Event-based backpropagation can compute exact gradients for spiking neural networks

Timo C. Wunderlich1,2,3,3 & Christian Pehle1,3

Spiking neural networks combine analog computation with event-based communication using discrete spikes. While the impressive advances of deep learning are enabled by training non-spiking artificial neural networks using the backpropagation algorithm, applying this algorithm to spiking networks was previously hindered by the existence of discrete spike events and discontinuities. For the first time, this work derives the backpropagation algorithm for a continuous-time spiking neural network and a general loss function by applying the adjoint method together with the proper partial derivative jumps, allowing for backpropagation through discrete spike events without approximations. This algorithm, EventProp, backpropagates errors at spike times in order to compute the exact gradient in an event-based, temporally and spatially sparse fashion. We use gradients computed via EventProp to train networks on the Yin-Yang and MNIST datasets using either a spike time or voltage based loss function and report competitive performance. Our work supports the rigorous study of gradient-based learning algorithms in spiking neural networks and provides insights toward their implementation in novel brain-inspired hardware.

How can we train spiking neural networks to achieve brain-like performance in machine learning tasks? The resounding success and pervasive use of the backpropagation algorithm in deep learning suggests an analogous approach. This algorithm computes the gradient of the neural network parameters with respect to a loss function that measures the network's performance in a given task. The parameters of the network are iteratively updated using the locally optimal direction given by the gradient.

Spiking neural networks have been referred to as the third generation of neural networks, superseding artificial neural networks as commonly used in deep learning and hold the promise for efficient and robust processing of event-based spatio-temporal data as found in biological systems. However, spiking models are not widely used in machine learning applications. At the same time, the development of spiking neuromorphic hardware receives increasing attention and learning in spiking neural networks is an active research subject, with a wide variety of proposed algorithms. A notorious issue in spiking neurons is the hard spiking threshold that does not permit a straight-forward application of differential calculus to compute gradients. Although exact gradients have been derived for special cases, this issue is commonly side-stepped by using smoothed or stochastic neuron models or by replacing the hard threshold function using a surrogate function, leading to the computation of surrogate gradients.

In contrast, this work provides an algorithm, EventProp, to compute the exact gradient for an arbitrary loss function defined using the state variables (spike times and membrane potentials) of a general recurrent spiking neural network composed of leaky integrate-and-fire neurons with hard thresholds. Since feed-forward architectures correspond to recurrent neural networks with block-diagonal weight matrices and convolutions can be represented as sparse linear transformations, deep feed-forward networks and convolutional networks are included as special cases.

The leaky integrate-and-fire neuron model describes a hybrid dynamical system that combines continuous dynamics between spikes with discontinuous state variable transitions at spike times. The computation of partial derivatives for hybrid dynamical systems is an established topic in optimal control theory. In hybrid systems, the time-dependent partial derivative $\frac{dx}{dp}(t)$ of a state variable $x$ with respect to a parameter $p$ generally experiences jumps at the points of discontinuity (see Fig. 1A,B). The relation between the partial derivatives before and after a given discontinuity was first studied in the 1960s. A more general theoretical framework was

---

1Kirchhoff-Institute for Physics, Heidelberg University, 69120 Heidelberg, Germany. 2Present address: Berlin Institute of Health, Charité–Universitätsmedizin, 10117 Berlin, Germany. 3These authors contributed equally: Timo C. Wunderlich and Christian Pehle. *email: timo.wunderlich@charite.de; christian.pehle@kip.uni-heidelberg.de
Figure 1. We derive the precise analogue to backpropagation for spiking neural networks by applying the adjoint method together with the jump conditions for partial derivatives at state discontinuities, yielding exact gradients with respect to loss functions based on membrane potentials or spike times. (A, B) Dynamical systems with parameter-dependent discontinuous state transitions typically have discontinuous partial derivatives of state variables with respect to system parameters, as is the case for the two examples shown here. Both examples model dynamics occurring on short timescales, namely inelastic reflection and the neuronal spike mechanism, using an instantaneous state transition. We denote quantities evaluated before and after a given transition by \( y_0 ^- \) and \( y_0 ^+ \). In A, a bouncing ball starts at height \( y_0 > 0 \) and is described by \( \ddot{y} = -g \) with gravitational acceleration \( g \). It is inelastically reflected as \( \dot{y}^+ = -0.8 \dot{y}^- \) as soon as \( y^- = 0 \) holds, causing the partial derivative with respect to \( y_0 \) to jump as \( \frac{\partial y^+}{\partial y_0} = -0.8 \frac{\partial y^-}{\partial y_0} \) (see first methods subsection). In B, a leaky integrate-and-fire neuron described by the system given in Table 1 with initial conditions \( I(0) = w, V(0) = 0 \) resets its membrane potential as \( V^+ = 0 \) when \( V^- = \vartheta \) holds, causing the partial derivative to jump as \( \frac{\partial V^+}{\partial w} = \left( \frac{\vartheta}{\tau_{\text{mem}}} + 1 \right) \frac{\partial V^-}{\partial w} \) (see methods for the full derivation). (C) Applying the adjoint method with partial derivative jumps to a network of leaky integrate-and-fire neurons (Table 1) yields the adjoint system (Table 2) that backpropagates errors in time. EventProp is an algorithm (Algorithm 1) returning the gradient of a loss function with respect to synaptic weights by computing this adjoint system. The forward pass computes the state variables \( V(t), I(t) \) and stores spike times \( t_{\text{post}} \) and each firing neuron's synaptic current. EventProp then performs the backward pass by computing the adjoint system backwards in time using event-based error backpropagation and gradient accumulation: each time a spike was transferred across a given synaptic weight in the forward pass, EventProp backpropagates the error signal represented by the adjoint variables \( \dot{\lambda}_V(t_{\text{post}}), \dot{\lambda}_I(t_{\text{post}}) \) of the post-synaptic (target) neuron and updates the corresponding component of the gradient by accumulating \( \dot{\lambda}_I(t_{\text{post}}) \), finally yielding sums as given in the figure.
Surrogate gradients use smooth activation functions for the purposes of backpropagation and have their computation in reverse time is analogous to the backpropagation of errors in discrete-time artificial neural regression tasks. Potential kernels, provide specialized algorithms tailored to specific loss functions and consider minimalistic (including negative threshold crossings), rely on the existence of analytic expressions for the post-synaptic per layer and this result was subsequently generalized to an arbitrary number of spikes as well as recurrent single leaky integrate-and-ﬁre neuron to learn a target spike train. Our work, as well as the works mentioned to continuous-time dynamical systems yields time-dependent adjoint variables.

Previous work. For a comprehensive survey of gradient-based approaches to learning in spiking neural networks, we refer the reader to review articles which discuss learning in deep spiking networks2,14,15, discuss learning along with the history and future of neuromorphic computing2 or focus on the surrogate gradient approach3. Surrogate gradients use smooth activation functions for the purposes of backpropagation and have been used to train spiking networks in a variety of settings16–19. This approach is typically derived by considering the Euler discretization of a spiking neural network where the Heaviside step function is used to couple neurons at every time step in the backward pass (all neurons backpropagate error signals at every time step), EventProp only requires computing vector-vector products at spike events (only the firing neuron receives backpropagated errors at a given spike time). In this way, EventProp leverages the sparseness of spike-based communication for both the forward and backward pass.

We demonstrate the training of spiking neural networks with a single hidden layer using EventProp and the Yin-Yang and MNIST datasets, resulting in competitive classiﬁcation performance.
At which a leaky integrate-and-fire neuron transitions from emitting a spike, the adjoint state variable \( \dot{I}_n = -\partial V_i / \partial t = -I \) is the unit vector with a 1 at index \( n \) to \( I_{n-1} = -R_i \), where \( \phi_{ij} \in \mathbb{R}^N \) is the unit vector with a 1 at index \( n \) and 0 at all other indices. We use \( - \) and \( + \) to denote quantities before and after a given spike.

### Table 1. The leaky integrate-and-fire spiking neural network model. Inbetween spikes, the vectors of membrane potentials \( V \) and synaptic currents \( I \) evolve according to the free dynamics. When some neuron \( n \in \{1...N\} \) crosses the threshold \( \vartheta \), the transition condition is fulfilled, causing a spike. This leads to a reset of the membrane potential as well as post-synaptic current jumps. \( W \in \mathbb{R}^{N \times N} \) is the weight matrix with zero diagonal and \( \phi_{ij} \in \mathbb{R}^N \) is the unit vector with a 1 at index \( n \) and 0 at all other indices. We use \( - \) and \( + \) to denote quantities before and after a given spike.

| Free dynamics | Transition condition | Jumps at transition |
|---------------|----------------------|---------------------|
| \( \tau_{\text{mem}} \frac{d}{dt} V = -V + I \) | \( (V)_n - \vartheta = 0 \) | \( (V^n)_n = 0 \) |
| \( \tau_{\text{syn}} \frac{d}{dt} I = -I \) | \( (V)_n \neq 0 \) for any \( n \) | \( I^+ = I^- + W_n \) |

### Table 2. The adjoint spiking network to Table 1 that computes the adjoint variable \( \lambda_I \) needed for the gradient [Eq. (2)]. The adjoint variables are computed in reverse time (i.e., from \( t = T \) to \( t = 0 \)) with \( t' = -\frac{t}{\tau} \) denoting the reverse time derivative. \( \lambda^0 \) experiences jumps at the spikes times \( t_{\text{post}} \), where \( n(k) \) is the index of the neuron that caused the kth spike. Computing this system amounts to the backpropagation of errors in time. The initial conditions are \( \lambda^0(V)(T) = \lambda_I(T) = 0 \) and we provide \( \lambda^0 \) in terms of \( \lambda_I \) because the computation happens in reverse time.

| Free dynamics | Transition condition | Jump at transition |
|---------------|----------------------|---------------------|
| \( \tau_{\text{mem}} \frac{d}{dt} \lambda^0 = -\lambda^0 - \frac{\partial V^0}{\partial V} \) | \( t - t_{\text{post}} = 0 \) for any \( k \) | \( \lambda^0(V)(n(k)) = \lambda^0(V)(n(k)) + \frac{1}{\tau_{\text{mem}}(V^0)(n(k))} \cdot \vartheta(\lambda^0(V)(n(k))) \) |
| \( \tau_{\text{syn}} \lambda^0 = -\lambda^0 + \lambda^0 \) | \( \frac{\partial V^0}{\partial p_{ij}} \) | \( + \left( W^\top (\lambda^0(V) - \lambda_I) \right)(n(k)) + \frac{\partial V^0}{\partial p_{ij}} I^0 - I^0 \) |

Above which derive exact gradients, applies the implicit function theorem to differentiate spike times with respect to synaptic weights. A different approach is to consider ratios of the neuronal time constants where analytic expressions for first spike times can be given and to derive the corresponding gradients, as done in27–30. Our work encompasses the contained methods to compute the gradient as special cases.

The seminal Tempotron model uses gradient descent to adjust the sub-threshold voltage maximum in a single neuron31 and has recently been generalized to the spike threshold surface formalism32 that uses the exact gradient of the critical thresholds \( \vartheta^k \) at which a leaky integrate-and-fire neuron transitions from emitting \( k \) to \( k-1 \) spikes; computing this gradient is not considered in this work. The adjoint method was recently used to optimize neural ordinary differential equations33 and neural jump stochastic differential equations34 as well as to derive the gradient for a smoothed spiking neuron model without reset44.

We first define the used spiking neuron model and then proceed to state our main results.

**Leaky integrate-and-fire neural network model.** We define a network of \( N \) leaky integrate-and-fire neurons with arbitrary (up to self-connections) recurrent connectivity (Table 1). We set the leak potential to zero and choose parameter-independent initial conditions. Note that the Spike-Response Model (SRM)35 with double-exponential or \( \alpha \)-shaped PSPs is generally an integral expression of the model given in Table 1 with corresponding time constants.

**Gradient via backpropagation.** Consider smooth loss functions \( l_V(V, t), l_p(p_{\text{post}}) \) that depend on the membrane potentials \( V \), time \( t \) and the set of post-synaptic spike times \( t_{\text{post}} \). The total loss is given by

\[
\mathcal{L} = l_p(p_{\text{post}}) + \int_0^T l_V(V(t), t)dt.
\]

Our main result is that the derivative of the total loss with respect to a specific weight \( w_{ji} = (W)_{ji} \) that connects pre-synaptic neuron \( i \) (the firing neuron) to post-synaptic neuron \( j \) (the receiving neuron) is given by a sum over the spikes caused by \( i \),

\[
\frac{d\mathcal{L}}{dw_{ji}} = -\tau_{\text{syn}} \sum_{\text{spikes from } i} (\lambda_I)_j,
\]

where \( \lambda_I \) is the adjoint variable (Lagrange multiplier) corresponding to the synaptic current \( I \). Equation (2) therefore samples the post-synaptic neuron’s adjoint variable \( (\lambda_I)_j \) at the spike times caused by neuron \( i \).

After the neuron dynamics given by Table 1 have been computed from \( t = 0 \) to \( t = T \), the adjoint state variable \( \lambda_I \) is computed in reverse time (i.e., from \( t = T \) to \( t = 0 \)) as the solution of the system of adjoint equations defined in Table 2. The dynamical system defined by Table 2 is the adjoint spiking network to the leaky integrate-and-fire network (Table 1) which backpropagates error signals at the spike times \( p_{\text{post}} \).
Equation (2) and Table 2 suggest a simple algorithm, EventProp, to compute the gradient (Algorithm 1). Notably, if the loss is voltage-independent (i.e., $l_V = 0$), the backward pass of the algorithm requires only the spike times $t_{\text{post}}$ and the synaptic current of the firing neurons at their respective firing times to be retained from the forward pass. The membrane potential at spike times is fixed to the threshold $\vartheta$ and therefore implicitly retained; the synaptic current therefore determines the temporal derivative of the membrane potential at the spike time, $\dot{V}$, and needs to be stored for the backward pass. The memory requirement of the algorithm scales as $O(S)$, where $S$ is the number of post-synaptic spikes in the network. A feed-forward architecture corresponds to a block matrix $W$ with each block being a strictly triangular matrix that connects two given layers. In that case, the forward and backward pass can be computed in a layer-wise fashion.

In case of a voltage-dependent loss $l_V \neq 0$, the algorithm has to store the non-zero components of $\frac{\partial l_V}{\partial V}$ along the forward trajectory. The loss $l_V$ may depend on the voltage at a discrete time $t_i$ using the Dirac delta, $l_V (V(t_i), t_i) = V(t_i) \delta(t_i - t_i)$, causing a jump of $\dot{V}$ of magnitude $\tau_{\text{mem}}$ at time $t_i$. Note that in many practical scenarios as found in deep learning, the loss $l_V$ depends only on the state of a constant number of neurons, irrespective of network size. If $l_V$ depends on the voltage of non-firing readout neurons, we have $l_V = l_V$ and the corresponding term in the jump given in Table 2 vanishes.

If $l_V$ is either zero or depends only on voltages at discrete points in time, EventProp can be computed in a purely event-based manner. Figure 2 illustrates how EventProp computes the gradient of a spike time based loss function for two leaky integrate-and-fire neurons where one neuron receives Poisson spike trains via 100 synapses and is connected to the other neuron via a single feed-forward weight $w$.
Simulation results. We demonstrate learning using EventProp using a custom event-based simulator and the Yin-Yang\textsuperscript{37} and MNIST\textsuperscript{38} datasets. In both cases, we use a single hidden layer and spike latency encoding of the input data. The Yin-Yang dataset is classified using the time to first spike of a layer of readout neurons while the MNIST dataset is classified using the voltage maxima of a layer of non-firing readout neurons. The simulator computes gradients using EventProp as described in Algorithm 1; specifically, it uses an event queue and root-bracketing to compute post-synaptic spike times in the forward pass (using exact integration of the membrane potential\textsuperscript{39}) and backpropagates errors by attaching error signals to spikes in the backward pass and using reverse traversal of the event queue. We optimized synaptic weights using the calculated gradients via the Adam optimizer\textsuperscript{40}, without clipping gradients.

By initializing synaptic weights such that the network started in a non-quiescent state, we found that no explicit regularization of firing rates was needed to obtain the reported results in both cases. Hyperparameters were optimized using Gaussian process optimization\textsuperscript{41} and manual tuning using the validation set of the respective dataset. The resulting parameters (see Table 3) were then evaluated using the test set.

### Table 3. Simulation parameters used for the results described in the main text

| Symbol   | Description                    | Value (Yin-Yang dataset) | Value (MNIST dataset) |
|----------|--------------------------------|--------------------------|-----------------------|
| $\tau_{\text{mem}}$ | Membrane time constant         | 20 ms                    | 20 ms                 |
| $\tau_{\text{syn}}$ | Synaptic time constant      | $\geq$ ms                | $\geq$ ms             |
| $\vartheta$ | Threshold               | 1                        | 1                     |
| $\theta$ | Input size                  | 5                        | 784                   |
|         | Hidden size                 | 200                      | 350                   |
|         | Output size                 | 3                        | 10                    |
| $\theta_{\text{bias}}$ | Bias time         | 0 ms                     | n/a                   |
| $\tau_{\text{max}}$ | Maximum time               | 30 ms                    | 20 ms                 |
|         | Hidden weights mean         | 1.5                      | 0.078                 |
|         | Hidden weights standard deviation | 0.78                 | 0.045                 |
|         | Output weights mean         | 0.93                     | 0.2                   |
|         | Output weights standard deviation | 0.1                     | 0.37                  |
|         | Minibatch size              | 32                       | 5                     |
|         | Optimizer                   | Adam                     | Adam                  |
| $f_1$   | Adam parameter              | 0.9                      | 0.9                   |
| $f_2$   | Adam parameter              | 0.999                    | 0.999                 |
| $e$     | Adam parameter              | $1 \times 10^{-8}$      | $1 \times 10^{-8}$    |
| $\eta$  | Learning rate               | $5 \times 10^{-3}$      | $5 \times 10^{-3}$    |
|         | Learning rate decay factor  | 0.95                     | 0.95                  |
|         | Learning rate decay step    | 1 epoch                  | 1 epoch               |
| $p_{\text{drop}}$ | Prob. of dropping input spike | n/a                     | 0.2                   |
| $\alpha$ | Regularization factor       | $3 \times 10^{-3}$      | n/a                   |
| $t_0$   | First loss time constant    | 0.5 ms                   | n/a                   |
| $t_1$   | Second loss time constant   | 0.4 ms                   | n/a                   |

### Algorithm 1 EventProp: Algorithm to compute eq. (2).

**Require:** Input spikes, losses $I_p$, $I_R$, parameters $W$, $\tau_{\text{mem}}$, $\tau_{\text{syn}}$, initial conditions $V(0)$, $I(0)$

**grad** ← 0

Compute neuron state trajectory (table 1) from $t = 0$ to $t = T$:

for all spikes $k$, store spike time $t\textsuperscript{\text{post}}_k$ and the firing neuron’s component of $I(t\textsuperscript{\text{post}}_k)$

if $I_p \neq 0$, also store $\partial I_p \partial V$

Compute adjoint state trajectory (table 2) from $t = T$ to $t = 0$:

accumulate

$\text{grad}_{ji} \leftarrow \text{grad}_{ji} - \tau_{\text{syn}}(\lambda)j$

for each spike from neuron $i$ to $j$

**return** grad
Yin-Yang dataset. The Yin-Yang dataset\(^\text{37}\) is a two-dimensional non-linearly separable dataset, with a shallow classifier achieving around 64% accuracy, and it therefore requires a hidden layer and backpropagation of errors for high classification accuracy. Consider that in contrast, the MNIST dataset can be classified using a linear classifier with at least 88% accuracy\(^\text{38}\).

Each two-dimensional data point of the dataset \((x, y)\) was transformed into four dimensions as \((x, 1-x, y, 1-y)\) and encoded using spike latencies in the interval \([0, t_{\text{max}}]\) (see Fig. 3D). We added a fixed bias spike at time \(t_{\text{bias}}\) for a total of five input spikes per data point. The resulting spike patterns were used as input to a two-layer network composed of leaky integrate-and-fire neurons. The output layer consisted of three neurons that each encoded one of the three classes, with each data point being assigned the class of the neuron that fired the earliest spike.

In analogy to\(^\text{27}\), we used a cross-entropy loss defined using the first output spike times per neuron, where \(t_{\text{post}}^{i}, k\) is the first spike time of neuron \(k\) for the \(i\)th sample, \(l(i)\) is the index of the correct label for the \(i\)th sample, \(N_{\text{batch}}\) is the number of samples in a given batch and \(\tau_0, \tau_1\) are hyperparameters of the loss function.

The first term corresponds to a cross-entropy loss function over the softmax function applied to the negative spike times (we use negative spike times as the class assignment is determined by the smallest spike time) and encourages an increase of the spike time difference between the label neuron and all other neurons. As the first term depends only on the relative spike times, the second term is a regularization term that encourages early spiking of the label neuron.

Training results are shown in Fig. 3. After training, the test accuracy was 98.1(2)% (mean and standard deviation over 10 different random seeds). This is comparable to the results shown in\(^\text{27}\), who report 95.9(7)% accuracy with a smaller hidden layer (200 vs. 120 neurons).

MNIST dataset. We encoded each digit of the MNIST dataset\(^\text{38}\) by transforming each of the \(28 \times 28 = 784\) pixels into spike latencies in the interval \([0, t_{\text{max}}]\) (pixels corresponding to a value of 0 or 1 out of 255 were not converted to spikes). The resulting spike patterns were used as input to a two-layer network composed of a hid-
den layer of leaky integrate-and-fire neurons and a readout layer of non-firing neurons. We used a cross-entropy loss function over the softmax function applied to the voltage maxima of the readout neurons (max-over-time), where $V_k(t)$ is the voltage trace of the $k$th readout neuron, $l(i)$ is the index of the correct label for the $i$th sample and $N_{\text{batch}}$ is the number of samples in a given batch. Note that we can write the maximum voltage as $\max_t V_k(t) = \int V_k(t)\delta(t - t_{\text{max}})\,dt$ with the time of the maximum $t_{\text{max}}$ and the Dirac delta $\delta$, allowing us to apply the chain rule to find the jump of $\dot{V}_k$ (cf. Table 2) at time $t_{\text{max}}$ (terms containing the distributional derivative of $\delta$ are always zero).

During training, input spikes were dropped with probability $p_{\text{drop}}$ in order to avoid overfitting. To obtain a validation set, we extracted and removed 5000 samples from the training set. Training results are shown in Fig. 4. After training, the test accuracy was 97.6(1)% (mean and standard deviation over 10 different random seeds). This represents competitive classification performance when compared with previously published results using spiking networks with a single, fully connected hidden layer (Table 4).

**Discussion**

We have derived and provided an algorithm (EventProp) to compute the gradient of a general loss function for a spiking neural network composed of leaky integrate-and-fire neurons. The parameter-dependent spike discontinuities were treated in a well-defined manner using the adjoint method in combination with partial derivative jumps, without approximations or smoothing operations. EventProp uses the resulting adjoint spiking network to backpropagate errors in order to compute the exact gradient. Its forward pass requires computing the spike times of pre-synaptic neurons that transmit spikes to post-synaptic neurons, while the backward pass backpropagates errors at these spike times using the reverse path (i.e., from post-synaptic to pre-synaptic neurons). The rigorous treatment of spike discontinuities in combination with an event-based computation of the exact gradient represent a significant conceptual advance in the study of gradient-based learning methods for spiking neural networks.

An apparent issue with gradient descent based learning in the context of spiking networks is that the magnitude of the gradient diverges at the critical points in parameter space (note the $\dot{\nu}^{-1}$ term in the jump term given in Table 2; this term diverges as the membrane potential becomes tangent to the threshold and we have $\dot{\nu} \to 0$). Indeed, this is a known issue in the broader context of optimal control of dynamical systems with parameter-dependent state transitions. While this divergence can be mitigated using gradient clipping in practice, exact
Table 2. Comparison of previously published classification results on the MNIST dataset for spiking neural networks that are trained using supervised learning with a single, fully connected (non-convolutional) hidden layer and temporal encoding of input data. The second column provides the number of hidden neurons.

| Publication                        | # Hidden | Test accuracy | Comments                          |
|------------------------------------|----------|---------------|-----------------------------------|
| This Work                          | 350      | 97.6(1)%      |                                   |
| Cramer et al.                      | 246      | 97.5(1)%      | Downsampled to 16 by 16 pixels    |
| Zenke and Vogels                   | 512      | 98.3(9)%      | Including recurrent connections   |
| Kheradpisheh and Masquelier        | 400      | 97.4(2)%      |                                   |
| Comsa et al.                       | 340      | 97.9% (Max.)  | Bias spikes at learned times      |
| Göltz et al.                       | 350      | 97.5(1)%      |                                   |
| Mostafa                            | 800      | 97.55%        |                                   |
| Neftci et al.                      | 500      | 97.77% (Max.) |                                   |
| Lee et al.                         | 800      | 98.71% (Max.) |                                   |

Table 4. Comparison of previously published classification results on the MNIST dataset for spiking neural networks that are trained using supervised learning with a single, fully connected (non-convolutional) hidden layer and temporal encoding of input data. The second column provides the number of hidden neurons.

gradients of commonly considered loss functions lead to learning dynamics that are ignorant with respect to these critical points and are therefore unable to selectively recruit additional spikes or dismiss existing spikes. In contrast, surrogate gradient methods continuously transmit errors across neurons and combine these with a non-linear function of the distance of the membrane potential to the threshold. It is therefore plausible that surrogate gradients represent a form of implicit regularization. Neftci et al. reports that the surrogate gradient approximates the true gradient in a minimalistic binary classification task while at the same time remaining finite and continuous along an interpolation path in weight space. Hybrid algorithms that combine the exact gradient with explicit regularization techniques could be a direction for future research and provide more principled learning algorithms as compared to ad-hoc replacements of threshold functions.

This work is based on the widely used leaky integrate-and-fire neuron model. Extensions to this model, such as fixed refractory periods, adaptive thresholds or multiple compartments can be treated in an analogous way. Learning algorithms as compared to ad-hoc replacements of threshold functions. While the absence of explicit solutions to the resulting differential equations can require the use of sophisticated numerical techniques for event-based simulations, such extensions can significantly enhance the computational capabilities of spiking networks. For example, uses adaptive thresholds to implement LSTM-like memory cells in a recurrent spiking neural network.

Neuromorphic hardware is an increasingly active research subject and implementing EventProp on such hardware is a natural consideration. The adjoint dynamics as given in Table 2 represent a type of spiking neural network which, instead of spiking dynamically, transmits errors at fixed times that are scaled with factors retained from the forward pass. Therefore, a neuromorphic implementation could store spike times and scaling factors locally at each neuron, where they could be combined with the dynamic error signal in the forward and backward pass (membrane potential and synaptic current). The resulting error signals could be distributed across the network using event-based communication schemes similar to, for example, the address-event representation protocol. As mentioned above, EventProp can be extended to multi-compartment neuron models as used in a recent neuromorphic architecture.

We used a two-layer feed-forward architecture to demonstrate learning using EventProp. The algorithm can, however, compute the gradient for arbitrary recurrent or convolutional architectures. Its computational and spatial complexity scales linearly with network size (assuming constant average firing rates per neuron), analogous to backpropagation in non-spiking artificial neural networks. The performance in more complex tasks therefore hinges on the general efficacy of gradient-based optimization in spiking networks. As mentioned above, gradients with respect to loss functions defined in terms of spike times or membrane potentials ignores the presence of critical parameters where spikes appear or disappear. We suggest that studying regularization techniques which deal with this fundamental issue in a targeted manner could enable powerful learning algorithms for spiking networks. By providing a theoretical foundation for backpropagation in spiking networks, we support future research that combines such regularization techniques with the computation of exact gradients.

Methods

Partial derivatives in a hybrid system. In the following, we use the example of a bouncing ball (Fig. 1A) to illustrate the calculation of partial derivatives in a dynamical system with state discontinuities. A general treatment of the topic is given in other literature. The discontinuities occurring in the leaky integrate-and-fire neuron are treated analogously in our derivation of the gradient (see corresponding methods subsection).

The differential equation describing the bouncing ball with height \( \dot{y} = -g \) (5) with gravitational acceleration \( g \). Defining the ball’s velocity as \( v = \dot{y} \), this is equivalent to a two-dimensional system

\[ \dot{v} = -g, \] (6a)
The initial conditions are

\[
\begin{align*}
\dot{y} &= v, \\
v(0) &= 0, \\
y(0) &= y_0
\end{align*}
\]

(6b)

where \( y_0 > 0 \) is the parameter of interest defining the ball's initial height. The given equations determine the state trajectory \( y(t) \) up to the moment of impact with the ground at \( y = 0 \). Likewise, the trajectories of the partial derivatives with respect to \( y_0 \) are given by differentiation of Eqs. (6) and (7)\textsuperscript{61},

\[
\begin{align*}
\frac{d}{dt} \frac{\partial v}{\partial y_0} &= 0, \\
\frac{d}{dt} \frac{\partial y}{\partial y_0} &= \frac{\partial v}{\partial y_0}
\end{align*}
\]

(8a)

(8b)

with initial conditions

\[
\begin{align*}
\frac{\partial v}{\partial y_0}(0) &= 0, \\
\frac{\partial y}{\partial y_0}(0) &= 1.
\end{align*}
\]

(9a)

(9b)

The state discontinuity occurs when the ball hits the ground and we have

\[
y = 0
\]

(10)

at the time of impact \( t_r \). The ball is inelastically reflected, losing a fraction of its energy. Specifically, the system is re-initialized as

\[
\begin{align*}
v^+ &= -0.8v^-, \\
y^+ &= y^-,
\end{align*}
\]

(11a)

(11b)

where \( - \) and \( + \) denote the state before and after the transition \( (v^\pm, y^\pm) \) are functions of \( t_r \) and \( y_0 \). Equations (10) and (11) together uniquely determine the partial derivatives after the reflection. The implicit function theorem\textsuperscript{62} applied to Eq. (10) guarantees (because \( v \neq 0 \)) the existence of a function \( t_r(y_0) \) that locally describes how the impact time changes with \( y_0 \), with its derivative given by

\[
\frac{dt_r}{dy_0} = -\frac{1}{\dot{y}^-} \frac{\partial y^-}{\partial y_0} = -\frac{1}{v^-} \frac{\partial y^-}{\partial y_0},
\]

(12)

Likewise, the implicit function theorem applies to Eq. (11) (because \( v \neq 0, \dot{v} \neq 0 \)), yielding after differentiation

\[
\begin{align*}
\frac{\partial v^+}{\partial y_0} + \dot{v}^+ \frac{dt_r}{dy_0} &= \frac{\partial v^-}{\partial y_0} + \dot{v}^- \frac{dt_r}{dy_0}, \\
\frac{\partial y^+}{\partial y_0} + \dot{y}^+ \frac{dt_r}{dy_0} &= \frac{\partial y^-}{\partial y_0} + \dot{y}^- \frac{dt_r}{dy_0}.
\end{align*}
\]

(13a)

(13b)

The partial derivatives after the transition can now be found by solving the system of equations given by Eqs. (11) and (12) and (13),

\[
\begin{align*}
\frac{\partial v^+}{\partial y_0} &= -0.8 \frac{\partial v^-}{\partial y_0} - 1.8g \frac{1}{v^-} \frac{\partial y^-}{\partial y_0}, \\
\frac{\partial y^+}{\partial y_0} &= -0.8 \frac{\partial y^-}{\partial y_0},
\end{align*}
\]

(14a)

(14b)

where we have used \( \ddot{y} = -g \). Equation (14) provides the initial conditions for the integration of the partial derivatives after the transition; subsequent ground impacts can be treated equivalently. Figure 1A illustrates the behaviour of \( y(t) \) and \( \frac{\partial y}{\partial y_0}(t) \) using trajectories calculated numerically using the equations given here.
**Adjoint method.** We apply the adjoint method to a continuous, first order system of ordinary differential equations and refer the reader to\(^{63,64}\) for a more general setting. Consider an \(N\)-dimensional dynamical system \(x : t \mapsto x(t) \in \mathbb{R}^N\) with parameters \(p \in \mathbb{R}^P\) defined by the system of implicit first order ordinary differential equations

\[
\dot{x} = F(x, p) = 0
\]  

(15)

and constant initial conditions \(G(x(0)) = 0\) where \(F, G\) are smooth vector-valued functions.

We are interested in computing the gradient of a loss that is the integral of a smooth function \(l\) over the trajectory of \(x\),

\[
\mathcal{L} = \int_0^T l(x, t) dt.
\]  

(16)

We have

\[
\frac{d\mathcal{L}}{dp_i} = \int_0^T \frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i} dt,
\]  

(17)

where \(\cdot\) is the dot product and the dynamics of the partial derivatives \(\frac{\partial x}{\partial p_i}\) are given by applying Gronwall’s theorem\(^{61}\),

\[
\frac{d}{dt} \frac{\partial x}{\partial p_i} = \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p_i} + \frac{\partial F}{\partial p_i}.
\]  

(18)

Computing \(x(t)\) along with \(\frac{\partial x}{\partial p_i}(t)\) using Eqs. (15) and (18) allows us to calculate the gradient in Eq. (17) in a single forward pass. However, this procedure can incur prohibitive computational cost. When considering a recurrent neural network with \(N\) neurons and \(P = N^2\) synaptic weights, computing \(\frac{\partial x}{\partial p_i}(t)\) for all parameters requires storing and integrating \(PN = N^2\) partial derivatives.

The adjoint method allows us to avoid computing \(PN\) partial derivatives in the forward pass by instead computing \(N\) adjoint variables \(\lambda(t)\) in an additional backward pass. We add a Lagrange multiplier \(\lambda : t \mapsto \lambda(t) \in \mathbb{R}^N\) that constrains the system dynamics as given in Eq. (15),

\[
\mathcal{L} = \int_0^T \left[l(x, t) + \lambda \cdot (\dot{x} - F(x, p))\right] dt.
\]  

(19)

Along trajectories where Eq. (15) holds, \(\lambda\) can be chosen arbitrarily without changing \(\mathcal{L}\) or its derivative. We get

\[
\frac{d\mathcal{L}}{dp_i} = \int_0^T \left[ \frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i} + \lambda \cdot \left( \frac{d}{dt} \frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial p_i} \right) \right] dt.
\]  

(20)

Using partial integration, we have

\[
\int_0^T \lambda \cdot \frac{d}{dt} \frac{\partial x}{\partial p_i} dt = - \int_0^T \lambda \cdot \frac{\partial x}{\partial p_i} dt + \left[ \lambda \cdot \frac{\partial x}{\partial p_i} \right]_0^T.
\]  

(21)

By setting \(\lambda(T) = 0\), the boundary term vanishes because we chose parameter independent initial conditions \((\frac{\partial x}{\partial p_i}(0) = 0)\). The gradient becomes

\[
\frac{d\mathcal{L}}{dp_i} = \int_0^T \left[ \left( \frac{\partial l}{\partial x} - \frac{\partial F}{\partial x} \lambda \right) \cdot \frac{\partial x}{\partial p_i} - \lambda \cdot \frac{\partial F}{\partial p_i} \right] dt.
\]  

(22)

By choosing \(\lambda\) to fulfill the adjoint differential equation

\[
\dot{\lambda} = \frac{\partial l}{\partial x} - \frac{\partial F}{\partial x} \lambda
\]  

(23)

we are left with

\[
\frac{d\mathcal{L}}{dp_i} = - \int_0^T \lambda \cdot \frac{\partial F}{\partial p_i} dt.
\]  

(24)

The gradient can therefore be computed using Eq. (24), where the adjoint state variable \(\lambda\) is computed from \(t = T\) to \(t = 0\) as the solution of the adjoint differential equation Eq. (23) with initial condition \(\lambda(T) = 0\). This corresponds to backpropagation through time (BPTT) in discrete time artificial neural networks.

**Derivation of gradient.** We apply the adjoint method (see previous methods subsection) to the case of a spiking neural network (i.e., a hybrid, discontinuous system with parameter dependent state transitions). The following derivation is specific to the model given in Table 1. A fully general treatment of (adjoint) sensitivity analysis in hybrid systems can be found in\(^6\) or\(^{10}\).
The differential equations defining the free dynamics in implicit form are

\[ f_V = \tau_{\text{mem}} \dot{V} + V - I = 0, \quad (25a) \]

\[ f_I = \tau_{\text{syn}} \dot{I} + I = 0, \quad (25b) \]

where \( f_V, f_I \) are again vectors of size \( N \). We now split up the loss integral in Eq. (1) at the spike times \( t_{\text{post}} \) and use vectors of Lagrange multipliers \( \lambda_V, \lambda_I \) that fix the system dynamics \( f_V, f_I \) between transitions.

\[
\frac{d\mathcal{L}}{dw_{ji}} = \frac{d}{dw_{ji}} \left[ l_p(t_{\text{post}}) + \sum_{k=0}^{N_{\text{post}}} \int_{t_k}^{t_{k+1}} \left[ \lambda_V(V, t) + \lambda_I \cdot f_V + \lambda_I \cdot f_I \right] dt \right],
\]

where we set \( t_0 = 0 \) and \( t_{N_{\text{post}}+1} = T \) and \( x \cdot y \) is the dot product of two vectors \( x, y \). Note that because \( f_V, f_I \) vanish along all considered trajectories, \( \lambda_V \) and \( \lambda_I \) can be chosen arbitrarily without changing \( \mathcal{L} \) or its derivative. Using Eq. (25) we have, as per Gronwall’s theorem\(^{26}\),

\[
\frac{\partial f_V}{\partial w_{ji}} = \tau_{\text{mem}} \frac{d}{dt} \frac{\partial V}{\partial w_{ji}} + \frac{\partial V}{\partial w_{ji}} - \frac{\partial I}{\partial w_{ji}},
\]

\[
\frac{\partial f_I}{\partial w_{ji}} = \tau_{\text{syn}} \frac{d}{dt} \frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}},
\]

where we have used the fact that the derivatives commute, \( \frac{\partial}{\partial w_{ji}} \frac{d}{dt} = \frac{d}{dt} \frac{\partial}{\partial w_{ji}} \) (the weights are fixed and have no time dependence). The gradient then becomes, by application of the Leibniz integral rule,

\[
\frac{d\mathcal{L}}{dw_{ji}} = \sum_{k=0}^{N_{\text{post}}} \left[ \int_{t_k}^{t_{k+1}} \left[ \frac{\partial \lambda_V}{\partial w_{ji}} \cdot \frac{\partial V}{\partial w_{ji}} + \lambda_V \cdot \left( \tau_{\text{mem}} \frac{d}{dt} \frac{\partial V}{\partial w_{ji}} + \lambda_I \cdot \frac{\partial V}{\partial w_{ji}} \right) + \lambda_I \cdot \left( \tau_{\text{syn}} \frac{d}{dt} \frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}} \right) \right] dt
\]

\[ + \frac{\partial l_p}{\partial w_{ji}} \frac{d}{dt} \int_{t_k}^{t_{k+1}} \frac{d}{dt} w_{ji} + \frac{d}{dt} \int_{t_k}^{t_{k+1}} \frac{d}{dt} w_{ji} \right].
\]

where \( t_k^{\text{V,k}} \) is the voltage-dependent loss evaluated before (−) or after (+) the transition and we have used that \( f_V = f_I = 0 \) along all considered trajectories. Using partial integration, we have

\[
\int_{t_k}^{t_{k+1}} \frac{d}{dt} \frac{\partial V}{\partial w_{ji}} dt = -\int_{t_k}^{t_{k+1}} \frac{\partial V}{\partial w_{ji}} \frac{\partial \lambda_V}{\partial w_{ji}} dt + \int_{t_k}^{t_{k+1}} \frac{\partial \lambda_V}{\partial w_{ji}} \frac{\partial V}{\partial w_{ji}} \frac{d}{dt} \frac{\partial \lambda_V}{\partial w_{ji}} dt,
\]

\[
\int_{t_k}^{t_{k+1}} \frac{d}{dt} \frac{\partial I}{\partial w_{ji}} dt = -\int_{t_k}^{t_{k+1}} \frac{\partial I}{\partial w_{ji}} \frac{\partial \lambda_I}{\partial w_{ji}} dt + \int_{t_k}^{t_{k+1}} \frac{\partial \lambda_I}{\partial w_{ji}} \frac{\partial I}{\partial w_{ji}} \frac{d}{dt} \frac{\partial \lambda_I}{\partial w_{ji}} dt.
\]

Collecting terms in \( \frac{\partial V}{\partial w_{ji}}, \) we have

\[
\frac{d\mathcal{L}}{dw_{ji}} = \sum_{k=0}^{N_{\text{post}}} \left[ \int_{t_k}^{t_{k+1}} \left[ \left( \frac{\partial \lambda_V}{\partial V} \frac{\partial V}{\partial w_{ji}} + \frac{\partial \lambda_V}{\partial V} \right) \cdot \frac{\partial V}{\partial w_{ji}} + \left( -\tau_{\text{mem}} \lambda_V + \lambda_I \right) \cdot \frac{\partial I}{\partial w_{ji}} \right] dt
\]

\[ + \frac{\partial l_p}{\partial w_{ji}} \frac{d}{dt} \int_{t_k}^{t_{k+1}} \frac{d}{dt} w_{ji} + \tau_{\text{mem}} \left[ \frac{\partial V}{\partial w_{ji}} \frac{\partial \lambda_V}{\partial w_{ji}} + \tau_{\text{mem}} \left[ \frac{\partial \lambda_I}{\partial V} \frac{\partial V}{\partial w_{ji}} + \frac{\partial \lambda_I}{\partial V} \right] \right] \frac{d}{dt} w_{ji} \right].
\]

Since the Lagrange multipliers \( \lambda_V(t), \lambda_I(t) \) can be chosen arbitrarily, this form allows us to set the dynamics of the adjoint variables between transitions. Since the integration of the adjoint variables is done from \( t = T \) to \( t = 0 \) in practice (i.e., reverse in time), it is practical to transform the time derivative as \( \frac{d}{dt} \rightarrow -\frac{d}{dt} \). Denoting the new time derivative by \( \tilde{\cdot} \), we have

\[
\tau_{\text{mem}} \dot{\lambda}_V = -\lambda_V - \frac{\partial \lambda_V}{\partial V},
\]

\[
\tau_{\text{syn}} \dot{\lambda}_I = -\lambda_I + \lambda_V.
\]

The integrand in Eq. (31) therefore vanishes along the trajectory and we are left with a sum over the transitions. Since the initial conditions of \( V \) and \( I \) are assumed to be parameter independent, we have \( \frac{\partial V}{\partial w_{ji}} = \frac{\partial I}{\partial w_{ji}} = 0 \) at \( t = 0 \). We set the initial condition for the adjoint variables to be \( \lambda_V(T) = \lambda_I(T) = 0 \) to eliminate the boundary term for \( t = T \). We are therefore left with a sum over transitions \( \xi_k \) evaluated at the transition times \( t_{k_{\text{post}}} \).
We proceed by deriving the relationship between the adjoint variables before and after each transition. Since the computation of the adjoint variables happens in reverse time in practice, we provide

\[ \frac{\partial I}{\partial t_{\text{post}}} + \frac{\partial I}{\partial w_{ji}} = \frac{\partial I}{\partial w_{ji}} \]

in terms of \( \frac{\partial I}{\partial w_{ji}} \).

Consider a spike caused by the \( n \)th neuron, with all other neurons \( m \neq n \) remaining silent. We start by first deriving the relationships between

\[ \frac{\partial V}{\partial t_{\text{post}}} + \frac{\partial V}{\partial w_{ji}}, \quad \frac{\partial V}{\partial t_{\text{post}}} - \frac{\partial V}{\partial w_{ji}}, \quad \frac{\partial I}{\partial t_{\text{post}}} + \frac{\partial I}{\partial w_{ji}}, \quad \frac{\partial I}{\partial t_{\text{post}}} - \frac{\partial I}{\partial w_{ji}}. \]

Membrane potential transition. By considering the relations between \( V^+, V^- \) and \( \dot{V}^+, \dot{V}^- \), we can derive the relation between \( \frac{\partial V^+}{\partial w_{ji}} \) and \( \frac{\partial V^-}{\partial w_{ji}} \) at each spike. Each spike at \( t_{\text{post}} \) is triggered by a neuron's membrane potential crossing the threshold. We therefore have, at \( t_{\text{post}} \),

\[ (V^-)_n - \vartheta = 0. \]

This relation defines \( s_{\text{post}} \) as a differentiable function of \( w_{ji} \) via the implicit function theorem (illustrated in Fig. 5, see also65), under the condition that \((V^-)_n \neq 0\). Differentiation of this relation yields

\[ \left( \frac{\partial V^-}{\partial w_{ji}} \right)_n + (V^-)_n \frac{\partial s_{\text{post}}}{\partial w_{ji}} = 0. \]

Since we only allow transitions for \((V^-)_n \neq 0\), we have

\[ \frac{\partial s_{\text{post}}}{\partial w_{ji}} = -\frac{1}{(V^-)_n} \left( \frac{\partial V^-}{\partial w_{ji}} \right)_n. \]

Note that corresponding relations were previously used to derive gradient-based learning rules for spiking neuron models20–22,26,66; in contrast to the suggestion in20, Eq. (37) is not an approximation but rather an exact relation at all non-critical parameters and invalid at all critical parameters.

Because the spiking neuron's membrane potential is reset to zero, we have

\[ (V^+)_n = 0. \]

This implies by differentiation

\[ \left( \frac{\partial V^+}{\partial w_{ji}} \right)_n + (V^+)_n \frac{\partial s_{\text{post}}}{\partial w_{ji}} = 0. \]
Using Eq. (37), this allows us to relate the partial derivative after the spike to the partial derivative before the spike,

\[
\left( \frac{\partial V^+}{\partial w_{ji}} \right)_n = \left( \frac{\partial V^-}{\partial w_{ji}} \right)_n.
\]

(40)

Since we have \((V^+)_m = (V^-)_m\) for all other, non-spiking neurons \(m \neq n\), it holds that

\[
\left( \frac{\partial V^+}{\partial w_{ji}} \right)_m + (V^+)_m \frac{dI_{\text{post}}}{dw_{ji}} = \left( \frac{\partial V^-}{\partial w_{ji}} \right)_m + (V^-)_m \frac{dI_{\text{post}}}{dw_{ji}}.
\]

(41)

Because the spiking neuron \(n\) causes the synaptic current of all neurons \(m \neq n\) to jump by \(w_{mn}\), we have

\[
\tau_{\text{mem}}(\dot{V}^+)_m = \tau_{\text{mem}}(\dot{V}^-)_m + w_{mn}
\]

(42)

and therefore get with Eq. (36)

\[
\frac{\partial V^+}{\partial w_{ji}}_m = \frac{\partial V^-}{\partial w_{ji}}_m - \tau_{\text{mem}}^{-1} w_{mn} \frac{dI_{\text{post}}}{dw_{ji}}.
\]

(43)

\[
\frac{\partial V^-}{\partial w_{ji}}_m + \frac{1}{\tau_{\text{mem}}} w_{mn}\left( \frac{\partial V^-}{\partial w_{ji}} \right)_n = 0.
\]

(44)

**Synaptic current transition.** The spiking neuron \(n\) causes the synaptic current of all neurons \(m \neq n\) to jump by the corresponding weight \(w_{mn}\). We therefore have

\[
(I^+)_m = (I^-)_m + w_{mn}.
\]

(45)

By differentiation, this relation implies the consistency equations for the partial derivatives \(\frac{\partial I}{\partial w_{ji}}\) with respect to the considered weight \(w_{ji}\),

\[
\left( \frac{\partial I^+}{\partial w_{ji}} \right)_m + (I^+)_m \frac{dI_{\text{post}}}{dw_{ji}} = \left( \frac{\partial I^-}{\partial w_{ji}} \right)_m + (I^-)_m \frac{dI_{\text{post}}}{dw_{ji}} + \delta_{in}\delta_{jm}.
\]

(46)

where \(\delta_{ji}\) is the Kronecker delta. Because

\[
\tau_{\text{syn}}(\dot{I}^+)_m = \tau_{\text{syn}}(\dot{I}^-)_m - w_{mn},
\]

(47)

we get with Eq. (36)

\[
\left( \frac{\partial I^+}{\partial w_{ji}} \right)_m = \left( \frac{\partial I^-}{\partial w_{ji}} \right)_m + \tau_{\text{syn}}^{-1} w_{mn} \frac{dI_{\text{post}}}{dw_{ji}} + \delta_{in}\delta_{jm}.
\]

(48)

\[
\left( \frac{\partial I^-}{\partial w_{ji}} \right)_m = \left( \frac{\partial I^-}{\partial w_{ji}} \right)_n - \frac{1}{\tau_{\text{syn}}} w_{mn}\left( \frac{\partial V^-}{\partial w_{ji}} \right)_n + \delta_{in}\delta_{jm}.
\]

(49)

With \((I^+)_n = (I^-)_n\) and \((I^+)_n = (I^-)_n\), we have

\[
\left( \frac{\partial I^+}{\partial w_{ji}} \right)_n = \left( \frac{\partial I^-}{\partial w_{ji}} \right)_n.
\]

(50)

Using the relations of the partial derivatives from Eqs. (37), (40), (44), (49) and (50) in the transition equation Eq. (34), we now derive relations between the adjoint variables. Collecting terms in the partial derivatives and writing the index of the spiking neuron for the \(k\)th spike as \(n(k)\), we have

\[
\xi_k = \sum_{m \neq n(k)} \left[ \tau_{\text{mem}}(\dot{V}^- - \dot{V}^+)_m \frac{\partial V^-}{\partial w_{ji}}_m + \tau_{\text{syn}}(\dot{V}^- - \dot{V}^+)_m \left( \frac{\partial I^-}{\partial w_{ji}}_m - \tau_{\text{syn}} \delta_{in(k)} \delta_{jm} (\dot{I}^+_j)_m \right) \right]
\]

\[
+ \left( \frac{\partial V^-}{\partial w_{ji}} \right)_{n(k)} \left[ \tau_{\text{mem}} (\dot{V}^- - \dot{V}^+ \delta_{in(k)} \dot{I}^+_j) \right]_{n(k)}
\]

\[
+ \frac{1}{(V^-)_{n(k)}} \left( \sum_{m \neq n(k)} w_{mn(k)}(\dot{I}^- - \dot{I}^+_j)_m - \frac{\partial l_p}{\partial I_{\text{post}}} + l_p - l^- \right) \right] \right|_{I_{\text{post}}^{\text{out}}}
\]

(51)
This form dictates the jumps of the adjoint variables for the spiking neuron \( n \) and all other, silent neurons \( m \),

\[
(\lambda^-_V)_n = \frac{(V^+)_n}{(V^-)_n} (\lambda^+_V)_n + \frac{1}{\tau_{\text{mem}}(V^-)_n} \sum_{m \neq n} w_{\text{syn}} (\lambda^+_V)_m + \frac{\partial I_p}{\partial I_k} + l_{\text{post}}^- - l_V^-,
\]

(52a)

\[
(\lambda^-_V)_m = (\lambda^+_V)_m,
\]

(52b)

\[
\lambda^+_V = \lambda^+_V^-.
\]

(52c)

With these jumps, the gradient reduces to

\[
\frac{dL}{dw_{ji}} = -\tau_{\text{syn}} \sum_{k=1}^{N_{\text{syn}}} \delta_{m(k)} (\lambda^-_I)_j
\]

\[
= -\tau_{\text{syn}} \sum_{\text{spikes from } i} (\lambda^-_I)_j.
\]

(53)

(54)

**Summary.** The free adjoint dynamics between spikes are given by Eq. (32) while spikes cause jumps given by Eq. (52). The gradient for a given weight samples the post-synaptic neuron’s \( \lambda^-_V \) when spikes are transmitted across the corresponding synapse [Eq. (53)]. Since we can identify, with \( (V^+)_n - (V^-)_n = \tau_{\text{mem}} \theta_i \),

\[
\frac{(V^+)_n}{(V^-)_n} = \frac{(V^+)_n - (V^-)_n}{(V^-)_n} + 1 = \frac{\partial}{\tau_{\text{mem}}(V^-)_n} + 1
\]

the derived solution is equivalent to Eq. (2) and Table 2.

**Fixed Input Spikes.** If a given neuron \( i \) is subjected to a fixed pre-synaptic spike train across a synapse with weight \( w_{\text{input}} \), the transition times are fixed and the adjoint variables do not experience jumps. The gradient simply samples the neuron’s \( \lambda^-_I \) at the times of spike arrival,

\[
\frac{dL}{dw_{\text{input}}} = -\tau_{\text{syn}} \sum_{\text{input spikes}} (\lambda^-_I)_i.
\]

(56)

**Coincident spikes.** The derivation above assumes that only a single neuron of the recurrent network spikes at a given \( \lambda^+_V^- \). In general, coincident spikes may occur. If neurons \( a \) and \( b \) spike at the same time and the times of their respective threshold crossing vary independently as function of \( w_{ji} \), the derivation above still holds, with both neuron’s \( \lambda^-_V \) experiencing a jump as in Eq. (52a).

**Code availability**

Code to reproduce the shown results will be made available at https://github.com/eventprop.

Received: 13 November 2020; Accepted: 28 May 2021

Published online: 18 June 2021

**References**

1. Maass, W. Networks of spiking neurons: the third generation of neural network models. Neural Netw. 10, 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7 (1997).

2. Roy, K., Jaiswal, A. & Panda, P. Towards spike-based machine intelligence with neuromorphic computing. Nature 575, 607–617. https://doi.org/10.1038/s41586-019-1677-2 (2019).

3. Nefci, 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. Mag. 36, 51–63 (2019).

4. Barton, P. I. & Lee, C. K. Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. ACM Trans. Model. Comput. Simul. 12, 256–289. https://doi.org/10.1145/643120.643122 (2002).

5. Rozenwasser, E. & Yusupov, R. Sensitivity of Automatic Control Systems. Control Series (CRC Press, 2019).

6. De Backer, W. Jump conditions for sensitivity coefficients. IFAC Proceedings Volumes 1, 168–175. https://doi.org/10.1016/S1474-6670(17)69603-4 (1964) International Symposium on Sensitivity Methods in Control Theory, Dubrovnik, Yugoslavia, August 31–September 5 (1964).

7. Rozenwasser, E. General sensitivity equations of discontinuous systems. Automatika i telemekhanika 3, 52–56 (1967).

8. Galán, S., Feherly, W. F. & Barton, P. I. Parametric sensitivity functions for hybrid discrete/continuous systems. Appl. Numer. Math. 31, 17–47. https://doi.org/10.1016/S0168-9274(98)00125-1 (1999).

9. LeCun, Y., Touresky, D., Hinton, G. & Sejnowski, T. A theoretical framework for back-propagation. In Proceedings of the 1988 Connectionist Models Summer School, vol. 1, 21–28 (1988).

10. Serban, R. & Recuero, A. Sensitivity analysis for hybrid systems and systems with memory. J. Comput. Nonlinear Dyn. https://doi.org/10.1115/1.4044028 (2019).

11. Pleiss, G. et al. Memory-efficient implementation of densenets. arXiv: 1707.06990 (2017).
12. Kumar, R., Purohit, M., Srivikana, Z., Vec, E. & Wang, I. Efficient rematerialization for deep networks. In Advances in Neural Information Processing Systems Vol. 32 (eds Wallach, H. et al.) (Curran Associates Inc, 2019).
13. Ojika, D. et al. Addressing the memory bottleneck in AI model training. arXiv:2003.08732 (2020).
14. Pfeffer, M. & Pfeil, T. Deep learning with spiking neurons: opportunities and challenges. Front. Neurosci. 12, 774. https://doi.org/10.3389/fnins.2018.00774 (2018).
15. Tavaazee, A., Ghodrat, M., Kheradpisheh, S. R., Masquelier, T. & Maida, A. Deep learning in spiking neural networks. Neural Netw. 111, 47–63. https://doi.org/10.1016/j.neunet.2018.12.002 (2019).
16. Esser, S. K. et al. Convolutional networks for fast, energy-efficient neuromorphic computing. Proc. Natl. Acad. Sci. 113, 11441–11446. https://doi.org/10.1073/pnas.1604850113 (2016).
17. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R. & Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. In Advances in Neural Information Processing Systems 787–797 (2018).
18. Zenke, F. & Ganguli, S. Superspike: supervised learning in multi-layer spiking neural networks. Neural Comput. 30, 1514–1541 (2018).
19. Shrestha, S. B. & Orchard, G. Slayer: Spike layer error reassignment in time. In The 2010 International Joint Conference on Neural Networks (IJCNN), 1–8. https://doi.org/10.1109/IJCNN.2010.5596914 (2010).
20. Xu, Y., Zeng, X., Han, L. & Yang, J. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. Neural Netw. 43, 99–113. https://doi.org/10.1016/j.neunet.2013.02.003 (2013).
21. Kuroe, Y. & Ueyama, T. Learning methods of recurrent spiking neural networks based on adjoint equations approach. In The 2010 International Joint Conference on Neural Networks (IJCNN), 1–8. https://doi.org/10.1109/IJCNN.2010.5596914 (2010).
22. Kuroe, Y. & Iima, H. A learning method for synthesizing spiking neural oscillators. In The 2006 IEEE International Joint Conference on Neural Network Proceedings, 3882–3886. https://doi.org/10.1109/IJCNN.2006.246885 (2006).
23. Selvaratnam, K., Kuroe, Y. & Mori, T. Learning methods of recurrent spiking neural networks. Trans. Inst. Syst. Control Inf. Eng. 13, 95–104. https://doi.org/10.5687/isic2013.3_95 (2000).
24. Florian, R. The chronotron: a neuron that learns to fire temporally precise spike patterns. PLoS ONE 7, 1–27. https://doi.org/10.1371/journal.pone.0040233 (2012).
25. Göltz, J. et al. Fast and deep: energy-efficient neuromorphic learning with first-spike (2019).
26. 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 (2020).
27. Mostafa, H. Supervised learning based on temporal coding in spiking neural networks. IEEE Trans. Neural Netw. Learn. Syst. https://doi.org/10.1109/tnnls.2017.2726060 (2017).
28. Kheradpisheh, S. R. & Masquelier, T. Temporal backpropagation for spiking neural networks with one spike per neuron. Int. J. Neural Syst. 30, 2050027. https://doi.org/10.1142/S0129065720500276 (2020).
29. Güttig, R. & Sompolinsky, H. The tempotron: a neuron that learns spike timing-based decisions. Nat. Neurosci. 9, 420–428. https://doi.org/10.1038/30893-6080(9700011)-72 (2006).
30. Güttig, R. Spiking neurons can discover predictive features by aggregate-label learning. Science, 351, 115. https://doi.org/10.1126/science.aab4113 (2016).
31. Chen, T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. In Advances in Neural Information Processing Systems6571–6583 (2018).
32. Iia, S. & Besson, A. R. Neural jump stochastic differential equations. In Advances in Neural Information Processing Systems9843–9854 (2019).
33. Huh, D. & Sejnnowski, T. J. Gradient descent for spiking neural networks. In Advances in Neural Information Processing Systems Vol. 31 (eds Bengio, S. et al.) 1433–1443 (Curran Associates Inc, 2018).
34. Gerstner, W. & Kastler, W. Spiking Neuron Models: Single Neurons, Populations (Single Neurons, Populations, Plasticity (Cambridge University Press, Plasticity: Spiking Neuron Models, 2002).
35. Kriener, L. Yin-yang dataset. https://doi.org/10.1109/30893-6080(9700011)-74 (2020).
36. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324 (1998).
37. Rollets, S. & Diesmann, M. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biol. Cybern. 81, 381–402. https://doi.org/10.1007/s00422-0000011-75 (1999).
38. Kngma, D. P. & Ba, J. Adam: a method for stochastic optimization (2014).
39. Pedregosa, F. et al. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830 (2011).
40. Cramer, B. et al. Surrogate gradients for analog neuromorphic computing (2021). arXiv:2006.07239.
41. Zenke, F. & Vogels, T. P. The remarkable robustness of surrogate gradient learning for training complex function in spiking neural networks. Neural Comput. 33, 899–925. https://doi.org/10.1162/neco_a_01367 (2021).
42. Nefci, E. G., Augustin, C., Paul, S. & Detorakis, G. Event-driven random back-propagation: enabling neuromorphic deep learning machines. Front. Neurosci. 11, 324. https://doi.org/10.3389/fnins.2017.00324 (2017).
43. Lee, J. H., Delbruck, T. & Pfeiffer, M. Training deep spiking neural networks using backpropagation. Front. Neurosci. 10, 508. https://doi.org/10.3389/fnins.2016.00508 (2016).
44. Pei, J. et al. Towards artificial general intelligence with hybrid tianjic chip architecture. Science 572, 106. https://doi.org/10.1126/science.aae1982 (2019).
45. Billaudelle, S. et al. Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate. In 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 1–5. https://doi.org/10.1109/ISCAS545731.2020.9180741 (2020).
46. Feldmann, J., Younghood, N., Wright, C., Bhaskaran, H. & Pernice, W. All-optical spiking neuromorphic networks with self-learning capabilities. Nature 569, 208–214. https://doi.org/10.1038/s41586-019-1677-20 (2019).
47. Boyat, I. et al. Neuromorphic computing with multi-memristive synapses. Nat. Commun. https://doi.org/10.1038/s41467-018-04933-y (2017).
Acknowledgements
We would like to express gratitude to Eric Mülller and Johannes Schemmel for discussions, continued support and encouragement during the preparation of this work. We thank Laura Kriener and Julian Göltz for their support regarding time to first spike experiments and helpful discussions. We thank Korbinian Schreiber and Mihai Petrovici for helpful discussions.

Author contributions
C.P. conceived of the presented idea and outlined the application of sensitivity analysis to spiking neuron models. C.P. and T.W. derived the adjoint equations for LIF neurons and the resulting EventProp algorithm. T.W. implemented the event based simulation code. T.W. conducted and analyzed the presented simulations. C.P. and T.W. wrote and edited the manuscript.

Funding
The research has received funding from the EC Horizon 2020 Framework Programme under Grant Agreements 785907 and 945539 (HBP) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy EXC 2181/1 - 390900948 (the Heidelberg STRUCTURES Excellence Cluster) and was financially supported by the Joachim Herz foundation.

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to T.C.W. or C.P.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2021