frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 08 February 2023
DOI 10.3389/fnins.2023.1110444

OPEN ACCESS

EDITED BY
Thomas Nowotny,
University of Sussex, United Kingdom

REVIEWED BY
Christian Pehle,
Heidelberg University, Germany
Axel Laborieux,
Friedrich Miescher Institute for Biomedical Research (FMI), Switzerland

*CORRESPONDENCE
Felix C. Bauer
✉ felix.bauer@synsense.ai

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 28 November 2022
ACCEPTED 09 January 2023
PUBLISHED 08 February 2023

CITATION
Bauer FC, Lenz G, Haghighatshoar S and Sheik S (2023) EXODUS: Stable and efficient training of spiking neural networks. *Front. Neurosci.* 17:1110444. doi: 10.3389/fnins.2023.1110444

EXODUS: Stable and efficient training of spiking neural networks

Felix C. Bauer*, Gregor Lenz, Saeid Haghighatshoar and Sadique Sheik

SynSense AG, Zurich, Switzerland

Introduction: Spiking Neural Networks (SNNs) are gaining significant traction in machine learning tasks where energy-efficiency is of utmost importance. Training such networks using the state-of-the-art back-propagation through time (BPTT) is, however, very time-consuming. Previous work employs an efficient GPU-accelerated backpropagation algorithm called SLAYER, which speeds up training considerably. SLAYER, however, does not take into account the neuron reset mechanism while computing the gradients, which we argue to be the source of numerical instability. To counteract this, SLAYER introduces a gradient scale hyper parameter across layers, which needs manual tuning.

Methods: In this paper, we modify SLAYER and design an algorithm called EXODUS, that accounts for the neuron reset mechanism and applies the Implicit Function Theorem (IFT) to calculate the correct gradients (equivalent to those computed by BPTT). We furthermore eliminate the need for ad-hoc scaling of gradients, thus, reducing the training complexity tremendously.

Results: We demonstrate, via computer simulations, that EXODUS is numerically stable and achieves comparable or better performance than SLAYER especially in various tasks with SNNs that rely on temporal features.

KEYWORDS

spiking neural network (SNN), backpropagation (BP) algorithm, supervised learning, neuromorphic computing, neuromorphic algorithms, learning algorithm, computational efficiency, neuromorphic engineering

1. Introduction

Spiking Neural Networks (SNNs) are a class of biologically-inspired networks with single bit activations, fine-grained temporal resolution and highly sparse outputs. Their memory makes them especially suitable for sequence tasks and their sparse output promises extremely low power consumption, especially when combined with an event-based sensor and asynchronous hardware (Göltz et al., 1995; Davies et al., 2021). SNNs have thus garnered considerable attention for machine learning tasks that aim to achieve low power consumption and/or biological realism (Cao et al., 2015; Diehl and Cook, 2015; Roy et al., 2019; Panda et al., 2020; Comşa et al., 2021).

SNNs are notoriously difficult to train due to the highly non-linear neuron dynamics, extreme output quantization and potential internal state resets, even for relatively simple neuron models such as Integrate-and-Fire (IF) or Leaky-Integrate-and-Fire (LIF) (Burkitt, 2006; Gerstner, 2021). Different approaches and learning rules have thus emerged to train SNNs. Biologically-inspired local learning rules (Choe, 2013; Lee et al., 2018; Lobov et al., 2020) do not rely on a global error signal, but often fail to scale to larger architectures (Bartunov et al., 2018). Methods that work directly with spike timings to calculate gradients bypass the issue of non-differentiable spikes, but are typically limited to time-to-first spike encoding (Göltz et al., 1995; Bohte et al., 2000; Mostafa, 2017). A notable exception to this has been proposed by Wunderlich and Pehle (2021), where gradients of threshold crossings are determined in continuous-time on an event-by-event basis, which results in reduced memory footprint. A parallelized, GPU-based implementation has been proposed recently (Nowotny et al., 2022).

Fueled by the success of deep learning, surrogate gradient methods more recently have paved the way for flexible gradient-based optimization in SNNs with the aim to close the accuracy gap to ANN counterparts (Esser et al., 2016; Bellec et al., 2018; Neftci et al., 2019; Safa et al., 2021).

Surrogate gradient methods make use of a smoothed output activation (usually a function of internal state variables) during the backward pass to approximate the discontinuous activation. This is well-supported in modern deep learning frameworks and allows the direct application of back-propagation through time (BPTT) to train SNNs. This alone would make it feasible to train SNNs successfully were it not for the high temporal resolution needed in SNNs. Activation is typically very sparse across time because data from an event-based sensor has a native time resolution of micro-seconds. This makes it necessary to simulate input using a lot of time steps on today's von Neumann machines, which work in discrete time. A naive implementation of BPTT with T discrete-time steps and N fully connected neurons incurs a computational complexity of order $O(N^2T)$ and a memory overhead of $O(NT)$ on such architectures (Martín-Sánchez et al., 2022), which makes training SNNs tremendously slow and computationally expensive.

To alleviate this issue, Shrestha and Orchard (2018) propose SLAYER, an algorithm in which gradients are back-propagated across layers as usual, but they are computed jointly in time such that the complexity due to *sequential* back-propagation in time is fairly eliminated by highly parallelized and GPU-accelerated joint gradient computation. Mathematically speaking, SLAYER can be seen as gradient computation for a modified forward computation graph underlying the chain rule, in which all the time dynamics are contracted into a single node (see Section 2.3 for further details). However, this comes at the cost of creating a loop in the computation graph where the gradients cannot be back-propagated via chain rule. To solve this issue, SLAYER ignores a term, known as reset kernel, which models the effect of output spike generation on the internal neuron potential. In e-prop (Bellec et al., 2020), a learning algorithm for recurrent SNNs, the same term among others is omitted in the gradient computaion for reasons of biological plausibility.

As a result, SLAYER yields a considerable speed-up for training SNNs at the cost of the deviation of gradients from what would be computed by BPTT, as well as some numerical instability. SLAYER typically deals with this instability by tweaking a hyperparameter which scales the gradient magnitude. This needs considerable hand-tuning and scales unfavorably to deeper architectures and longer time sequences. For completeness, it is possible to reduce computational complexity of gradient compuation in SNNs in other specialized implementations by optimizing the forward call (Knight et al., 2021), computing sparse gradients (Perez-Nieves and Goodman, 2021) or taking advantage of CUDA graph replay.

In this paper we propose EXODUS (EXact computation Of Derivatives as Update to SLAYER), in which we address numerical issues induced by SLAYER and compute gradients equivalently to what BPTT computes, while at the same time achieving a significant speedup of one order of magnitude in comparison to a non-optimized implementation. We achieve this by applying the Implicit Function Theorem (IFT) in a similar approach as Blondel et al. (2021), hence resolving the loopy structure in each layer's computation graph. Examples of other work that make use of the IFT to find gradients in the context of machine learning include (Scellier and Bengio, 2017; Bai et al., 2019), which formulate optimization processes for deep learning models as fixed point problems, as well as above mentioned (Wunderlich and Pehle, 2021).

In summary, our contributions are as follows:

(i) We improve the SLAYER algorithm by taking into account the reset response of neurons during the backward pass.

(ii) We compute gradients that are equivalent to BPTT and can be back-propagated through each layer.

(iii) We eliminate the need for ad-hoc scaling of gradients, needed for solving the numerical instability of SLAYER, thus, reducing the training complexity tremendously.

(iv) We demonstrate, via numerical simulations, that EXODUS is robust to changes in gradient scaling and achieves comparable or better performance than SLAYER in various tasks using snn that rely on temporal features.

2. Preliminaries and background

2.1. Implicit Function Theorem

In many problems in statistics, mathematics, control theory, machine learning, etc. the state of a problem is represented in terms of a collection of variables. However, in many cases, these variables are correlated due to existence of constraints. Here, we are interested in a setting where one may have a collection of $m + n$ variables and a set of m equations (equality constraints) connecting them together. Since there are m equations, one may hope to solve for m variables, at least locally, as a function of the remaining n variables. It is conventional to call the first m variables *dependent* and the remaining n variables *independent* as the latter may vary (at least locally) independently of one another while the values of the former depend on the specific choice of those n independent variables.

The Implicit Function Theorem (IFT) provides rigorous conditions under which this is possible and specifies when the m dependent variables are differentiable with respect to n independent ones.

Theorem 2.1 (Implicit Function Theorem). Let $\boldsymbol{\phi} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ be a differentiable function, let $\mathcal{Z} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}^m : \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = 0\}$ be the zero-set of $\boldsymbol{\phi}$, and let $(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathcal{Z}$ be an arbitrary point in $\mathcal{Z}$. If the $m \times m$ matrix $\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}}(\boldsymbol{x}_0, \boldsymbol{y}_0)$ is non-singular, i.e., $\det\left(\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}}(\boldsymbol{x}_0, \boldsymbol{y}_0)\right) \neq 0$, then,

- there is an open neighborhood $\mathcal{N}_\mathrm{x}$ around $\boldsymbol{x}_0$ and an open neighborhood $\mathcal{N}_\mathrm{y}$ around $\boldsymbol{y}_0$ such that $\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{y})$ is non-singular for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{N} := \mathcal{N}_\mathrm{x} \times \mathcal{N}_\mathrm{y}$ [including of course the original point $(\boldsymbol{x}_0, \boldsymbol{y}_0)$].
- there is a function $\boldsymbol{\psi} : \mathcal{N}_\mathrm{x} \to \mathcal{N}_\mathrm{y}$ such that $(\boldsymbol{x}, \boldsymbol{\psi}(\boldsymbol{x}))$ belongs to the zero-set $\mathcal{Z}$, i.e., $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{\psi}(\boldsymbol{x})) = 0$, for all $\boldsymbol{x} \in \mathcal{N}_\mathrm{x}$. Therefore, $\boldsymbol{y} = \boldsymbol{\psi}(\boldsymbol{x})$ can be written as function of $\boldsymbol{x}$.
- (Chain rule) $\boldsymbol{\psi}$ is a differentiable function of $\boldsymbol{x}$ in $\mathcal{N}_\mathrm{x}$ and

$$\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}} = 0, \tag{1}$$

which from the non-singularity of $\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}}$ yields

$$\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{x}} = -\left(\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{y}}\right)^{-1} \cdot \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}}. \tag{2}$$

Remark 2.2. Note that here, for simplicity, we denoted the independent and dependent variables with $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$. In general, one may choose any disjoint subsets of the variables of size m and n as dependent and independent variables and verify the conditions of the IFT.

In Appendix 1, we provide examples to illustrate how IFT is applied for computing the derivative of the dependent variables with respect to independent ones. In particular, we show that intuitive ad-hoc application of the chain rule in a loopy computation graph and neglecting the conditions of ift may indeed yield wrong results.

2.2. Spike Response Model

Neuron dynamics in SNNs can be described by a state-space model where the internal state of each neuron depends on both its current input and its previous states. When applying the conventional back-propagation algorithm, therefore, the gradients need to be back-propagated not only through layers of the network but also through time. This paper builds upon SLAYER (Spike LAYer Error Reassignment) (Shrestha and Orchard, 2018), an algorithm for training feedforward SNNs. SLAYER is based on the spike response model (SRM) (Gerstner, 2021), where the state of a spiking neuron at each time instant n is described by its membrane potential $u[n]$, given by

$$u[n] = \sum_i w_i(\boldsymbol{\epsilon} * s_i^{in})[n] + (\boldsymbol{\nu} * s^{out})[n-1], \tag{3}$$

$$s^{out}[n] = f_s(u[n]). \tag{4}$$

Here $s_i^{in}[n]$ and w_i denote the input spikes received from the i-th pre-synaptic neuron and the corresponding weight. Furthermore, $\boldsymbol{\epsilon}$ and $\boldsymbol{\nu}$ denote the spike response and reset kernel of the neuron (with $*$ denoting the convolution operation in time) to the incoming and outgoing spikes, where a discrete delay of size 1 is introduced for the reset kernel $\boldsymbol{\nu}$ to defer the effect of outgoing spikes to the next time instant.

Outgoing spikes are obtained from the membrane potential through a memory-less binary spike generating function $f_s : \mathbb{R} \to \{0, 1\}$ where $f_s(u[n]) = 1$ when the membrane potential $u[n]$ reaches or exceeds the neuron firing threshold $\theta > 0$, and is 0 otherwise. Since f_s is not differentiable, a common solution for obtaining well-defined gradients for training SNNs is to use f_s in the forward pass to produce outgoing spikes but replace it with a differentiable function in the backward pass, where the gradients are computed through back-propagation. This is known as the surrogate gradient method in SNN literature. Different surrogate gradients have been proposed such as piecewise linear (Bohte, 2011; Esser et al., 2016; Bellec et al., 2018, 2020), tanh (Woźniak et al., 2020), fast sigmoids (Zenke and Ganguli, 2018), or exponential functions (Shrestha and Orchard, 2018). In this paper, with some abuse of notation, we denote the surrogate gradient by $f_s'(.)$. Our derivations are valid for any f_s' as long as $f_s'(u[n])$ is well-defined for all feasible values of the membrane potential $u[n]$ at all time instants n.

2.3. Vectorized network model

As in Shrestha and Orchard (2018), we focus on a feedforward network architecture with L layers. Using Equations (3, 4) and applying vectorization, we may write the forward dynamics of a layer l with N_l neurons and input weights $W^{(l-1)} \in \mathbb{R}^{N_l \times N_{l-1}}$ equivalently as:

$$\boldsymbol{a}^{(l)}[n] = \mathbf{W}^{(l-1)}\boldsymbol{s}^{(l-1)}[n], \tag{5}$$

$$\mathbf{z}^{(l)}[n] = (\boldsymbol{\epsilon} * \boldsymbol{a}^{(l)})[n], \tag{6}$$

$$\boldsymbol{u}^{(l)}[n] = \mathbf{z}^{(l)}[n] + (\boldsymbol{\nu} * \boldsymbol{s}^{(l)})[n-1], \tag{7}$$

$$\boldsymbol{s}^{(l)}[n] = f_s(\boldsymbol{u}^{(l)}[n]), \tag{8}$$

Here, $\boldsymbol{a}^{(l-1)}[n]$ represents the weighted input spikes (output spikes of previous layer) $\boldsymbol{s}^{(l-1)}$. Filtering/smoothing out by the neuron spike response $\boldsymbol{\epsilon}$ yields the post-synaptic response $\mathbf{z}^{(l)}$.

We take $\{\boldsymbol{s}^{(l-1)}[n] : n \in [T]\}$ and $\{\boldsymbol{s}^{(l)}[n] : n \in [T]\}$ as the input and output of a specific layer $l \in [L]$ across T time instants, where we used the short-hand notation $[N] = \{1, 2, \ldots, N\}$. The model output is given by $\{\boldsymbol{s}^{(L)}[n] : n \in [T]\}$. We also define the loss as $\mathcal{L}(\boldsymbol{s}^{(L)}[0], ..., \boldsymbol{s}^{(L)}[T-1])$ in terms of the network output over all time instants $[T]$.

The computational graph of the described network model has a directed acyclic graph structure in spatio-temporal dimensions, as illustrated in Figure 1. Therefore, the gradients can be propagated from the loss $\mathcal{L}$ to the trainable weights backward across the layers (spatial) and also time instants (temporal). However, as explained above, this approach is computationally slow.

The SLAYER algorithm introduced by Shrestha and Orchard (2018) avoids this by vectorizing the variables over time, as illustrated in Figure 1. Instead of assigning to each state variable at each point in time an individual node in the computational graph, here every node corresponds to a state across all time instants. This, however, introduces loops in the computational graph due to the mutual dependence of the vectorized variables $\boldsymbol{u}^{(l)}[.]$ and $\boldsymbol{s}^{(l)}[.]$ in Equations (7, 8). This prohibits back-propagating through these variables. To solve this issue, SLAYER ignores the reset kernel $\boldsymbol{\nu}$ [effect of output spikes on neuron potential in Equation (7)] in the calculation of its gradients. Apart from this omission, the algorithm is equivalent to ours, as described in detail below.

3. Derivation of EXODUS gradients

3.1. Vector back-propagation

We calculate gradients precisely by taking into account the reset kernel neglected by SLAYER. To do so, we apply the ift to back-propagate the gradients through the loopy computation graph. As the loops occur only between the variables $\boldsymbol{u}^l[.]$ and $\boldsymbol{s}^{(l)}[.]$ within the same layer, we apply IFT to each layer $l \in [L]$ individually. More specifically, by applying the chain rule, we back-propagate the gradients from the last layer to compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}^{(l)}[.]}$. We show that, although there is a loop between $\boldsymbol{u}^{(l)}[.]$ and $\boldsymbol{s}^{(l)}[.]$, we are still able to compute $\frac{\partial \boldsymbol{s}^{(l)}[.]}{\partial \boldsymbol{a}^{(l)}[.]}$ (see Figure 1). Multiplying this gradient with $\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}^{(l)}[.]}$ enables us to compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(l)}[.]}$, which is the gradient back-propagated to the previous layer.

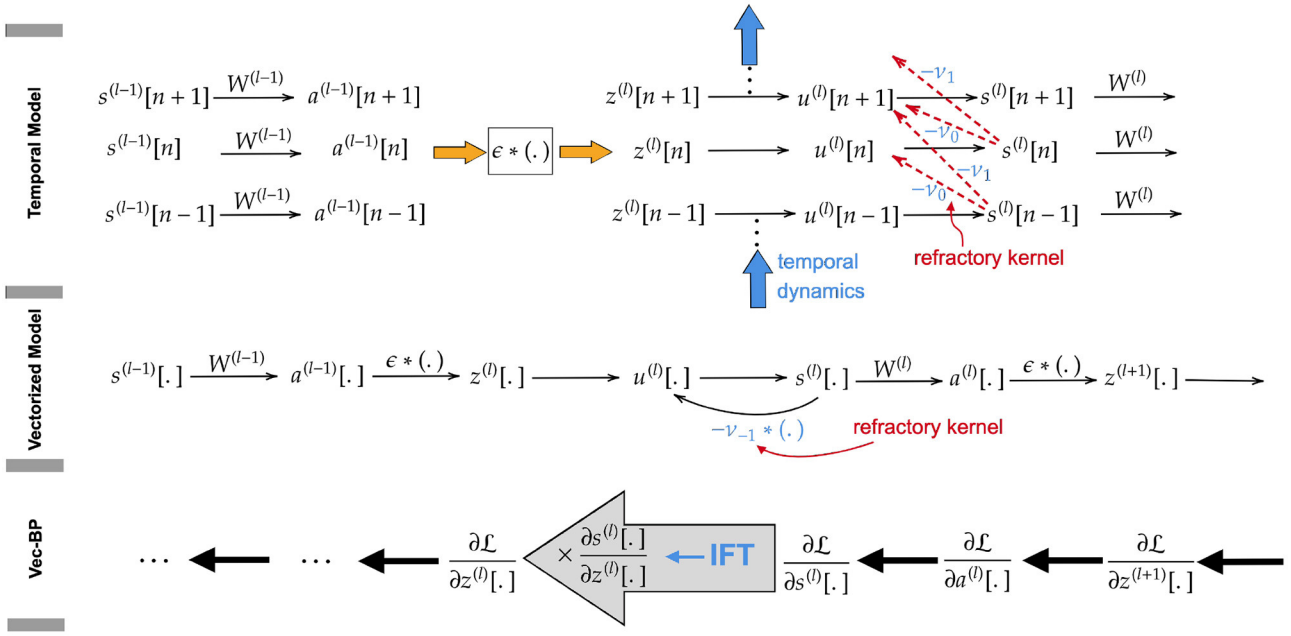

FIGURE 1
(Upper) Directed acyclic computation graph along temporal and spatial dimension. **(Middle)** Loopy computation graph after vectorization along the temporal dimension. The loop is introduced by the reset kernel and is ignored in the backward path of SLAYER. **(Lower)** Vectorized backward path, applying IFT to back-propagate the gradients through the loopy computation graph.

3.2. Derivations for a generic model

To apply IFT, we need to specify the underlying equations and also the dependent and independent variables. Let us consider (Equations 7, 8) for all $n \in [T]$:

$$\boldsymbol{\phi}_u^{(l)}[n] := \boldsymbol{u}^{(l)}[n] - \boldsymbol{z}^{(l)}[n] - (\boldsymbol{\nu} * \boldsymbol{s}^{(l)})[n-1] = 0,$$
$$\boldsymbol{\phi}_s^{(l)}[n] := \boldsymbol{s}^{(l)}[n] - f_s(\boldsymbol{u}^{(l)}[n]) = 0.$$

This is a system of $2N_lT$ equations in terms of three vectorized variables $\boldsymbol{u}^{(l)}[.], \boldsymbol{s}^{(l)}[.], \boldsymbol{z}^{(l)}[.]$, each of dimension N_lT. Therefore, by a simple dimensionality check, we can see that we may write two of these variables (dependent) as a differentiable function of the other variable (independent) provided that the conditions of IFT hold.

To pass the chain rule through the loopy computation graph at layer l, we need to compute $\frac{\partial \boldsymbol{s}^{(l)}[.]}{\partial \boldsymbol{z}^{(l)}[.]}$ (see, e.g., Figure 1). This implies that we need to treat $\boldsymbol{z}^{(l)}[.]$ as the independent and $(\boldsymbol{s}^{(l)}[.], \boldsymbol{u}^{(l)}[.])$ as the dependent variables. For simplicity of notation, we denote these independent and dependent variables and the corresponding equations by

$$\boldsymbol{x}^{(l)} := \left\{\boldsymbol{z}^{(l)}[n] : n \in [T]\right\}, \quad \boldsymbol{\psi}^{(l)} := \left\{\boldsymbol{u}^{(l)}[n], \boldsymbol{s}^{(l)}[n] : n \in [T]\right\},$$
$$\boldsymbol{\phi}^{(l)} := \left\{\boldsymbol{\phi}_u^{(l)}[n], \boldsymbol{\phi}_s^{(l)}[n] : n \in [T]\right\}.$$

Let $\mathbf{J}^{\psi^{(l)}} = \frac{\partial \boldsymbol{\phi}^{(l)}}{\partial \boldsymbol{\psi}^{(l)}} \in \mathbb{R}^{2N_lT \times 2N_lT}$ and $\mathbf{J}^{x^{(l)}} = \frac{\partial \boldsymbol{\phi}^{(l)}}{\partial \boldsymbol{x}^{(l)}} \in \mathbb{R}^{2N_lT \times N_lT}$ be the Jacobian matrices of the equations $\boldsymbol{\phi}^{(l)}$ with respect to the dependent and independent variables, respectively. Let us also define $\mathbf{G}^{(l)} = \frac{\partial \boldsymbol{\psi}^{(l)}}{\partial \boldsymbol{x}^{(l)}} \in \mathbb{R}^{2N_lT \times N_lT}$ as the gradients of dependent variables with respect to the independent ones.

With this, we verify the IFT conditions: (i) All the equations are differentiable, provided that f_s is a differentiable function. (ii) By bringing $\mathbf{J}^{\psi^{(l)}}$ into a row-echelon form, we can prove that $\mathbf{J}^{\psi^{(l)}}$ is non-singular. We refer to Appendix 2 for a detailed derivation. The gradients $\mathbf{G}^{(l)}$ are then found by solving IFT Equation (1) $\mathbf{J}^{\psi^{(l)}} \cdot \mathbf{G}^{(l)} = -\mathbf{J}^{x^{(l)}}$.

Applying a forward substitution method (see Appendix 2) yields the desired gradients:

$$\boldsymbol{\sigma}_m^{(l)}[n] := \left(\frac{\partial \mathbf{s}^{(l)}[.]}{\partial \mathbf{z}^{(l)}[.]}\right)_{n,m} = \frac{\partial \mathbf{s}^{(l)}[n]}{\partial \mathbf{z}^{(l)}[m]}$$
$$= \begin{cases} 0 & n < m \\ \mathbf{f}'^{(l)}[n] & n = m \\ \mathbf{f}'^{(l)}[n]\left(\boldsymbol{\nu} * \boldsymbol{\sigma}_m^{(l)}\right)[n-1] & n > m, \end{cases} \tag{9}$$

where $\mathbf{f}'^{(l)}[n]$ is the *diagonal matrix* holding the surrogate gradients $(\mathbf{f}'^{(l)}[n])_{ii} = f_s'(u_i^{(l)}[n])$.

As we explained before, computing $\frac{\partial \mathbf{s}^{(l)}[.]}{\partial \mathbf{z}^{(l)}[.]}$ *via* IFT allows us to push the back-propagation (chain rule) through loops in the computational graph.

The remaining steps needed for back-propagation are quite straightforward and are obtained from Equations (5, 6) as follows:

$$\frac{\partial \mathbf{z}^{(l)}[m]}{\partial \mathbf{a}^{(l)}[n]} = \frac{\partial (\boldsymbol{\epsilon} * \mathbf{a}^{(l)})[m]}{\partial \mathbf{a}^{(l)}[n]} = \frac{\partial \sum_{k=1}^{m} \epsilon_{m-k} \cdot \mathbf{a}^{(l)}[k]}{\partial \mathbf{a}^{(l)}[n]} = \epsilon_{m-n} \cdot \boldsymbol{I} \tag{10}$$

$$\frac{\partial \boldsymbol{a}^{(l)}[m]}{\partial \mathbf{s}^{(l-1)}[n]} = \delta_{m,n} \cdot \mathbf{W}^{(l-1)} \tag{11}$$

$$\frac{\partial \boldsymbol{a}^{(l)}[m]}{\partial \mathbf{W}^{(l-1)}} = \mathbf{s}^{(l-1)}[m]^{\top}, \tag{12}$$

where $\boldsymbol{I}$ denotes the identity matrix and $\top$ the transpose operation. Similar to Shrestha and Orchard (2018), we define $\boldsymbol{e}^{(l)}[n]$ and $\mathbf{d}^{(l)}[n]$ as the derivative of the loss $\mathcal{L}$ with respect to the spike output and the weighted input, respectively, of layer l at time n. Making use of Equations (10, 11) we obtain:

$$\boldsymbol{e}^{(l)}[n] := \frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(l)}[n]} = \sum_{m=n}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l+1)}[m]} \frac{\partial \mathbf{a}^{(l+1)}[m]}{\partial \mathbf{s}^{(l)}[n]} = \mathbf{d}^{(l+1)}[n] \boldsymbol{W}^{(l)} \tag{13}$$

$$\begin{aligned} \mathbf{d}^{(l)}[n] := \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(l)}[n]} &= \sum_{m=n}^{T} \sum_{k=m}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(l)}[k]} \frac{\partial \mathbf{s}^{(l)}[k]}{\partial \mathbf{z}^{(l)}[m]} \frac{\partial \mathbf{z}^{(l)}[m]}{\partial \boldsymbol{a}^{(l)}[n]} \\ &= \sum_{m=n}^{T} \sum_{k=m}^{T} \boldsymbol{e}^{(l)}[k] \boldsymbol{\sigma}_m^{(l)}[k] \epsilon_{m-n}. \end{aligned} \tag{14}$$

These equations are solved for $\boldsymbol{e}^{(l)}[.]$ and $\boldsymbol{d}^{(l)}[.]$ iteratively starting from $\mathbf{e}^{(L)}[.] = \frac{\partial \mathcal{L}}{\partial \mathbf{s}^{(L)}[.]}$ at the last layer. Using Equation (12), we arrive at the gradients of the loss with respect to weight matrix $\boldsymbol{W}^{(l)}$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \sum_{n=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(l+1)}[n]} \frac{\partial \mathbf{a}^{(l+1)}[n]}{\partial \boldsymbol{W}^{(l)}} = \sum_{n=1}^{T} \mathbf{d}^{(l+1)}[n] \cdot \mathbf{s}^{(l)}[n]^{\top} \tag{15}$$

3.3. Simplification for LIF and IF neurons

Our derivation of the gradients in the previous section applies to an arbitrary Spike Response Model (SRM). Since many SNNs are based on the Leaky Integrate-and-Fire (LIF) neuron model, in this section, we derive a more compact expression for this special case. Using the same notation as in Section 2.2, LIF dynamics can be expressed in terms of the SRM (Gerstner, 2021) by choosing spike response and reset kernels $\epsilon_n = \alpha^n \mathbb{1}_{\{n \geq 0\}}$ and $\nu_n = -\alpha^n \theta \mathbb{1}_{\{n \geq 0\}}$, respectively. Here, $\alpha := \exp \frac{-\Delta}{\tau} \in (0, 1)$ is the decay factor in the LIF model determined by the membrane time constant τ and simulation time step Δ. Furthermore, $\mathbb{1}$ denotes the indicator function and θ the firing threshold. This analysis can be applied equivalently to if neurons without leak by setting the decay factor $\alpha \equiv 1$.

The derivatives of $\mathbf{s}^{(l)}[.]$ in Equation (9) can be expressed in closed-form as follows

$$\boldsymbol{\sigma}_m^{(l)}[n] = \begin{cases} 0 & n < m \\ \mathbf{f}'^{(l)}[n] & n = m \\ -\theta \mathbf{f}'^{(l)}[n] \mathbf{f}'^{(l)}[m] \boldsymbol{\chi}_m^{(l)}[n] & n > m \end{cases} \tag{16}$$

$$\boldsymbol{\chi}_m^{(l)}[n] := \begin{cases} \boldsymbol{I} & n = m+1 \\ \prod_{k=m+1}^{n-1} (\alpha \boldsymbol{I} - \theta \mathbf{f}'^{(l)}[k]) & n > m+1 \end{cases} \tag{17}$$

where $\boldsymbol{I}$ denotes the identity matrix. We refer to Appendix 3 for further details.

3.3.1. Computational efficiency

In the following we show how the gradients can be computed efficiently for LIF and IF neurons. We first note that the gradient terms $\boldsymbol{e}^{(l)}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}}$ (Equations 13, 14) are matrix products, for which efficient GPU-accelerated implementations are commonly available. The remaining complexity arises from the terms $\mathbf{d}^{(l)}$ (Equation 14), which can be simplified drastically for the given neuron dynamics. To this end, we introduce an auxiliary variable $\boldsymbol{\zeta}_n^{(l)}[k]$, defined for $n \in [T]$, $l \in [L]$ and $k \geq n$, as

$$\boldsymbol{\zeta}_n^{(l)}[k] := \begin{cases} \boldsymbol{I} & k = n \\ \prod_{m=n}^{k-1} (\alpha \boldsymbol{I} - \theta \mathbf{f}'^{(l)}[m]) & k > n. \end{cases} \tag{18}$$

As demonstrated in Appendix 3.1, for LIF and IF neurons, Equation 14 can be expressed as

$$\mathbf{d}^{(l)}[n] = \sum_{k=n}^{T} e^{(l)}[k] \mathbf{f}'^{(l)}[k] \boldsymbol{\zeta}_n^{(l)}[k], \tag{19}$$

which can be computed in $\mathcal{O}(T)$ time. Furthermore, the vector components of $\mathbf{d}^{(l)}$ are independent of each other, allowing for a highly parallelized, GPU-accelerated gradient computation. From this analysis we expect EXODUS to reach similar computational efficiency as SLAYER (Shrestha and Orchard, 2018). The experimental results in Section 4.5 show that this is indeed the case.

3.4. Comparison with SLAYER gradients

By comparing the gradients computed in Section 3 with those in SLAYER (Shrestha and Orchard, 2018), we find that the expressions for $e^{(l)}[.]$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}}$ (see Equations 13, 15) match. The difference lies mainly in $\mathbf{d}^{(l)}$ (Equation 14) and more specifically in the derivatives $\frac{\partial \mathbf{s}^{(l)}[n]}{\partial \mathbf{z}^{(l)}[m]}$, which are set to 0 for $m \neq n$ in SLAYER. In particular, we would obtain the gradients in SLAYER by setting the reset kernel to 0, in which case $\boldsymbol{\sigma}_m^{(l)}[n]$ is only nonzero for $m = n$. In the concrete case of LIF neuron dynamics (cf. Section 3.3), we may indeed notice that the quality of the approximation of the gradients in SLAYER depends on how the matrix $\boldsymbol{\chi}_m^{(l)}$ (see Equation 17) decays as a function of index-difference $|n - m - 1|$. This decay, in general, can be characterized in terms of singular values of $\mathbf{f}'^{(l)}[n]$. However, since the matrix of surrogate gradients $\mathbf{f}'^{(l)}[n]$ are all diagonal, this boils down to how the diagonal elements

$$\left(\boldsymbol{\chi}_m^{(l)}[n]\right)_{i,i} = \prod_{k=m+1}^{n} (\alpha - \theta f_s'(u_i^{(l)}[k]))$$

decay as a function of $n - m - 1$ (for $n \geq m + 1$). We may study several interesting scenarios:

(i) In the usual case where f_s is an increasing function, $f_s'(u)$ is positive for all $u \in [0, \theta]$. If in addition, one designs the surrogate gradient f_s' such that $f_s'(u) \in [0, \frac{\alpha - \mu}{\theta}]$ for some $\mu \in [0, \alpha]$ for all $u \in [0, \theta]$, one may obtain the bounds $0 \leq \left(\boldsymbol{\chi}_m^{(l)}[n]\right)_{i,i} \leq \mu^{n-m-1}$, which implies that $\left(\boldsymbol{\chi}_m^{(l)}[n]\right)_{i,i}$ is quite small if $n \geq m + 1 + \mathcal{O}(\log \frac{1}{\mu})$. As a result, compared with SLAYER, which assumes $\boldsymbol{\chi}_m^{(l)}[n] \equiv 0$ for all m, n, l, our method takes into account additional $\mathcal{O}(\log \frac{1}{\mu})$ correction terms.

(ii) If the surrogate gradient is not designed properly such that it is larger than $\frac{\alpha}{\theta}$ for some range of u within $[0, \theta]$, the term $\boldsymbol{\chi}_m^{(l)}[n]$ may become significantly large as $n \gg m + 1$. In such a case, we may expect a large deviation between the corrected gradients derived from our method and the approximate gradients in

TABLE 1 Validation accuracy and backward pass speedup per epoch for different datasets.

	Validation accuracy [%]		Mean backward pass speedup compared to BPTT	
	EXODUS	SLAYER	EXODUS	SLAYER
CIFAR10-DVS	**72.28 ± 0.13**	71.53 ± 0.18	–	–
DVS gesture (5 layers)	**92.8 ± 2.2**	87.8 ± 3.0	**16.57×**	14.09×
DVS gesture (8 layers)	**94.54 ± 0.8**	93.64 ± 0.49	–	–
SHD	**78.01 ± 0.2**	70.58 ± 1.9	9.1×	**11.22×**
SSC	**55.41 ± 0.4**	40.1 ± 0.8	**17.3×**	14.59×
Average for 3 datasets			**14.3×**	13.3×

Validation accuracy is mean and standard deviation of maximum accuracies across three runs with a gradient scale of 1. Backward pass speedup is measured across three epochs on a NVIDIA GeForce 1,080 Ti. Bold indicate best value for each category (accuracy, speedup) and dataset.

SLAYER. We speculate that this may be the reason SLAYER gradients are more sensitive to the scaling of the surrogate gradients, as shown in our results.

4. Simulation results

We show that EXODUS leads to faster convergence than SLAYER when applied to different tasks, as our method computes the same gradients as BPTT. We benchmark both algorithms on four neuromorphic tasks with increasingly temporal features. To compare numerical stability, experiments are repeated over different gradient scaling factors, incorporated in the surrogate gradients f_s' (see Section 2.2). It is worthwhile to mention that in our experiments we do not aim to illustrate superior classification performance of our algorithm compared with other state-of-the-art algorithms, but the achievable performance improvement over SLAYER. We make a conscious decision not to use any training tricks that improve accuracy (learning rate schedulers, data augmentation, etc.) as this would harm a direct comparison. In brief, any results in literature that have been obtained by using BPTT can be recreated using EXODUS if the neuron model supports it, because we compute the same gradients (see Section 5 in the Appendix for a numerical demonstration). When comparing EXODUS and SLAYER, we make sure that forward pass neuron dynamics, random seed and initial weights are the same. We used SLAYER's official PyTorch implementation available on Github. Detailed architecture and training parameters are described in the Table 1 in Appendix.

4.1. CIFAR10-DVS

This dataset is a neuromorphic version of the original image classification dataset (Li et al., 2017). For this experiment we used a spiking ResNet Wide-7B architecture with 1.19M parameters (Fang et al., 2021). The best previously published result using that architecture achieves 70.2% accuracy for 8 time steps and LIF neurons, which we beat with both EXODUS and SLAYER (see Table 1). EXODUS achieves the highest accuracy overall at 72.28%.

4.2. DVS Gesture

We also test the common classification dataset of 11 different hand and arm gestures (Amir et al., 2017). Similar to Shrestha and Orchard (2018), we only use the first 1.5 s of each sample, binned to 300 time steps. To classify the gestures, we test two different spiking convolutional architectures. The smaller version has four convolutional layers as a feature extractor, followed by a fully connected layer. The larger architecture uses seven convolutional layers as feature extractor. Both versions use Integrate-and-Fire neurons, Adam and sum-over-time cross entropy loss. The left-most columns in Figure 2 and Table 1 show validation accuracies over the course of 100 training epochs for the 5-layer architecture. EXODUS reaches a higher accuracy at 92.8% than SLAYER at 87.8%. For context, the state-of-the-art result using an optimized architecture and different data pre-processing reports 98% accuracy (She et al., 2021). The center column in Figure 2 shows how weight gradients for individual layers scale under different surrogate gradient scales. SLAYER is much more sensitive to the scale, with gradients often taking excessive values for earlier layers.

4.3. Spiking Heidelberg Digits

This dataset is an event-based audio classification dataset with highly temporal features, with samples recorded from a silicon cochlea (Cramer et al., 2020). We used a 4-layer fully-connected architecture with Integrate-and-Fire neurons and max-over-time loss for 250 time steps. As shown in Figure 2 and Table 1, training converges faster and reaches a significantly higher accuracy using EXODUS at 78.01% compared to SLAYER at 70.58%. Gradients are much more stable for earlier layers in EXODUS. The highest accuracy result reported in the literature for this task is 83.2% using a recurrent architecture (Cramer et al., 2020).

4.4. Spiking speech commands

For this task we used a neuromorphic version of Google's Speech Command dataset (Cramer et al., 2020). We use the same 4-layer fully-connected architecture with Integrate-and-Fire neurons and 250 time steps as for the SHD task. Here once again our simulation results in Figure 2 and Table 1 show that

10.3389/fnins.2023.1110444

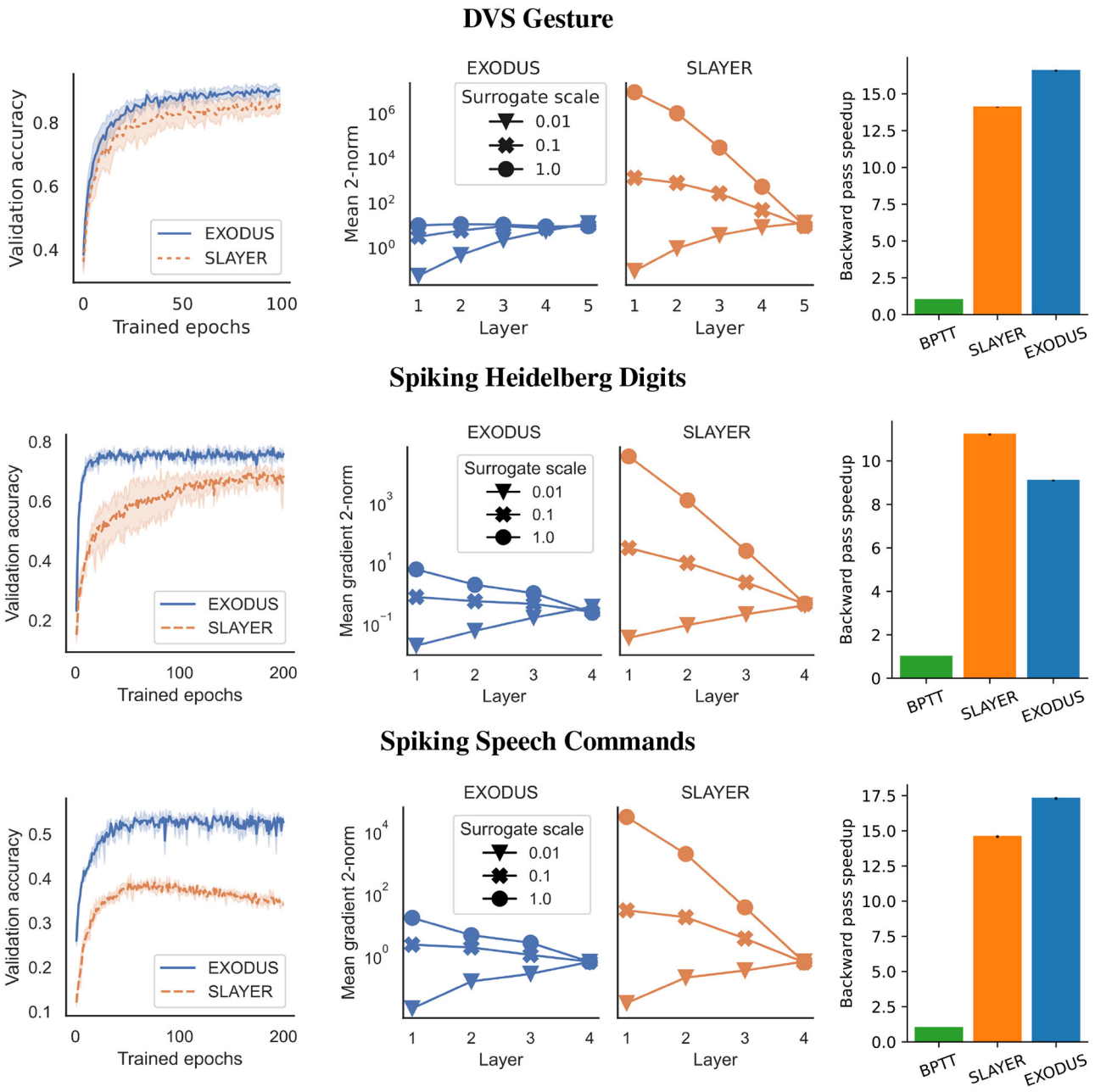

FIGURE 2
Performance on DVS Gesture **(Top row)** with 5-layer architecture, Spiking Heidelberg Digits **(Center row)** and Spiking Speech Commands **(Bottom row)** datasets. **(Left column)** Validation accuracy averaged over three runs for each algorithm using a gradient scale of 1. EXODUS (solid blue curves) generally converges much faster and reaches a higher accuracy than SLAYER (dashed orange curves), in particular for tasks with strong temporal dependencies. **(Center column)** Mean two-norms of weight gradients during training, for individual layers and different scaling of surrogate gradient. When surrogate gradients are not down-scaled, gradients for SLAYER explode toward the input layer, whereas for EXODUS they remain mostly stable. Only for very low scaling, gradients vanish for both algorithms. **(Right column)** Backward pass speedup relative to BPTT. EXODUS is the fastest on average across the three datasets.

gradients across layers in EXODUS are much more stable than when using SLAYER. Validation accuracy is significantly higher in this task using EXODUS with 55.4% vs 40.1% for SLAYER averaged across 3 runs. The highest accuracy result reported in the literature is 50.9% using a recurrent architecture (Cramer et al., 2020).

4.5. Training speedup

The training speedup in comparison to an unoptimized implementation of BPTT is shown in the right-most column in Figure 2 and Table 1. It shows relative speedups of SLAYER and EXODUS backward passes for three datasets, averaged over three

runs each. We focus on the backward pass as forward passes are mathematically equivalent for all three methods. Different computation times in the forward passes are possible due to the individual implementations of the algorithms. All speed tests are executed on a NVIDIA GeForce 1,080 Ti. Our BPTT algorithm is implemented in PyTorch 1.11 and makes use of the library's native automatic differentiation functionality and GPU backend, but otherwise does not contain any custom CUDA code, which makes it much more flexible. Across three datasets, SLAYER's backward pass is 13.3 times faster than BPTT on average, whereas EXODUS is 14.3 times faster. This shows that both algorithms achieve significant speedup in comparison to a non-optimized implementation and that taking into account the reset kernel in the backward pass in EXODUS does not hurt performance at all. The minor difference between SLAYER and EXODUS can be attributed to small differences in the CUDA code implementation. Absolute backward pass timings are provided in the Table 2 in Appendix.

5. Discussion

We have shown that while SLAYER enables training of spiking neural networks at high computational efficiency, it does so at the cost of omitting the effect of the neuron's reset mechanism on the gradients. With our newly proposed algorithm EXODUS, we present a modification of SLAYER that computes the same gradients as those in the original BPTT while maintaining the high computational efficiency of SLAYER.

Tuning the surrogate gradient function is important when training deeper architectures (Ledinauskas et al., 2020), which can be a costly manual process when using SLAYER. EXODUS makes training deep SNN architectures from scratch easier by providing less sensitivity to the gradient scale hyperparameter. Not only does the gradient magnitude scale in a stable manner, but also the gradient direction leads to faster convergence as shown in our experiments. To demonstrate this, we picked tasks that require both spatial and temporal depth in the computational graph.

The difference between EXODUS and SLAYER is especially noticeable for long time constants and Integrate-and-Fire neurons (which do not have any leak) in the extreme case, but also persists for shorter time constants (see Appendix 6). We argue that because of the neuron's longer memory the contribution of the neuron's reset mechanism increases, which is not taken into account in SLAYER's gradient computation.

For Integrate-and-Fire neurons, both with or without leak, the terms that ensure correct representation of the reset mechanism in the gradients with EXODUS allow for a computationally efficient implementation, resulting in similar computation speeds as SLAYER. It is possible that other GPU-accelerated spiking neural network simulators, such as Norse (Pehle and Pedersen, 2021) or Spiking Jelly (Fang et al., 2020) achieve higher computational speeds than our baseline BPTT implementation. However, due to incompatibilities in the supported neuron models, a direct comparison was not possible.

Similar to Shrestha and Orchard (2018), EXODUS works exclusively with feedforward network architectures. In recurrent architectures, dynamics of individual neurons are coupled more strongly, which complicates a parallelized implementation. The application of EXODUS to such types of SNNs might be an interesting topic for future research. We hope that our method of deriving gradients through the ift is of independent interest for devising new learning strategies in a rigorous manner when no explicit functional relation exists between two or more variables.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

FB and SH developed the algorithm described in this work. GL and FB conducted and analyzed the computer simulations. All authors wrote sections of the manuscript, contributed to manuscript revision, read, and approved the submitted version.

Funding

This work was partially funded by the ECSEL Joint Undertaking (JU) under grant agreements number 876925, "ANDANTE" and number 826655, "TEMPO". The JU receives support from the European Union's Horizon 2020 research and innovation program and France, Belgium, Germany, Netherlands, Portugal, Spain, Switzerland.

Conflict of interest

FB, GL, SH, and SS were employed by SynSense AG.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2023.1110444/full#supplementary-material

References

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7243–7252. doi: 10.1109/CVPR.2017.781

Bai, S., Kolter, J. Z., and Koltun, V. (2019). Deep equilibrium models. *Adv. Neural Inform. Process. Syst*. 32.

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. (2018). Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Adv. Neural Inform. Process. Syst*. 31.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning -to-learn in networks of spiking neurons. *Adv. Neural Inform. Process. Syst*. 31.

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun*. 11, 3625. doi: 10.1038/s41467-020-17236-y

Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., et al. (2021). Efficient and modular implicit differentiation. *arXiv Preprint*. arXiv:2105.15183.

Bohte, S. M. (2011). "Error-backpropagation in networks of fractionally predictive spiking neurons," in *International Conference on Artificial Neural Networks* (Springer), 60–68.

Bohte, S. M., Kok, J. N., and La Poutre, J. A. (2000). "Spikeprop : backpropagation for networks of spiking neurons," in *ESANN, Vol. 48* (Bruges), 419–424.

Burkitt, A. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybernet*. 95, 1–19. doi: 10.1007/s00422-006-0068-6

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Choe, Y. (2013). *Hebbian Learning*. New York, NY: Springer.

Comsa, I.-M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. (2021). "Temporal coding in spiking neural networks with alpha synaptic function: learning with backpropagation," in *IEEE Transactions on Neural Networks and Learning Systems* (IEEE), p. 5939–5952.

Cramer, B., Stradmann, Y., Schemmel, J., and Zenke, F. (2020). "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks," in *IEEE Transactions on Neural Networks and Learning Systems* (IEEE), p. 2744–2757.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., et al. (2021). Advancing neuromorphic computing with loihi: a survey of results and outlook. *Proc. IEEE* 109, 911–934. doi: 10.1109/JPROC.2021.3067593

Diehl, P., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9, 99. doi: 10.3389/fncom.2015.00099

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A*. 113, 11441–11446. doi: 10.1073/pnas.1604850113

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. (2021). Deep residual learning in spiking neural networks. *Adv. Neural Inform. Process. Syst*. 34, 21056–21069.

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed January 4, 2023).

Gerstner, W. (1995). Time structure of the activity in neural network models. *Phys. Rev. E* 51, 738–758. doi: 10.1103/PhysRevE.51.738

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., et al. (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nat. Mach. Intell*. 3, 823–835. doi: 10.1038/s42256-021-00388-x

Knight, J. C., Komissarov, A., and Nowotny, T. (2021). Pygenn: a python library for gpu-enhanced neural networks. *Front. Neuroinform*. 15, 659005. doi: 10.3389/fninf.2021.659005

Ledinauskas, E., Ruseckas, J., Juršenas, A., and Buračas, G. (2020). Training deep spiking neural networks. *arXiv [Preprint] arXiv* 2006.04436.

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci*. 12, 435. doi: 10.3389/fnins.2018.00435

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci*. 11, 309. doi: 10.3389/fnins.2017.00309

Lobov, S. A., Mikhaylov, A. N., Shamshin, M., Makarov, V. A., and Kazantsev, V. B. (2020). Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot. *Front. Neurosci*. 14, 88. doi: 10.3389/fnins.2020.00088

Martín-Sánchez, G., Bohté, S., and Otte, S. (2022). A taxonomy of recurrent learning rules. *arXiv Preprint* arXiv:2207.11439.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst*. 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). 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. doi: 10.1109/MSP.2019.2931595

Nowotny, T., Turner, J. P., and Knight, J. C. (2022). Loss shaping enhances exact gradient learning with eventprop in spiking neural networks. *arXiv Preprint*. arXiv:2212.01232.

Panda, P., Aketi, A., and Roy, K. (2020). Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Front. Neurosci*. 14, 653. doi: 10.3389/fnins.2020.00653

Pehle, C., and Pedersen, J. E. (2021). *Norse–A Deep Learning Library for Spiking Neural Networks*. Available online at: https://norse.ai/docs/ (accessed January 04, 2023).

Perez-Nieves, N., and Goodman, D. (2021). Sparse spiking gradient descent. *Adv. Neural Inform. Process. Syst*. 34, 11795–11808.

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

Safa, A., Catthoor, F., and Gielen, G. G. (2021). Convsnn: a surrogate gradient spiking neural framework for radar gesture recognition. *Softw. Impacts* 10, 100131. doi: 10.1016/j.simpa.2021.100131

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci*. 11, 24. doi: 10.3389/fncom.2017.00024

She, X., Dash, S., and Mukhopadhyay, S. (2021). "Sequence approximation using feedforward spiking neural network for spatiotemporal learning: theory and optimization methods," in *International Conference on Learning Representations*.

Shrestha, S. B., and Orchard, G. (2018). Slayer: spike layer error reassignment in time. *Adv. Neural Inform. Process. Syst*. 31.

Wozniak, S., Pantazi, A., Bohnstingl, T., and Eleftheriou, E. (2020). Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nat. Mach. Intell.* 2, 325–336. doi: 10.1038/s42256-020-0187-0

Wunderlich, T. C., and Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Sci. Rep*. 11, 12829. doi: 10.1038/s41598-021-91786-z

Zenke, F., and Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput*. 30, 1514–1541. doi: 10.1162/neco_a_01086