overbrace{\mathbf{h}^T}^{1.}, \tag{25}$$

$$\Delta W_1 = \eta\,\underbrace{\mathrm{sgn}(W_2^T\mathbf{d}_2)}_{4.} \circ \underbrace{f'(W_1\mathbf{x})}_{3.}\,\underbrace{\mathbf{x}^T}_{1.}. \tag{26}$$

1. **Relay neurons**. The memory used to store the activity of the input and the hidden layer is a single relay layer connected both from and to the respective layer in a one-to-one manner with the proper delays. The input layer $\mathbf{x}$ sends its activity to the relay layer $\mathbf{m}_x$ so that the activity can be restored in the W_1 update phases in time steps 7 and 11. The hidden layer $\mathbf{x}$ sends its activity to the relay layer $\mathbf{m}_h$ so that the activity can be restored in the W_2 update phases in time steps 5 and 9.
2. **Error calculation**. The error calculation requires a representation of signed quantities, which is not directly possible in a spiking network because there are no negative spikes. This is achieved here by splitting the error evaluation into two parts, $\mathbf{t} - \mathbf{o}$ and $\mathbf{o} - \mathbf{t}$, to yield the positive and negative components separately. Similarly, the calculation of back-propagated local gradients, $\mathbf{d}_1$, is performed using a negative copy of the transpose weight matrix, and it is done in 2 phases for the positive and negative local gradient, respectively. In the spiking neural network, $\mathbf{t} - \mathbf{o}$ is implemented by an excitatory synapse from $\mathbf{t}$ and an inhibitory synapse of the same strength from $\mathbf{o}$, and vice versa for $\mathbf{o} - \mathbf{t}$. Like almost all other nonplastic synapses in the network, the absolute weight of the synapses is just above the value that makes the target neuron fire when gated on. However, the difference between the output and the target is just one part of the local gradient $\mathbf{d}_2$. The other part is the derivative of the activation function (box function).
3. **Start and stop learning conditions**. The box function (Equation (17)) can be split into two conditions: a 'start' learning and a 'stop' learning condition. These two conditions are calculated in parallel with the feedforward pass. The feedforward activation $f(\mathbf{x}) = H(\mathbf{x} - 0.5V_{\mathrm{thr}})$ corresponding to Equation (1) is an application of the spiking threshold to the layer's input with an offset of $0.5V_{\mathrm{thr}}$, which is given by the additional input from the gating neurons. The first term of the box function (Equation (9)), $H(\mathbf{x})$, is also an application of the spiking threshold, but this time with an offset equal to the firing threshold so that any input larger than 0 elicits a spike. We call this first term the 'start' learning condition, and it is represented in $h^{<}$ for the hidden and in $\mathbf{o}^{<}$ for the output layer. The second term of the box function in Equation (9), $-H(\mathbf{x} - 1V_{\mathrm{thr}})$, is also an application of the spiking threshold, but this time without an offset so that only an input larger than the firing threshold elicits a spike. We call this second term the 'stop' learning condition, and it is represented in $\mathbf{h}^{>}$ and $\mathbf{o}^{>}$ for the hidden and output layers, respectively. For the W_1 update, the two conditions are combined in a box function layer $\mathbf{b_h} = \mathbf{h}^{<} - \mathbf{h}^{>}$ that then gates the $\mathbf{d}_1$ local gradient layer. For the W_2 update, the two conditions are directly applied to the $\mathbf{d}_2$ layers because an intermediate $\mathbf{b_o}$ layer would waste one time step. The function is, however, the same: the stop learning $\mathbf{o}^{>}$ inhibits the $\mathbf{d}_2$ layers, and the 'start' learning signal $\mathbf{o}^{<}$ gates them. In our implementation, the two conditions are obtained in parallel with the feedforward pass, which requires two additional copies of each of the two weight matrices. To avoid these copies, one could do this computation sequentially, using the synapses and layers of the feedforward pass three times per layer with different offsets and then routing the outputs to their respective destinations. This would, however, require more time steps.
4. **Error backpropagation**. Error calculation and gating by the start learning signal and inhibition by the stop learning signal are combined in time step 4 to calculate the last layer local gradients $\mathbf{d}_2^+$ and $\mathbf{d}_2^-$. From there, the local gradients are routed into the output layer and its copies for the last layer weight update. This happens in 2 phases: The positive local gradient $\mathbf{d}_2^+$ is connected without delay, leading to potentiation of the forward and backward last layer weight matrices in time step 5. The negative local gradient is connected with a delay of four time steps so that it arrives in the depression phase in time step 9. The opposite delays are used to get a weight update with the opposite sign for the connections to the $\mathbf{o}^{T-}$ layer, which is connected to the negative copy $-W_2^T$. See Fig. 2 for a visualization of this mechanism. Effectively, this leads to the last layer weight update.

$$\begin{aligned}\Delta W_2 &= \eta(H(\mathbf{t}-\mathbf{o}) \circ f'(W_2\mathbf{h}))\mathbf{h}^T \\ &\quad - \eta(H(\mathbf{o}-\mathbf{t}) \circ f'(W_2\mathbf{h}))\mathbf{h}^T,\end{aligned} \tag{27}$$

where the first term on the right-hand side is non-zero when the local gradient is positive, corresponding to an update happening in the potentiation phase, and the second term is nonzero when the local gradient is negative, corresponding to an update happening in the depression phase. The functions f and f' are as described in Equations (15) and (9).

The local gradient activation in the output layer does not only serve the purpose of updating the last layer weights, but it is also directly used to backpropagate the local gradients.

Propagating the signed local gradients backward through the network layers requires a positive and negative copy of the transposed weights, W_2^T and $-W_2^T$, which are the weight matrices of the synapses between the output layer $\mathbf{o}$ and the back-propagated local gradient layer $\mathbf{d}_1$, and between $\mathbf{o}^{T-}$ and $\mathbf{d}_1$, respectively. Here $\mathbf{o}^{T-}$ is an intermediate layer that is created exclusively for this purpose. The local gradient is propagated backward in two phases. The part of the local gradient that leads to potentiation is propagated back in time steps 5 to 7, and the part of the local gradient that leads to depression of the W_1 weights is propagated back in time steps 9 to 11. In time step 6, the potentiating local gradient is calculated in layer $\mathbf{d}_1$ as

$$\mathbf{d}_1^+ = H(W_2^T\mathbf{d}_2^+ + (-W_2^T)\mathbf{d}_2^-) \circ \mathbf{b}_\mathbf{h}, \tag{28}$$

and in timestep 10, the depressing local gradient is calculated in layer $\mathbf{d}_1$ as

$$\mathbf{d}_1^- = H((-W_2^T)\mathbf{d}_2^+ + W_2^T\mathbf{d}_2^-) \circ \mathbf{b}_\mathbf{h}. \tag{29}$$

Critically, this procedure does not simply separate the weights by sign but maintains a complete copy of the weights that is used to associate appropriate sign information to the back-propagated local gradient values. Note that here, the Heaviside activation function $H(\mathbf{x})$ is used rather than the binary activation function $f = H(\mathbf{x} - 0.5\,V_{\text{thr}})$, so that any positive gradient will induce an update of the weights. Any positive threshold in this activation will lead to poor performance by making the learning insensitive to small gradient values. The transposed weight copies must be kept in sync with their forward counterparts; therefore, the updates in the potentiation and depression phases are applied concurrently to the forward and backward weights.

So, in total, after each trial, the actual first layer weight update is the sum of four different parts:

$$\begin{aligned}\Delta W_1 = {}& \eta((H(W_2^T\mathbf{d}_2^+ + (-W_2^T)\mathbf{d}_2^-)) \circ \mathbf{b}_\mathbf{h})\mathbf{x}^T \\ &- \eta((H((-W_2^T)\mathbf{d}_2^+ + W_2^T\mathbf{d}_2^-)) \circ \mathbf{b}_\mathbf{h})\mathbf{x}^T.\end{aligned} \tag{30}$$

These four terms are necessary because, e.g., a positive error can also lead to depression if backpropagated through a negative weight matrix and vice versa.

Additional details can be found in the Supplementary Methods.

The nBP implementation on Loihi

Partitioning on the chip. To distribute the spike load over cores, neurons of each layer are approximately equally distributed over 96 cores of a single chip. This distribution is advantageous because only a few layers are active at each time step, and Loihi processes spikes within a core sequentially. In total, the network as presented here needs $2N_{\text{in}}+6N_{\text{hid}}+7N_{\text{out}}+12N_{\text{gat}}$ neurons. With $N_{\text{in}} = 400$, $N_{\text{hid}} = 400$, $N_{\text{out}} = 10$, and $N_{\text{gat}} = 1$, these are 3282 neurons and about 200 k synapses, most of which are synapses of the three plastic all-to-all connections between the input and the hidden layer.

Learning implementation. The learning rule on Loihi is implemented as given in Equation (23). Because the precision of the plastic weights on Loihi is maximally 8 bits with a weight range from − 256 − 254, we can only change the weight in steps of 2 without making the update non-deterministic. This is necessary for keeping the various copies of the weights in sync (hence the factor of 2 in Equation (23)). With $V_{\text{thr}} = 1024$, this corresponds to a learning rate of $\eta = \frac{2}{1024} \approx 0.002$. The learning rate can be changed by increasing the factor in the learning rule, which leads to a reduction in usable weight precision, or by changing the weight exponent (a factor that scales the weight by powers of 2) or V_{thr}, which changes the range of possible weights according to Equations (21) and (22). The range of possible learning rates is therefore limited and may reduce the achievable accuracy in this setting. Several learning rates (settings of V_{thr}) were tested with the result that the final accuracy is not very sensitive to small changes. The learning rate that yielded the best accuracy within the possible range is reported here. In the NxSDK API, the neuron traces that are used for the learning rule are `x0`, `y0`, and `r1` for $x(t)$, $y(t)$ and $r(t)$ in Equation (23) respectively. `r1` was used with a decay time constant of 1 so that it is only active in the respective time step, effectively corresponding to `r0`. To provide the r signal, a single reinforcement channel was used and activated by a regularly firing spike generator in steps 5 and 7.

Weight initialization. After the He initialization as described in the model section of the methods, the weights are mapped to Loihi weights according to Equations (21) and (22). Then, the weights are truncated to the interval [− 240, 240] and discretized to 8-bit resolution, i.e., steps of 2, by rounding them to the next valid number towards 0. The truncation was performed to discourage weights close to the margin of the 8-bit range as the positive and negative weight copies get out of sync when the negative weight goes to -256 while the positive is limited to 254. This happened rarely with a minor effect on learning and was corrected every epoch.

Power measurements. All Loihi power measurements are obtained using NxSDK version 0.9.9 on the Nahuku32 board ncl-ghrd-01. Both software API and hardware were provided by Intel Labs. All other probes, including the output probes, are deactivated. For the inference measurements, we use a network consisting only of the three feed-forward layers with non-plastic weights and the gating chain of four neurons. The power was measured for the first 10, 000 samples of the training set for the training measurements and all 10, 000 samples of the test set for the inference measurements.

ANN simulation

Tensorflow FMNIST baseline. Since there is no accuracy data for the binarized Fashion MNIST (FMNIST) dataset in the literature, we conducted experiments using a simple ANN on the regular and binarized datasets. The network achieved a test accuracy of 84.4% on the binarized dataset and 88.4% on the regular dataset. The FMNIST dataset comprises grayscale images of 10 different classes of clothing items. Based on these results, we conclude that binarizing the images leads to a reduction in accuracy. However, for the MNIST dataset, comparisons were made using the regular dataset, as binarization does not significantly impact the classification of handwritten digits.

The experiments were implemented in Python 3.8 using TensorFlow Keras 2.4.0. Like the network on Loihi, we use an architecture with a single hidden layer (784-400-10). The network was trained using sparse categorical cross-entropy loss and the stochastic gradient descent (SGD) optimizer with a fixed learning rate of 0.01 for 40 epochs. We report the learning rate that led to the best accuracy, architecture and optimizer were chosen to match our network on Loihi. However, the ANN was not quantized or constrained.

Data availability

No data was collected in this work. The MNIST and Fashion MNIST datasets used to evaluate the network are publicly available.

Code availability

The source code for the Loihi implementation of the nBP algorithm[105] is available on GitHub at https://github.com/lanl/spikingBackprop.

References

1. Linnainmaa, S. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *BIT Numer. Math.* **16**, 146–160 (1970).
2. Werbos, P. *Beyond Regression:" New Tools for Prediction and Analysis in the Behavioral Sciences*. https://perceptrondemo.com/assets/PJW_thesis_Beyond_Regression_1974-4b63aa5f.pdf (1974).
3. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. *Learning Internal Representations by Error Propagation*. http://www.dtic.mil/docs/citations/ADA164453 (1985).
4. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **21**, 335–346 (2020).
5. Roelfsema, P. R. & Holtmaat, A. Control of synaptic plasticity in deep cortical networks. *Nat. Rev. Neurosci.* **19**, 166–180 (2018).
6. Yamins, D. L. & DiCarlo, J. J. Using goal-driven deep learning models to understand sensory cortex. *Nat. Neurosci.* **19**, 356–365 (2016).
7. Mead, C. Neuromorphic electronic systems. *Proc. IEEE* **78**, 1629–1636 (1990).
8. Davies, M. et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro.* **38**, 82–99 (2018).
9. Esser, S. et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl Acad. Sci. USA* **113**, 11441–11446 (2016).
10. Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M. & Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. NeuroscI.* **11**, 682 (2017).
11. Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J. & Aimone, J. B. Training deep neural networks for binary communication with the Whetstone method. *Nat. Mach. Intell.* **1**, 86–94 (2019).
12. Grossberg, S. Competitive learning: from interactive activation to adaptive resonance. In *Connectionist Models and Their Implications: Readings from Cognitive Science.* (eds. Waltz, D. & Feldman, J. A.) 243–283 (Ablex Publishing Corp., Norwood, NJ, USA, 1988).
13. Crick, F. The recent excitement about neural networks. *Nature* **337**, 129–132 (1989).
14. Painkras, E. et al. SpiNNaker: A multi-core system-on-chip for massively-parallel neural net simulation. In *Proc. IEEE 2012 Custom Integrated Circuits Conference* (IEEE, 2012).
15. Schemmel, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proc. IEEE 2010 IEEE International Symposium on Circuits and Systems* (IEEE, 2010).
16. Qiao, N. et al. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Front. Neurosci.* **9**, 141 (2015).
17. Grossberg, S. Competitive learning: from interactive activation to adaptive resonance. *Cogn. Sci.* **11**, 23–63 (1987).
18. Liao, Q., Leibo, J. & Poggio, T. How important is weight symmetry in backpropagation? In *Proc. AAAI Conference on Artificial Intelligence.* (IEEE, 2016).
19. Bohte, S. M., Kok, J. N. & La Poutré, H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* **48**, 17–37 (2002).
20. Pfister, J.-P., Toyoizumi, T., Barber, D. & Gerstner, W. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.* **18**, 1318–1348 (2006).
21. Wu, Y., Deng, L., Li, G., Zhu, J. & Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* **12**, 331 (2018).
22. Zenke, F. & Ganguli, S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Comput.* **30**, 1514–1541 (2018).
23. Huh, D. & Sejnowski, T. J. Gradient descent for spiking neural networks. *Adv. Neural Inform. Process. Syst.* https://doi.org/10.48550/arXiv.1706.04698 (2018).
24. Zhang, W. & Li, P. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Adv. Neural Inform. Process. Syst.* **33**, 12022–12033 (2020).
25. Rasmussen, D. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics* **17**, 611–628 (2019).
26. Sengupta, A., Ye, Y., Wang, R., Liu, C. & Roy, K. Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* **13**, 95 (2019).
27. Shrestha, S. B. & Orchard, G. Slayer: Spike layer error reassignment in time. *Adv. Neural Inform. Process. Syst.* https://doi.org/10.48550/arXiv.1810.08646 (2018).
28. Boeshertz, G., Indiveri, G., Nair, M. & Renner, A. Accurate mapping of RNNs on neuromorphic hardware with adaptive spiking neurons. *Int. Conf. Neuromorphic Syst.* https://arxiv.org/pdf/2407.13534 (2024).
29. Rueckauer, B. et al. Nxtf: An API and compiler for deep spiking neural networks on intel Loihi. *ACM J. Emerg. Technoll. Comput. Syst. (JETC)* **18**, 1–22 (2022).
30. Bu, T. et al. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *Proc. 10th International Conference on Learning Representations* (ICLR, 2022).
31. Wang, Z. et al. Toward high-accuracy and low-latency spiking neural networks with two-stage optimization. *IEEE Transactions on Neural Networks and Learning Systems* (IEEE, 2023).
32. Stewart, K., Orchard, G., Shrestha, S. B. & Neftci, E. On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* 223–227 (IEEE, 2020).
33. DeWolf, T., Jaworski, P. & Eliasmith, C. Nengo and low-power AI hardware for robust, embedded neurorobotics. *Front. Neurorobot.* https://doi.org/10.3389/fnbot.2020.568359 (2020).
34. Frenkel, C., Lefebvre, M., Legat, J.-D. & Bol, D. A 0.086-mm ^{2}12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28 nm cmos. *IEEE Trans. Biomed. Circ. Syst.* **13**, 145–158 (2018).
35. Kim, J. K., Knag, P., Chen, T. & Zhang, Z. A 640m pixel/s 3.65 mw sparse event-driven neuromorphic object recognition processor with on-chip learning. In *2015 Symposium on VLSI Circuits (VLSI Circuits)* C50–C51 (IEEE, 2015).
36. Buhler, F. N. et al. A 3.43 tops/w 48.9 pj/pixel 50.1 nj/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm cmos. In *2017 Symposium on VLSI Circuits* C30–C31 (IEEE, 2017).
37. Park, J., Lee, J. & Jeon, D. 7.6 a 65nm 236.5 nj/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* 140–142 (IEEE, 2019).
38. Nandakumar, S. et al. Experimental demonstration of supervised learning in spiking neural networks with phase-change memory synapses. *Sci. Rep.* **10**, 1–11 (2020).
39. Frenkel, C., Legat, J.-D. & Bol, D. A 28 nm convolutional neuromorphic processor enabling online learning with spike-based retinas. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE, 2020).
40. Shrestha, A., Fang, H., Rider, D., Mei, Z. & Qui, Q. In-hardware learning of multilayer spiking neural networks on a neuromorphic processor. In *2021 58th ACM/ESDA/IEEE Design Automation Conference (DAC)* (IEEE, 2021).
41. Imam, N. & Cleland, T. A. Rapid online learning and robust recall in a neuromorphic olfactory circuit. *Nat. Mach. Intell.* **2**, 181–191 (2020).

42. Friedmann, S. et al. Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Trans. Biomed. Circuits Systems* **11**, 128–142 (2016).
43. Nandakumar, S. et al. Mixed-precision deep learning based on computational memory. *Front. Neurosci.* **14**, 406 (2020).
44. Pehle, C., Blessing, L., Arnold, E., Müller, E. & Schemmel, J. Event-based backpropagation for analog neuromorphic hardware. *arXiv* https://doi.org/10.48550/arXiv.2302.07141 (2023).
45. Payvand, M., Fouda, M. E., Kurdahi, F., Eltawil, A. M. & Neftci, E. O. On-chip error-triggered learning of multi-layer memristive spiking neural networks. *IEEE J. Emerg. Selected Topics Circuits Syst.* **10**, 522–535 (2020).
46. Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nat. Neurosci.* **24**, 1010–1019 (2021).
47. Bellec, G. et al. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* **11**, 3625 (2020).
48. Sacramento, J., Costa, R. P., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Adv. Neural Inform. Process. Syst.* https://doi.org/10.48550/arXiv.1810.11393 (2018).
49. Stork, D. G. Is backpropagation biologically plausible? In *International Joint Conference on Neural Networks*, 241–246 (IEEE Washington, DC, 1989).
50. Zipser, D. & Rumelhart, D. The neurobiological significance of the new learning models. In *Computational Neuroscience* (ed. Schwartz, E. L.) 192—200 (The MIT Press, 1990).
51. Lee, D.-H., Zhang, S., Fischer, A. & Bengio, Y. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases* (ed. Appice, A. et all.) 498–515 (Springer International Publishing, Cham, 2015).
52. O'Reilly, R. C. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Comput.* **8**, 895–938 (1996).
53. Kolen, J. & Pollack, J. Backpropagation without weight transport. In *Proc. 1994 IEEE International Conference on Neural Networks (ICNN'94)* 1375–1380 (IEEE, 1994).
54. Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T. & Tweed, D. B. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, (ed. Wallach, H. et al.) 32 (Curran Associates, Inc., 2019).
55. Boone, R., Zhang, W. & Li, P. Efficient biologically-plausible training of spiking neural networks with precise timing. In *International Conference on Neuromorphic Systems 2021*, ICONS 2021 (Association for Computing Machinery, NY, 2021).
56. Lillicrap, T. P., Cownden, D., Tweed, D. B. & Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* **7**, 1–10 (2016).
57. Liao, Q., Leibo, J. Z. & Poggio, T. How important is weight symmetry in backpropagation? In *Proc. Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 1837–1844 (AAAI Press, 2016).
58. Richards, B. A. & Lillicrap, T. P. Dendritic solutions to the credit assignment problem. *Curr. Opin. Neurobiol.* **54**, 28–36 (2019).
59. Max, K. et al. Learning efficient backprojections across cortical hierarchies in real time. *Nat. Mach. Intell.* https://doi.org/10.48550/arXiv.2212.1024 (2024).
60. O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T. & Pfeiffer, M. Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.* **7**, 178 (2013).
61. Kim, R., Li, Y. & Sejnowski, T. J. Simple framework for constructing functional spiking recurrent neural networks. *Proc. Natl Acad. Sci. USA* **116**, 22811–22820 (2019).
62. Neftci, E. O., Mostafa, H. & Zenke, F. Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* **36**, 61–63 (2019).
63. Izhikevich, E. M. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex* **17**, 2443–2452 (2007).
64. Sporea, I. & Grüning, A. Supervised learning in multilayer spiking neural networks. *Neural Comput.* **25**, 473–509 (2013).
65. Legenstein, R., Pecevski, D. & Maass, W. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput. Biol.* **4**, e1000180 (2008).
66. Frémaux, N. & Gerstner, W. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Front. Neural Circ.* **9**, 85 (2015).
67. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. *Neural Netwk.***111**, 47–63 (2019).
68. Sornborger, A., Tao, L., Snyder, J. & Zlotnik, A. A pulse-gated, neural implementation of the backpropagation algorithm. In *Proc. 7th Annual Neuro-inspired Computational Elements Workshop* 10 (ACM, 2019).
69. Sornborger, A., Wang, Z. & Tao, L. A mechanism for graded, dynamically routable current propagation in pulse-gated synfire chains and implications for information coding. *J. Comput. Neurosci.* **39**, 181–95 (2015).
70. Wang, Z., Sornborger, A. & Tao, L. Graded, dynamically routable information processing with synfire-gated synfire chains. *PLoS Comp. Biol.* **12**, 6 (2016).
71. Wang, C., Xiao, Z., Wang, Z., Sornborger, A. T. & Tao, L. A Fokker-Planck approach to graded information propagation in pulse-gated feedforward neuronal networks. *arXiv* https://doi.org/10.48550/arXiv.1512.00520 (2015).
72. Xiao, Z., Wang, B., Sornborger, A. & Tao, L. Mutual information and information gating in synfire chains. *Entropy* **20**, 102 (2018).
73. Shao, Y., Sornborger, A. & Tao, L. A pulse-gated, predictive neural circuit. In *Proc. 50th Asilomar Conference on Signals, Systems and Computers*, 1051–1055 (Pacific Grove, CA, 2016).
74. Shao, Y., Wang, B., Sornborger, A. T. & Tao, L. A mechanism for synaptic copy between neural circuits. *Neural Comput.* **31**, 1964–1984 (2019).
75. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
76. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv* https://doi.org/10.48550/arXiv:1708.07747 (2017).
77. Bengio, Y., Léonard, N. & Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv* https://doi.org/10.48550/arXiv:1308.3432 (2013).
78. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Binarized neural networks. In *Adv. Neural Inform. Process. Syst.* 4107–4115 (Barcelona, 2016).
79. Hebb, D. *The Organization of Behavior: A Neuropsychological Approach* Vol. 378 (John Wiley & Sons, 1949).
80. Sornborger, A. & Tao, L. Exact, dynamically routable current propagation in pulse-gated synfire chains. *arXiv* https://doi.org/10.48550/arXiv:1410.1115 (2014).
81. Senn, W. & Fusi, S. Learning only when necessary: better memories of correlated patterns in networks with bounded synapses. *Neural Comput.* **17**, 2106–2138 (2005).
82. Davies, M. et al. *Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook* (IEEE, 2021).
83. Stöckl, C. & Maass, W. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nat. Mach. Intell.* **3**, 230–238 (2021).

84. Baddeley, R. et al. Responses of neurons in primary and inferior temporal visual cortices to natural scenes. *Proc. R. Soc. B. Biol. Sci.* **264**, 1775–1783 (1997).
85. Göltz, J. et al. Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nat. Mach. Intell.* **3**, 823–835 (2021).
86. Comsa, I. M. et al. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 8529–8533 (IEEE, 2020).
87. Rueckauer, B. & Liu, S.-C. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5 (IEEE, Florence, Italy, 2018).
88. Neftci, E. et al. Synthesizing cognition in neuromorphic electronic systems. *Proc. Natl Acad. Sci.* **110**, E3468–E3476 (2013).
89. Baumgartner, S. et al. Visual pattern recognition with on on-chip learning: towards a fully neuromorphic approach. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE, 2020).
90. Riehle, A., Grün, S., Diesmann, M. & Aertsen, A. Spike synchronization and rate modulation differentially involved in motor cortical function. *Science* **278**, 1950–1953 (1997).
91. Abeles, M., Bergman, H., Margalit, E. & Vaadia, E. Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *J. Neurophysiol.* **70**, 1629–1638 (1993).
92. Hahnloser, R. H., Kozhevnikov, A. A. & Fee, M. S. An ultra-sparse code underlies the generation of neural sequences in a songbird. *Nature* **419**, 65–70 (2002).
93. Ikegaya, Y. et al. Synfire chains and cortical songs: temporal modules of cortical activity. *Science* **304**, 559–564 (2004).
94. Foster, D. J. & Wilson, M. A. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature* **440**, 680–683 (2006).
95. Rajan, K., Harvey, C. D. & Tank, D. W. Recurrent network models of sequence seneration and memory. *Neuron* **90**, 128–142 (2016).
96. Pang, R. & Fairhall, A. L. Fast and flexible sequence induction in spiking neural networks via rapid excitability changes. *Elife* **8**, e44324 (2019).
97. Malvache, A., Reichinnek, S., Villette, V., Haimerl, C. & Cossart, R. Awake hippocampal reactivations project onto orthogonal neuronal assemblies. *Science* **353**, 1280–1283 (2016).
98. Luczak, A., McNaughton, B. L. & Harris, K. D. Packet-based communication in the cortex. *Nat. Rev. Neurosci.* **16**, 745–755 (2015).
99. Simons, T. & Lee, D.-J. A review of binarized neural networks. *Electronics* **8**, 661 (2019).
100. Orchard, G. et al. Efficient neuromorphic signal processing with Loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, 254–259 (IEEE, 2021).
101. Lin, C.-K. et al. Programming spiking neural networks on Intel's Loihi. *Computer* **51**, 52–61 (2018).
102. Chen, G. K., Kumar, R., Sumbul, H. E., Knag, P. C. & Krishnamurthy, R. K. A 4096-neuron 1 m-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 Circ.* **54**, 992–1002 (2018).
103. Marr, D. & Poggio, T. From understanding computation to understanding neural circuitry. *Tech. Rep.* http://hdl.handle.net/1721.1/5782 (1976).
104. He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In *Proc. IEEE international conference on computer vision*, 1026–1034 (IEEE, 2015).
105. Sornborger, A. & Renner, A. Neuromorphic backpropagation algorithm software. *Comput. Softw*. https://doi.org/10.11578/dc.20220509.6 (2022).
106. Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V. & Modha, D. S. Backpropagation for energy-efficient neuromorphic computing. *Adv. Neural InforM. Proc. Syst.* **28**, 1117–1125 (2015).
107. Stromatias, E. et al. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (IEEE, 2015).

Acknowledgements
This work was carried out at Los Alamos National Laboratory under the auspices of the National Nuclear Security Administration of the U.S. Department of Energy under Contract No. 89233218CNA000001—SEEK: Scoping neuromorphic architecture impact enabling advanced sensing capabilities (A.Z.). Additional funding was provided by the LANL ASC Beyond Moore's Law program (A.T.S.), the ASCR/CRCNS program - Neural Signatures of Learning in the What and Where Pathways (A.T.S.), and by LDRD Reserve Project 20180737ER - Neuromorphic Implementation of the Backpropagation Algorithm (A.T.S., A.R.). L.T. acknowledges support from the Natural Science Foundation of China Grant No. 91232715. L.T. was partially supported by the STI2030-Major Projects (No. 2022ZD0204600) and the Natural Science Foundation of China through Grant No. 31771147. L.T. and A.R. thank the Los Alamos National Laboratory for its hospitality. A.R. acknowledges support from the Swiss National Science Foundation (SNSF) grant Ambizione PZOOP2 168183 ELMA and the University of Zurich postdoc grant FK-21-136. F.S. carried out work under the kind support of the Center for Nonlinear Studies fellowship and final preparations at the London Institute for Mathematical Sciences. The authors thank Intel Labs for providing support and access to the Loihi API and hardware.

Author contributions
The authors contributed equally to the methodology presented here. Alpha Renner adapted and implemented the algorithm on Intel's Loihi chip. Alpha Renner, Forrest Sheldon, and Anatoly Zlotnik formalized neuromorphic information-processing mechanisms, implemented the algorithm in simulation and hardware, and developed figures. All authors wrote the manuscript, with Renner, Sheldon, and Zlotnik doing the bulk of the writing. Andrew Sornborger and Anatoly Zlotnik supervised the research, and Sornborger and Louis Tao developed the concepts and algorithmic basis of the neuromorphic backpropagation algorithm and circuit structure. Sornborger and Tao conceived of the overall project.

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/s41467-024-53827-9.

Correspondence and requests for materials should be addressed to Andrew Sornborger.

Peer review information *Nature Communications* thanks James Aimone, Anthony Maida and Richard Boone reviewer(s) for their contribution to the peer review of this work. A peer review file is available.