ARTICLE Communicated by Mina Aliakabari Khoei

Exploring Trade-Offs in Spiking Neural Networks

Florian Bacho
fb320@kent.ac.uk
Dominique Chu
d.f.chu@kent.ac.uk
CEMS, School of Computing, University of Kent, Canterbury CT2 7NF, U.K.

Spiking neural networks (SNNs) have emerged as a promising alternative to traditional deep neural networks for low-power computing. However, the effectiveness of SNNs is not solely determined by their performance but also by their energy consumption, prediction speed, and robustness to noise. The recent method Fast & Deep, along with others, achieves fast and energy-efficient computation by constraining neurons to fire at most once. Known as time-to-first-spike (TTFS), this constraint, however, restricts the capabilities of SNNs in many aspects. In this work, we explore the relationships of performance, energy consumption, speed, and stability when using this constraint. More precisely, we highlight the existence of trade-offs where performance and robustness are gained at the cost of sparsity and prediction latency. To improve these trade-offs, we propose a relaxed version of Fast & Deep that allows for multiple spikes per neuron. Our experiments show that relaxing the spike constraint provides higher performance while also benefiting from faster convergence, similar sparsity, comparable prediction latency, and better robustness to noise compared to TTFS SNNs. By highlighting the limitations of TTFS and demonstrating the advantages of unconstrained SNNs, we provide valuable insight for the development of effective learning strategies for neuromorphic computing.

1 Introduction

Over the past decade, deep neural networks (DNNs) have become indispensable tools in statistical machine learning, achieving state-of-the-art performance in various applications, including computer vision (Krizhevsky et al., 2012; Szegedy et al., 2013), natural language processing (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020), and reinforcement learning (Mnih et al., 2013, 2016). However, their impressive performance often comes at a significant hardware and energy cost. For example, natural language processing models can consist of billions of parameters and require energy-intensive GPU clusters to train efficiently (Brown et al., 2020). These hardware and energy requirements pose a significant challenge in

Neural Computation **35, 1627–1656** (2023)
https://doi.org/10.1162/neco_a_01609

terms of sustainability and restrict the practical applicability of DNNs in resource-limited environments such as low-powered edge devices. Therefore, exploring more energy-efficient alternatives to DNNs is crucial not only to address the environmental cost of machine learning but also to provide practical and sustainable solutions in edge computating.

One possible alternative to DNNs is spiking neural networks (SNNs). Spiking neurons process information through discrete spatiotemporal events known as *spikes* rather than continuous real-number values (Maass, 1997; Gerstner & Kistler, 2002). Spikes enable efficient implementations of neural networks on non-von Neumann neuromorphic hardware such as Intel Loihi, IBM TrueNorth, Brainscale2, and SpiNNaker (Painkras et al., 2013; Akopyan et al., 2015; Schmitt et al., 2017; Davies et al., 2018; Furber, 2016; Hendy & Merkel, 2022). Such hardware consumes only a fraction of the power required by DNNs on von Neumann computers and thus represents a suitable solution for energy-efficient edge computing (Blouw et al., 2019; Taunyazov et al., 2020).

The power consumption of neuromorphic hardware is closely related to the number of spikes they produce. Therefore, sparse SNNs that fire only a small number of spikes achieve high energy efficiency on hardware. However, such networks also transmit less information, creating a trade-off between energy consumption and model accuracy. In addition to sparsity, various trade-offs between performance and other aspects of SNNs are commonly explored in the literature (Park et al., 2021; Yin et al., 2023; Li et al., 2021; Diehl et al., 2015).

For instance, unsupervised learning rules such as spike time dependent plasticity (STDP) can be implemented directly on neuromorphic hardware, allowing for biologically plausible and energy-efficient training of SNNs that are resilient to substrate noise of analog circuits (Kim et al., 2020). However, the performance of unsupervised learning lags behind that achieved with supervised learning. Meanwhile, state-of-the-art performance with SNNs is currently achieved through various error backpropagation (BP) techniques adapted from deep learning (Bohté et al., 2000; Lee et al., 2016; Jin et al., 2018; Kheradpisheh & Masquelier, 2020; Kheradpisheh et al., 2021; Zhang et al., 2022; Shrestha & Orchard, 2018; Wu et al., 2018; Mostafa, 2016; Comsa et al., 2020; Göltz et al., 2021; Fang et al., 2021). However, BP algorithms often require constraints on spikes to achieve high sparsity and low prediction latency, at the cost of performance (Yan et al., 2022; Mostafa, 2016; Guo et al., 2020; Kheradpisheh & Masquelier, 2020; Kheradpisheh et al., 2021; Göltz et al., 2021). In addition, BP requires global transport of information that is incompatible with neuromorphic hardware, and training must be performed either offline or in-the-loop, where a conventional computer is used in conjunction with neuromorphic hardware (Schmitt et al., 2017; Göltz et al., 2021). Therefore, SNNs trained offline or in the loop must also be resilient to the substrate noise and weight quantization of analog hardware to avoid performance drops at deployment.

One particularly interesting approach for training fast, energy-efficient, and noise-resilient SNNs is Fast & Deep (Göltz et al., 2021). This exact BP method employs time-to-first-spike (TTFS) coding, which restricts neurons to fire only once. Inspired by the human visual system (Thorpe et al., 1996), TTFS is based on the idea that first spikes of neurons must carry most of the information about input stimuli, enabling fast, sparse, and energy-efficient computation. Due to this constraint imposed on firing, we thus referred to TTFS networks as *constrained* SNNs. However, relaxing the spike constraint of TTFS and allowing multiple spikes per neuron typically results in higher information rates, better performance, and increased noise resilience compared to TTFS networks (Jin et al., 2018; Zhang & Li, 2019; Shrestha & Orchard, 2018; Lee et al., 2016; Zhang et al., 2022). One might assume that such network, referred to as *unconstrained* SNNs, would improve performance and noise robustness but also result in slower inference and lower energy efficiency due to increased firing rates.

In this work, we explore the trade-offs among performance, convergence, energy consumption, prediction speed, and robustness of SNNs, with and without the spike constraint imposed by TTFS. We make the following main contributions:

- We demonstrate that many aspects are driven by the weight distribution in Fast & Deep, highlighting trade-offs among performance, energy consumption, latency, and stability in TTFS SNNs.
- We extend the Fast & Deep algorithm to multiple spikes per neuron and describe how errors are backpropagated in unconstrained SNNs.
- We show that our proposed method improves performance while providing better convergence rate, similar sparsity, comparable latency, and improved robustness to noise compared to Fast & Deep, suggesting that relaxing the spike constraints in TTFS can lead to better trade-offs.

The rest of the article is structured as follows. In section 2, we derive the closed-form solution for spike timings as well as their gradients, as described by Fast & Deep (Göltz et al., 2021), and describe how errors are backpropagated in unconstrained SNNs. In section 3, we show the results of our experiments on performance, convergence, sparsity, prediction latency, and robustness to noise and weight quantization of Fast & Deep and our proposed method.

2 Method

In this section, we describe our generalization of the Fast & Deep algorithm to multiple spikes per neuron. Our main contribution lies in the reset of the membrane potential and how errors are backpropagated through interneuron and intraneuron dependencies.

2.1 The CuBa LIF Neuron. We consider a neural network of current-based leaky integrate-and-fire neurons with a soft reset of the membrane potential. (Gerstner & Kistler, 2002; Davies et al., 2018; Göltz et al., 2021).

Formally, the dynamic of the membrane potential $u^{(l,j)}$ of the jth neuron in layer l is given by the system of linear ordinary differential equations:

$$\begin{aligned}\frac{du^{(l,j)}}{dt} &= -\frac{1}{\tau}u^{(l,j)}(t) + g^{(l,j)}(t) - \underbrace{\vartheta\delta\left(u^{(l,j)}(t) - \vartheta\right)}_{\text{Reset}},\\ \frac{dg^{(l,j)}}{dt} &= -\frac{1}{\tau_s}g^{(l,j)}(t) + \underbrace{\sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \delta(t - t_z^{(l-1,i)})}_{\text{Pre-synaptic spikes}},\end{aligned} \tag{2.1}$$

where

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

is the Dirac delta function that satisfies the identity $\int_{-\infty}^{+\infty} \delta(x)dx = 1$. We denote the number of neurons in the layer l as $N^{(l)}$, the number of spikes fired by the neuron j of this layer as $n^{(l,j)}$, and the kth spike of this neuron as $t_k^{(l,j)}$. Each presynaptic spike received at a synapse i of a neuron j induces an increase in postsynaptic current $g^{(l,j)}$ by an amount $w_{i,j}^{(l)}$, which defines the strength of the synaptic connection. The postsynaptic current $g^{(l,j)}$ is then integrated into the membrane potential $u^{(l,j)}$. We denote by τ and τ_s the membrane and synaptic time constants that control the decay of the membrane potential and the postsynaptic current, respectively. Finally, when the membrane potential reaches the threshold ϑ, the neuron emits a postsynaptic spike at time $t_k^{(l,j)}$ where k is the index of the emitted spike and $u^{(l,j)}$ is reset to zero due to an instantaneous negative current of the size of the threshold.

The reset of the membrane potential in equation 2.1 is the major difference with the TTFS model used in Fast & Deep (Göltz et al., 2021). By resetting the membrane potential after postsynaptic spikes, our model allows for further integration of inputs and thus the firing of several spikes. Therefore, this relaxes the constraint on spike counts imposed by Fast & Deep.

2.2 SRM Mapping. The spike response model (SRM) is a generalization of the LIF neuron where the subthreshold behavior of the neuron is defined by an integral over the past (Gerstner & Kistler, 2002; Göltz et al., 2021; Wunderlich & Pehle, 2021). This form is more convenient to derive

as it is an explicit function of time. Formally, the SRM neuron defines the membrane potential as

$$u^{(l,j)}(t) = \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \epsilon\left(t - t_z^{(l-1,i)}\right) - \sum_{z=1}^{n^{(l,j)}} \eta\left(t - t_z^{(l,j)}\right), \tag{2.3}$$

where $\epsilon(t)$ is the post-synaptic potential (PSP) kernel that represents the response of the neuron to a presynaptic spike and $\eta(t)$ is the refractory kernel that defines the reset behavior. In Fast & Deep (Göltz et al., 2021), the refractory kernel $\eta(t) = 0$ is zero as no reset of the membrane potential is required.

Using the definition of the CuBa LIF (see equation 2.1), we find by integration the following PSP and refractory kernels (Gerstner & Kistler, 2002):

$$\begin{aligned}\epsilon(t) &= \Theta(t)\frac{\tau\tau_s}{\tau - \tau_s}\left[\exp\left(\frac{-t}{\tau}\right) - \exp\left(\frac{-t}{\tau_s}\right)\right],\\ \eta(t) &= \Theta(t)\vartheta\exp\left(\frac{-t}{\tau}\right)\end{aligned} \tag{2.4}$$

where

$$\Theta(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

is the Heavyside step function.

2.3 Closed-Form Solution of Spike Timing. Let us consider the kth spike timing $t_k^{(l,j)}$ of the neuron j in layer l. Fast & Deep (Göltz et al., 2021) described that by constraining the membrane time constant as twice the synaptic time constant ($\tau = 2\tau_s$), the SRM mapping of the LIF neuron can be written as a polynomial of degree 2, such as

$$0 = -a_k^{(l,j)}\exp\left(\frac{-t_k^{(l,j)}}{\tau}\right)^2 + b_k^{(l,j)}\exp\left(\frac{-t_k^{(l,j)}}{\tau}\right) - c_k^{(l,j)}, \tag{2.6}$$

where $a_k^{(l,j)}$, $b_k^{(l,j)}$, and $c_k^{(l,j)}$ are three coefficients that depend on the definition of the model. Equation 2.6 can therefore be solved for $t_k^{(l,j)}$ by using the quadratic formula:

$$t_k^{(l,j)} = \tau \ln\left[\frac{2a_k^{(l,j)}}{b_k^{(l,j)} + x_k^{(l,j)}}\right] \tag{2.7}$$

with $x_k^{(l,j)} = \sqrt{\left(b_k^{(l,j)}\right)^2 - 4a_k^{(l,j)}c_k^{(l,j)}}$.

This equation can be thus used to infer the spike trains of neurons in an event-based manner.

According to the definition of our model, we find by factorizing equation 2.3 into equation 2.6, we derive the following coefficients:

$$a_k^{(l,j)} := \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right) \exp\left(\frac{t_z^{(l-1,i)}}{\tau_s}\right), \tag{2.8}$$

$$b_k^{(l,j)} := \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right) \exp\left(\frac{t_z^{(l-1,i)}}{\tau}\right) - \frac{\vartheta}{\tau} \sum_{z=1}^{n^{(l,j)}} \Theta\left(t_k^{(l,j)} - t_z^{(l,j)}\right) \exp\left(\frac{t_z^{(l,j)}}{\tau}\right) \tag{2.9}$$

and

$$c := c_k^{(l,j)} = \frac{\vartheta}{\tau}. \tag{2.10}$$

Note that the value of $c_k^{(l,j)}$ is common to every spike emitted by the neuron. We thus denote its value as c for short. Moreover, only the definition of the coefficient $b_k^{(l,j)}$ differs from Fast & Deep (Göltz et al., 2021) due to the reset of the membrane potential. For comparison, $b_k^{(l,j)}$ is defined as follows in Fast & Deep (Göltz et al., 2021):

$$b_k^{(l,j)} := \sum_{i=1}^{N^{(l-1)}} w_{i,j}^{(l)} \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right) \exp\left(\frac{t_z^{(l-1,i)}}{\tau}\right). \tag{2.11}$$

2.4 Spike Count Loss Function. For each neuron j in the output layer o, we define a spike count target y_j. The aim is to minimize the distance between the actual output spike counts and their corresponding targets. Therefore, we define the loss function as

$$\mathcal{L} := \frac{1}{2} \sum_{j=1}^{N^{(o)}} \left(y_j - n^{(o,j)}\right)^2, \tag{2.12}$$

where o is the index of the output layer and y_j is the spike count target associated with the same neuron.

2.5 Gradient of Unconstrained Neurons. Because the spike timing now has a closed-form solution, it becomes differentiable, which allows the

computation of an exact gradient. We first state the total change of weight between two neurons.

Let $\delta_k^{(l,j)}$ be the error received by the spike k of the neuron j in layer l; the change of weight $\Delta w_{i,j}^{(l)}$ between the presynaptic neuron i and the postsynaptic neuron j of the layer l is defined as a sum of all errors applied to their corresponding spike derivatives,

$$\begin{aligned}\Delta w_{i,j}^{(l)} &= \sum_{k=1}^{n^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} \\ &= \sum_{k=1}^{n^{(l,j)}} \delta_k^{(l,j)} \frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}},\end{aligned} \tag{2.13}$$

where

$$\begin{aligned}\frac{\partial t_k^{(l,j)}}{\partial w_{i,j}^{(l)}} = \sum_{z=1}^{n^{(l-1,i)}} \Theta\left(t_k^{(l,j)} - t_z^{(l-1,i)}\right) \Bigg[f_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau_s}\right) \\ - \; h_k^{(l,j)} \exp\left(\frac{t_z^{(l-1,i)}}{\tau}\right)\Bigg]\end{aligned} \tag{2.14}$$

is the partial derivative of equation 2.7 with respect to the weight $w_{i,j}^{(l)}$ and

$$\begin{aligned}f_k^{(l,j)} &:= \frac{\partial t_k^{(l,j)}}{\partial a_k^{(l,j)}} = \frac{\tau}{a_k^{(l,j)}} \left[1 + \frac{c}{x_k^{(l,j)}} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right)\right], \\ h_k^{(l,j)} &:= \frac{\partial t_k^{(l,j)}}{\partial b_k^{(l,j)}} = \frac{\tau}{x_k^{(l,j)}}.\end{aligned} \tag{2.15}$$

Therefore, the weight $w_{i,j}^{(l)}$ can be updated using the gradient descent algorithm as

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \lambda \Delta w_{i,j}^{(l)}, \tag{2.16}$$

where λ is a learning rate parameter.

2.6 Spike Errors. We now derive the spike error $\delta_k^{(l,j)}$ associated with the spike at time $t_k^{(l,j)}$. In unconstrained SNNs, errors are backpropagated through two distincts paths: from postsynaptic to presynaptic spikes (i.e., interneuron dependencies) due to the synaptic connections and through postsynaptic to postsynaptic spikes (intraneuron dependencies) due to the

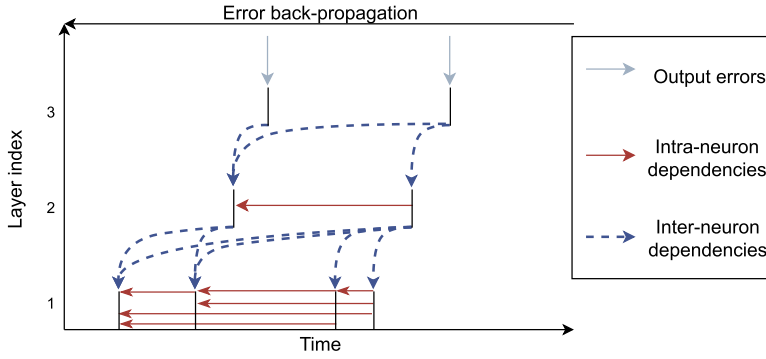


Figure 1: Illustration of error backpropagation through spikes. This figure represents a three-layered network where the spike trains of only one neuron per layer are shown. The gray arrows represent the error coming from the loss function, the dashed blue arrows are the errors backpropagated from the downstream spikes (i.e., interneuron dependencies), and the red arrows are the error backpropagated from the future activity of the neuron due to the recurrence of the reset function (i.e., intraneuron dependencies).

reset of the membrane potential after firing. In contrast, Fast & Deep (Göltz et al., 2021) propagates errors only through interneuron dependencies as the membrane potential of constrained neurons is never reset. Figure 1 provides a visual illustration of these dependencies. Therefore, we decompose the error received by the spike $t_k^{(l,j)}$ into two error components, such as

$$\begin{aligned}\delta_k^{(l,j)} &:= \frac{\partial \mathcal{L}}{\partial t_k^{(l,j)}} \\ &= \underbrace{\sum_{i=1}^{N^{(l+1)}} \sum_{z=1}^{n^{(l+1,i)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l+1,i)}} \frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}}}_{\text{Inter-Neuron}} + \underbrace{\sum_{z=k+1}^{n^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l,j)}} \frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}}}_{\text{Intra-Neuron}} \\ &= \phi_k^{((l,j)} + \mu_k^{(l,j)}, \end{aligned} \tag{2.17}$$

where $\phi_k^{((l,j)}$ is the error backpropagated from interneuron dependencies and $\mu_k^{(l,j)}$ is the error backpropagated from intraneuron dependencies.

The interneuron error $\phi_k^{((l,j)}$ has different definitions for output neurons and hidden neurons. For output neurons, the interneuron error $\phi_k^{((o,j)}$ received by the kth spike of the neuron j is defined as the derivative of the spike count loss function as no interneuron backpropagation is required:

$$\phi_k^{(o,j)} := \frac{\partial \mathcal{L}}{\partial n^{(o,j)}} = y_j - n^{(o,j)}. \tag{2.18}$$

For hidden neurons, the interneuron error $\phi_k^{(l,j)}$ is defined as the sum of all errors backpropagated from the downstream spikes that have been emitted since $t_k^{(l,j)}$, such as

$$\begin{aligned}\phi_k^{(l,j)} &:= \sum_{i=1}^{N^{(l+1)}} \sum_{z=1}^{n^{(l+1,i)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l+1,i)}} \frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}} \\ &= \sum_{i=1}^{N^{(l+1)}} \sum_{z=1}^{n^{(l+1,i)}} \delta_z^{(l+1,i)} \frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}},\end{aligned} \tag{2.19}$$

where

$$\begin{aligned}\frac{\partial t_z^{(l+1,i)}}{\partial t_k^{(l,j)}} = \Theta\left(t_z^{(l+1,i)} - t_k^{(l,j)}\right) w_{j,i}^{(l+1)} \Bigg[\frac{f_z^{(l+1,i)}}{\tau_s} \exp\left(\frac{t_k^{(l,j)}}{\tau_s}\right) \\ - \frac{h_z^{(l+1,i)}}{\tau} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right)\Bigg].\end{aligned} \tag{2.20}$$

For the intraneuron error $\mu_k^{(l,j)}$, all the errors backpropagated from the future spike activity of the neuron must be taken into account due to the temporal impact of the resets on the future membrane potential. Therefore, $\mu_k^{(l,j)}$ is defined as a sum over all errors backpropagated from the following postsynaptic spikes in time:

$$\begin{aligned}\mu_k^{(l,j)} &:= \sum_{z=k+1}^{n^{(l,j)}} \frac{\partial \mathcal{L}}{\partial t_z^{(l,j)}} \frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}} \\ &= \sum_{z=k+1}^{n^{(l,j)}} \delta_z^{(l,j)} \frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}},\end{aligned} \tag{2.21}$$

where

$$\frac{\partial t_z^{(l,j)}}{\partial t_k^{(l,j)}} = \frac{\vartheta}{\tau x_z^{(l,j)}} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right). \tag{2.22}$$

If evaluated as written, equation 2.21 implies a quadratic time complexity $\mathcal{O}\left(n^2\right)$ (where n is the total number of spikes emitted by the neurons) due to the recurrence. However, every reset has the same impact on the membrane potential, that is, an instantaneous negative current of the size of the

threshold. Therefore, equation 2.21 can be factorized as follows:

$$\begin{aligned}\mu_k^{(l,j)} &= \sum_{z=k+1}^{n^{(l,j)}} \frac{\vartheta}{\tau x_z^{(l,j)}} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right) \delta_z^{(l,j)} \\ &= \frac{\vartheta}{\tau} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right) \sum_{z=k+1}^{n^{(l,j)}} \frac{\delta_z^{(l,j)}}{x_z^{(l,j)}} \\ &= \alpha_k^{(l,j)} \beta_k^{(l,j)}, \end{aligned} \tag{2.23}$$

where

$$\alpha_k^{(l,j)} := \frac{\vartheta}{\tau} \exp\left(\frac{t_k^{(l,j)}}{\tau}\right) \tag{2.24}$$

is the unique factor and

$$\beta_k^{(l,j)} := \sum_{z=k+1}^{n^{(l,j)}} \frac{\delta_z^{(l,j)}}{x_z^{(l,j)}} \tag{2.25}$$

is the backpropagated factor that can be accumulated in linear time complexity $\mathcal{O}(n)$ during the backward pass, which significantly reduces the processing time.

3 Results

In this section, we compare our proposed method with Fast & Deep. This evaluation was conducted based on multiple criteria, including performance on benchmark data sets, convergence rate, sparsity, classification latency, as well as robustness to noise and weight quantization.

Experimental conditions were standardized for both methods except for weight distributions and thresholds. Two uniform weight distributions, ($w_{i,j} \sim U(-1, 1)$ and $w_{i,j} \sim U(0, 1)$), were used to evaluate Fast & Deep, to measure the effect of initial weight distributions on the different evaluation criteria. Our method was solely assessed using $w_{i,j} \sim U(-1, 1)$, as positive initial weights lead to excessive spiking activity, hindering computational and energy efficiency. Thresholds were manually tuned to find the best-performing networks and kept fixed during training. In our experiments, all layers (including convolutional layers) have been directly trained using our proposed method and Fast & Deep, and no conversion from DNN to SNN has been performed. More details about our experimental settings are in the appendixes.

Table 1: Performance Comparisons between Fast & Deep and Our Method on the MNIST, EMNIST, Fashion-MNIST, and Spiking Heidelberg Digits (SHD) Data Sets.

Data Set	Architecture	Fast & Deep $U(-1, 1)$	Fast & Deep $U(0, 1)$	Our Method $U(-1, 1)$
MNIST	800-10	$96.76 \pm 0.17\%$	$97.83 \pm 0.08\%$	$\mathbf{98.88 \pm 0.02\%}$
	Conv.	$99.01 \pm 0.16\%$	$99.22 \pm 0.05\%$	$\mathbf{99.38 \pm 0.04\%}$
EMNIST	800-47	$69.56 \pm 6.70\%$	$83.34 \pm 0.27\%$	$\mathbf{85.75 \pm 0.06\%}$
Fashion MNIST	400-400-10	$88.14 \pm 0.08\%$	$88.47 \pm 0.20\%$	$\mathbf{90.19 \pm 0.12\%}$
SHD	128-20	$33.84 \pm 1.35\%$	$47.37 \pm 1.65\%$	$\mathbf{66.8 \pm 0.76\%}$

Notes: The initial weight distribution used in each column is specified in the first row of the table. Conv. refers to a convolutional SNN with the following architecture: 15C5-P2-40C5-P2-300-10. The numbers in bold refer to the highest accuracy obtained by the method in each row.

3.1 Performance and Convergence Rate. To assess the performance of our proposed method, we trained fully connected SNNs on the MNIST (LeCun et al., 2010), EMNIST (Cohen et al., 2017) (Balanced Extended MNIST), and Fashion-MNIST (Xiao et al., 2017) data sets as well as convolutional SNNs on MNIST. We also evaluated our method on temporal data classification by training fully connected networks on the Spiking Heidelberg Digits (SHD) data set (Cramer et al., 2022), an English and German spoken digits classification task. We compared our results to those obtained using the original Fast & Deep algorithm. Table 1 summarizes the average test accuracies of both methods given the considered initial weight distributions. For completeness, a comparison of Fast & Deep and our method with other spike-based BP algorithms is in the appendix.

First, it should be noted that the Fast & Deep algorithm generally achieves better performance when the weights are initialized with positive values, which is consistent with the choice of weight distribution made by Göltz et al. (2021). Second, our proposed method demonstrates superior performance compared to the Fast & Deep algorithm, with improvement margins ranging from 1.05% on MNIST to 2.41% on the more difficult EMNIST data set. This is not surprising given that unconstrained SNNs are known to perform better compared to TTFS networks (Jin et al., 2018; Zhang & Li, 2019; Shrestha & Orchard, 2018; Lee et al., 2016; Zhang et al., 2022). Moreover, our method outperforms by at least 19% Fast & Deep on the SHD data set with a single hidden layer of 128 neurons. Note that no recurrent connections were used in these experiments. To understand the reason for this performance gap, we analyzed the spiking activity in the hidden layers after training. Figure 3 shows that TTFS neurons respond only to early stimuli. In temporal coding, high-valued information is encoded

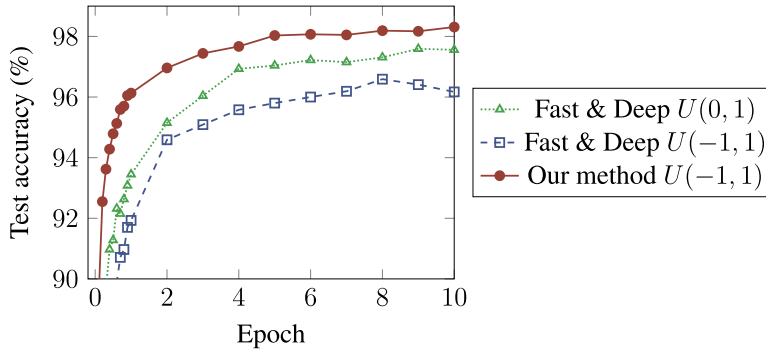


Figure 2: Test accuracy of Fast & Deep and our method on MNIST for the same learning rate. The unconstrained SNN trained with our method benefits from a higher convergence rate than the temporally-coded networks trained with Fast & Deep. The two SNNs trained with Fast & Deep have similar convergence rates despite their difference in initial weight distribution.

by early spikes. Training with a one-spike constraint is therefore energy efficient but tends to make SNNs spike as early as possible, thus missing the information occurring later in time. In contrast, neurons trained without spike constraint are able to respond throughout the duration of the sample, thus capturing all the information despite an increased number of spikes fired. This highlights the importance of firing more than once and demonstrates that a trade-off exists between performance and energy consumption when processing temporal data.

In addition, our results indicate that the convergence rate of SNNs with multiple spikes is higher compared to TTFS networks. Figure 2 depicts the evolution of the test accuracy of both methods on MNIST. This demonstrates that our method can reach desired accuracies in fewer epochs compared to Fast & Deep. Such improvement implies that discriminative features are learned earlier, during training.

3.2 Network Sparsity. Achieving high sparsity in trained SNNs is critical for energy efficiency, as neuromorphic hardware consumes energy only at spike events.

Figure 4 shows the population spike counts of fully connected SNNs trained on MNIST using both methods. It appears that the initial weight distribution plays an important role in the final sparsity level of the trained SNNs. In the analyzed case, SNNs initialized with positive weights appear to be less sparse after training than SNNs initialized with both negative and positive weights.

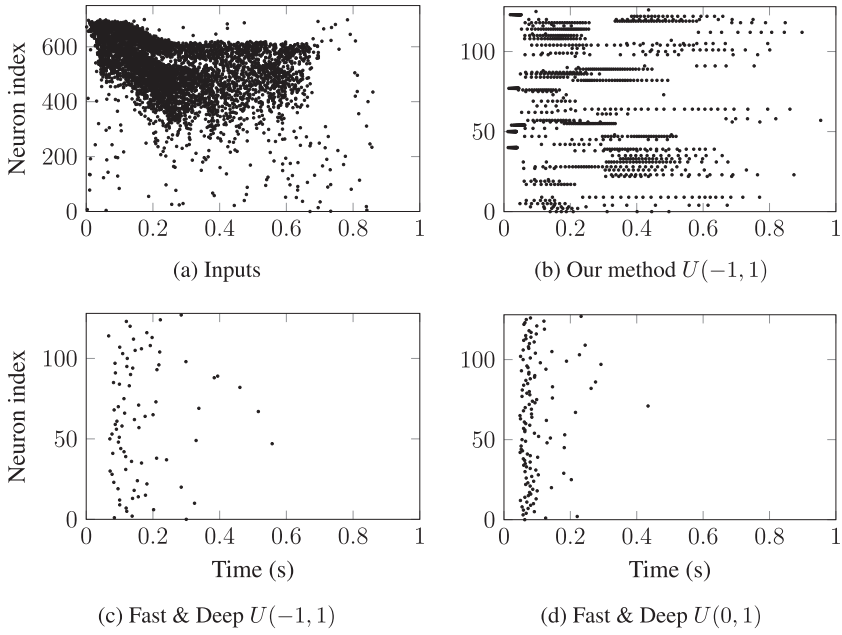


Figure 3: Spiking activity of hidden neurons in a SNN trained with our method (panel b) and Fast & Deep (panels c and d) given a spoken "zero" (panel a) from the SHD data set. TTFS neurons in Fast & Deep mainly respond to early stimuli, missing most of the input information. In contrast, our method allows for multiple spikes per neuron, which enables them to capture all the information from the inputs. This demonstrates the importance of relaxing the spike constraint of TTFS when processing temporal data.

While our proposed method allows for an increased number of spikes per neuron, which implies more energy consumption, we found that it can achieve similar sparsity as TTFS networks initialized with both negative and positive weights while performing better than TTFS networks initialized with positive weights only (see Figure 4 and Table 1). This suggests that our proposed method can offer improved trade-offs between accuracy and sparsity.

To understand why our method can achieve such levels of sparsity despite not imposing any constraints on neuron firing, we analyzed the average activity in each network. Figure 5 shows that neurons trained with Fast & Deep fire indiscriminately in response to any input digit, which is characteristic of temporal coding where information is represented by the timing of spikes rather than the presence or absence of spikes. In contrast, SNNs trained with our method exhibit a different distribution of firing activity, with certain key neurons selectively responding to specific digits. Figure 4

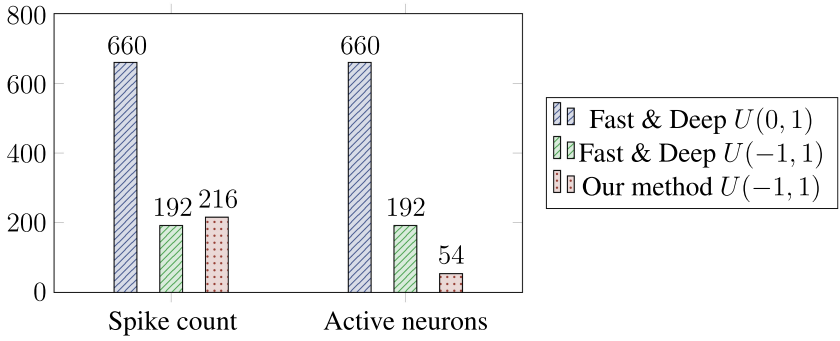
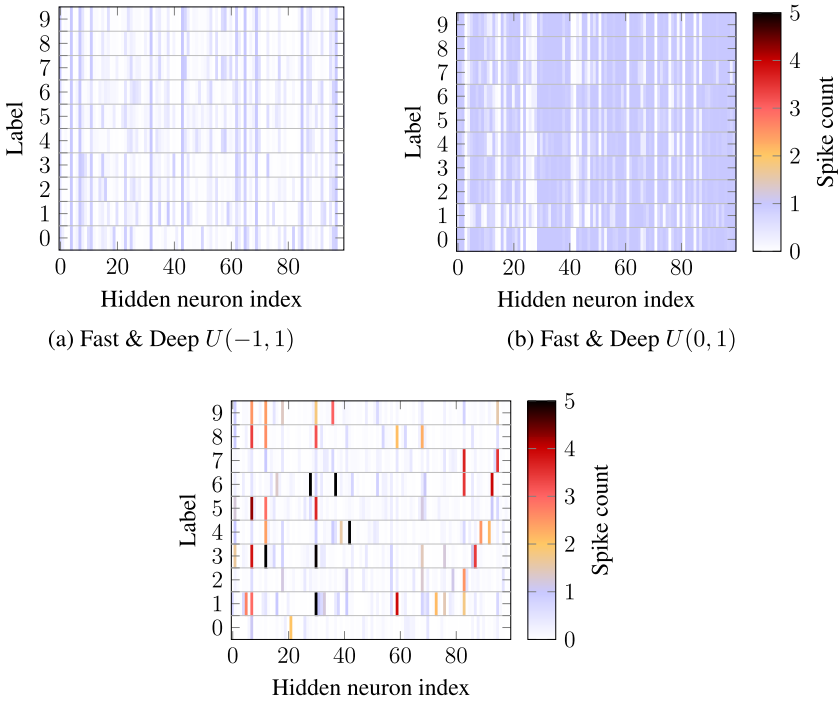


Figure 4: Comparison of the population spike count and the number of active neurons in fully connected SNNs trained using Fast & Deep and our proposed method on the MNIST data set. These results indicate that the sparsity of SNNs after training depends on the weight distribution, with the SNNs initialized with only positive weights appearing to be less sparse than those initialized with both negative and positive weights. In addition, our proposed method demonstrates a similar level of sparsity and fewer active neurons as Fast & Deep for the same initial weight distribution, despite the relaxed constraint on neuron spike counts.

indicates that only 7% of neurons trained with our method are active during inference (i.e., neurons that fire at least once). In comparison, the SNN trained with Fast & Deep and initialized with both negative and positive weights has 24% of its neurons firing. Therefore, the reduced proportion of active units in our method compensates for the increased number of spikes per neuron, leading to fewer spikes emitted in the network.

We found during our experiments that the sparsity of SNNs trained with our method was also influenced by the choice of threshold values. More precisely, we observed that decreasing the output threshold resulted in a reduction of activity in the network, as illustrated in Figure 6a. This decrease in activity occurred because the output neurons fired more often, which is illustrated in Figure 6b. The increased number of output spikes caused the loss function to produce more negative errors, which resulted in negative changes of weights in the hidden layer. However, lowering the threshold has a dual impact on activity: it increases the firing rates at initialization but also contributes to producing more negative errors, which can decrease activity during learning. While controlling sparsity using thresholds is trivial in shallow SNNs, a fine balance between thresholds has to be found to control sparsity in Deep SNNs, making it challenging to achieve sparsity with larger architectures. However, our findings suggest that threshold values play a crucial role in determining the sparsity of unconstrained SNNs and can be seen as a way to control activity without the need for firing rate regularization.



(a) Fast & Deep $U(-1, 1)$

(b) Fast & Deep $U(0, 1)$

(c) Our method $U(-1, 1)$

Figure 5: Panels a and b show the average spike count of hidden neurons trained with Fast & Deep on the MNIST data set while panel c shows the average spike count of hidden neurons trained with our proposed method. Each row corresponds to the average activity over all the test samples of a particular digit. As TTFS networks mainly encode information temporally, we observe that neurons trained with Fast & Deep fire indiscriminately in response to stimuli, making it difficult to differentiate the labels from the mean spike count, regardless of the initial weight distribution. However, our proposed SNN training method results in a different distribution of firing activity. More precisely, key neurons respond selectively to particular digits, while most of the other neurons remain mostly silent.

3.3 Prediction Latency. We also investigated the prediction latency of SNNs trained using different methods. This corresponds to the amount of simulation time the models need to reach full confidence in their predictions and determines the simulation duration required to achieve high accuracy.

Figure 7a shows the averaged prediction confidence over time for SNNs trained using each method. We found that when initialized with only positive weights, Fast & Deep achieved high confidence on predictions

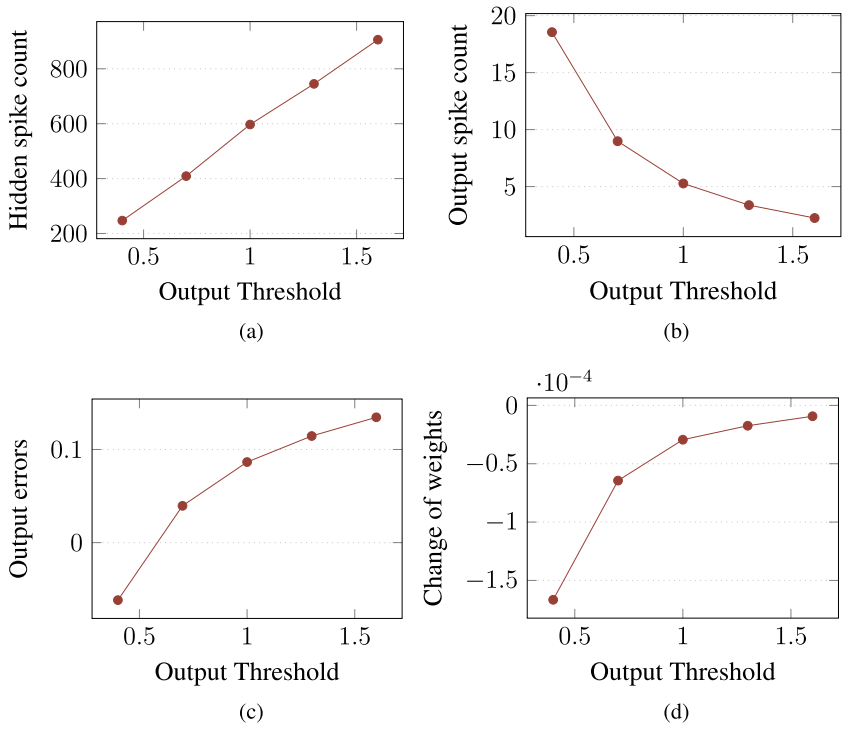


Figure 6: Influence of the output threshold on the sparsity of a two-layer SNN trained on MNIST with our method. Panel a illustrates that a lower output threshold results in fewer spikes generated after one epoch. Panel b indicates that decreasing the output threshold increases the initial activity in the output layer, thereby leading to a greater number of negative errors transmitted during the backward pass (as shown in panel c). This, in turn, leads to a decrease in the weights in the hidden layer, as depicted in panel d.

after more than 150 ms. However, when initialized with both negative and positive weights, both Fast & Deep and our proposed method achieved confidence earlier (in 20 ms and 50 ms, respectively). This suggests that initialization has a significant impact on prediction latency. However, SNNs trained with our proposed method are slightly slower than TTFS networks trained with Fast & Deep for the same weight distribution due to the increased spike count per neuron.

Despite this difference, the short latency of these networks allows for shorter simulation durations, which can further improve sparsity without affecting performance. In Figure 7b, we show the relationship between population spike count and accuracy for each SNN as simulation time increases from 0 to 200 milliseconds. This demonstrates that by reducing the

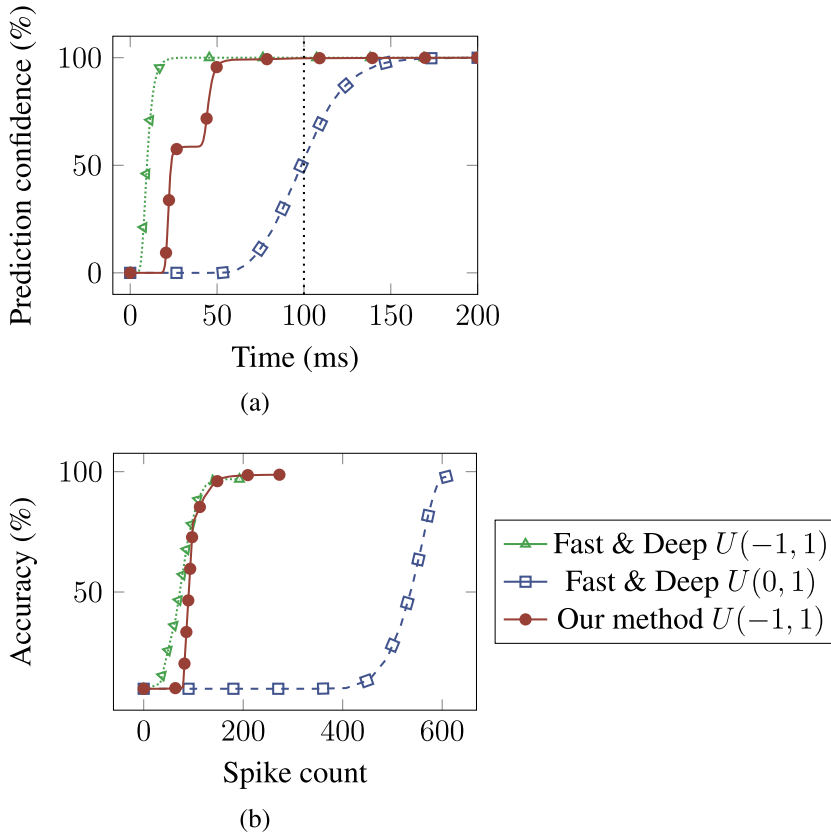


Figure 7: (a) The evolution of the average prediction confidence during simulations on the MNIST test set. To produce this figure, we measured the probability of predictions at a time t being equal to the final predictions at the end of the simulations. The vertical dotted line represents the end of input spikes. The initial weight distribution seems to have a crucial impact on the latency of predictions. More precisely, negative initial weights produce confidence earlier than positive initial weights. Therefore, simulation time can be reduced to further improve sparsity. (b) The relationship between spike count and accuracy as the simulation time increases. It demonstrates that the duration of simulation can be used as a posttraining method to further reduce energy consumption while maintaining high performance.

simulation time, SNNs can become sparser while maintaining high performance. Therefore, the sparsity-accuracy trade-off can be further improved after training by adjusting the simulation duration. This also demonstrates that our method can align with the level of the sparsity of Fast & Deep while still performing better.

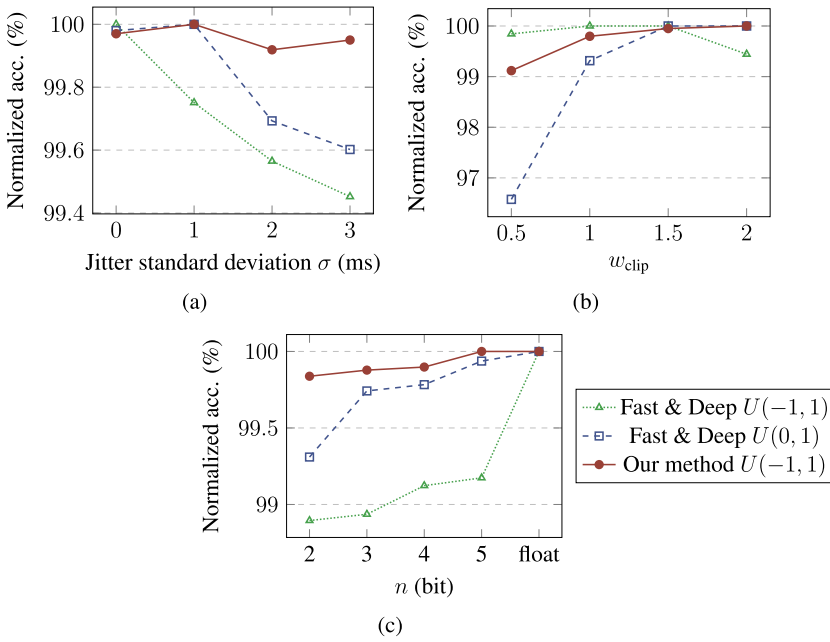


Figure 8: (a) The effect of spike jitter on the performance of each method. This was achieved by introducing artificial noise to the spike timings, following a normal distribution $\mathcal{N}(0, \sigma)$. (b) The impact of weight clipping, which involved restricting weights to the range $[-w_{\text{clip}}, w_{\text{clip}}]$ during training. (c) The effect of weight precision, which was obtained by discretizing weights into $2^{n+1} - 1$ bins (n bits plus one bit for the sign of the synapse) within the range $[-1, 1]$. Overall, our method was found to be more resilient to noise and reduced weight precision than Fast & Deep.

3.4 Robustness to Noise and Weights Quantization. Analog neuromorphic hardware is inherently noisy and often limited to specific ranges and resolutions of weights. Having a model that is robust to noise and weight quantization is therefore important to achieve high performance on such systems. To assess the robustness of each method, we measured the impact of spike jitter, weight clipping, and weight precision on their accuracy. In these experiments, performance was normalized with the maximum accuracy of each method to better compare variations.

Figure 8a shows the impact of spike jitter on the performance of each method. These results were produced by artificially adding noise to spike timings with a normal distribution $\mathcal{N}(0, \sigma)$ during training. When initialized with both negative and positive weights, Fast & Deep appears to be less robust than using only positive weights. With negative weights, only a

fraction of neurons transmits information, which leads to an increased sparsity, as illustrated in Figure 4. Therefore, introducing noise to spike timings has a significant impact on performance. In contrast, positive weights ensure consistent network activity and redundancy in transmitted information. SNNs initialized with positive weights are thus less affected by spike jitter. However, SNNs trained with Fast & Deep remain susceptible to noise, even with positive initial weights. Perturbations in spike timing still have a critical impact on temporal coding. In contrast, our proposed method demonstrates greater robustness to spike jitter than Fast & Deep, with minimal variation observed. This is a result of the redundancy created by the multiple spikes fired by neurons.

In Figure 8b, we demonstrate the effect of weight range on performance by clipping weights between the range $[-w_{\text{clip}}, w_{\text{clip}}]$ during training. The performance of Fast & Deep initialized with positive weights degrades when w_{clip} is lower than 1.5. However, both our method and Fast & Deep exhibit robustness to reduced weight ranges when initialized with both negative and positive weights. This suggests that weight distribution may play a role in the network's resilience to limited weight ranges.

Finally, Figure 8c shows the performance of each method with reduced weight resolutions from 5 to 2 bits (results with float precision are also given as a reference). It highlights that Fast & Deep is less robust to reduced weight precision than our method, particularly with negative weights. In contrast, our approach is only slightly affected by the decreased precision, even when reduced to as low as 2 bits.

4 Discussion

In this work, we explored the trade-offs between performance and various aspects of TTFS SNNs such as sparsity, classification latency, and robustness to noise and weight quantization. We also generalized the Fast & Deep algorithm by incorporating a reset of the membrane potential, which enables multiple spikes per neuron, and compared the improvements of the proposed method with the origin algorithm on those trade-offs.

We found that initializing Fast & Deep with positive weights leads to better generalization capabilities compared to initializing with both negative and positive weights. This observation is consistent across the benchmarked data sets, as shown in Table 1. However, relaxing the spike constraint improves the overall performance and convergence rate of SNNs, at least on the benchmark problems we considered. This result was expected before our experiments since BP methods that use multiple spikes per neuron generally perform better than methods that impose firing constraints (Jin et al., 2018; Zhang & Li, 2019; Shrestha & Orchard, 2018; Lee et al., 2016; Zhang et al., 2022).

Our experiments also demonstrate that the weight distribution significantly influences the sparsity of Fast & Deep. We observed that SNNs with

positive weight initialization tend to be less sparse than those initialized with weights between −1 and 1. However, the former consistently outperforms the latter in terms of performance. This highlights the accuracy-sparsity trade-off often observed when training SNNs (Yin et al., 2023; Li et al., 2021). The quasi-dense activity provided by positive weights explains the difference in sparsity, as shown in Figure 5b. In contrast, initializing Fast & Deep with both negative and positive weights leads to fewer active neurons due to the inhibition provided by negative weights. Additionally, neurons trained with Fast & Deep fire indiscriminately to stimuli, suggesting a pure temporal representation of information, whereas neurons trained with our proposed method selectively respond to their inputs and exhibit a different distribution of activity, as shown in Figure 5c. Our unconstrained SNNs allow for a different distribution of the spike activity, whereby key neurons can fire more often than others, while irrelevant neurons may not spike at all. This enables our method to achieve a level of sparsity comparable to Fast & Deep on image classification, as illustrated in Figure 4.

To achieve a high degree of sparsity without firing rate regularization, thresholds can be tuned to indirectly influence spiking activity through learning. Decreasing thresholds increases the firing rate of downstream layers, resulting in more negative errors at outputs and consequently negative weight changes in hidden layers, as shown in Figure 6. This mechanism offers a natural way to control sparsity in unconstrained SNNs without requiring any regularization technique. By leveraging this principle, we were able to train shallow SNNs with a sparsity level similar to Fast & Deep while achieving higher performance. This implies that allowing multiple spikes per neuron has the potential to enhance the accuracy-sparsity trade-off and prompts further investigation into the effectiveness of TTFS in achieving efficient computation. However, finding thresholds that lead to high sparsity is more difficult when networks become deeper due to the fluctuations in the firing rates of each layer. The factors that influence sparsity in unregularized SNNs are currently not fully understood and present an opportunity for future research to investigate how to naturally achieve sparsity in deep architectures.

Our experiments on the SHD data set also demonstrated the significance of the accuracy-sparsity trade-off when processing temporal data. We found that the ability of neurons to fire multiple spikes is critical in capturing all the information about the inputs over time. However, since TTFS neurons encode important information in early spikes, they tend to fire too early to capture all the information, which makes them less effective than unconstrained SNNs on temporal data. Although TTFS SNNs are more energy efficient, the relaxation of spike constraints in unconstrained SNNs allows neurons to fire throughout the simulation, thereby capturing all the relevant information. Consequently, they perform significantly better than TTFS SNNs when processing temporal data.

In addition to performance and sparsity, we measured the prediction latency of each method, which is the waiting time required before the system can make reliable predictions. We found that the speed of classification was primarily driven by the weight distribution. Figure 7a shows that both Fast & Deep and our method achieve similar latencies when initialized between -1 and 1, with a slight advantage for Fast & Deep. However, when initialized with only positive values, Fast & Deep requires more simulation time to achieve full confidence in predictions. Low latency is advantageous not only in terms of inference speed but also in improving energy efficiency. If predictions occur early enough, the duration of simulations can be significantly reduced, limiting the number of spikes fired by neurons. In Figure 7b, we demonstrated that reducing the simulation time can lead to a reduction in computational cost for both Fast & Deep and our method while maintaining the same performance. This shows that prediction latency, energy consumption, and performance are closely related and that unconstrained SNNs can offer better trade-offs between these aspects than TTFS SNNs.

The last characteristic that we investigated is the robustness to the noise and weight quantization inherent in analog neuromorphic hardware. The timing of spikes is a critical factor for the performance of TTFS SNNs as it carries most of the information. Therefore, perturbations in these timings and weight constraints can significantly affect the reliability of the feature extraction. In contrast, our proposed method benefits from an increased number of spikes per neuron, providing redundancy that enhances resilience to noise and weight constraints. For instance, Figure 8a demonstrates that our method is less affected by perturbations in spike timings than Fast & Deep. This suggests that our proposed method has the potential to provide more stable learning on analog neuromorphic hardware than Fast & Deep.

Finally, our work specifically concentrated on backpropagation in SNNs with fixed thresholds and time constants. However, several studies have demonstrated that incorporating adaptive thresholds and trainable time constants can enhance the convergence, sparsity, and performance of SNNs (Zambrano et al., 2019; Chen et al., 2022; Fang et al., 2021; Yin et al., 2021). Therefore, future studies could explore the integration of adaptive thresholds and trainable time constants into our proposed method to further enhance the sparsity-accuracy trade-offs in SNNs.

5 Conclusion

Our work demonstrates that relaxing the spike constraint of TTFS SNNs results in improved trade-offs among performance, sparsity, latency, and noise robustness. Our findings also highlight the crucial role of thresholds in regulating the sparsity of unconstrained SNNs during learning, which could serve as a natural alternative to firing rate regularization. Although error backpropagation algorithms for SNNs are incompatible with

neuromorphic hardware, their development provides valuable insights into how spiking neurons affect objective functions and could support the development of hardware-compatible algorithms. Therefore, our work contributes to a better understanding of how to compute exact gradients in SNNs and highlights the advantages of using multiple spikes per neuron over TTFS.

Appendix A: Experimental Settings

A.1 Simulations. We implemented the method introduced in section 2 in a custom Python simulator for GPUs using CuPy (Okuta et al., 2017). In our implementation, both simulations and error backpropagation are event based.

A.2 Input Encoding. We used a TTFS encoding scheme to benefit from a low number of input spikes and fast processing. For image classification tasks, we encoded the pixels into spike timing as follows:

Given the time window $T_{\text{enc}} = 100$ milliseconds and the maximum pixel value $X_{\max} = 255$, the input spike timing $t_1^{(1,ij_{\max}+j)}$ associated with the pixel value $x_{i,j}$ in row i and column j is computed as

$$t_1^{(1,ij_{\max}+j)} = \frac{T_{\text{enc}}}{X_{\max}} \left(X_{\max} - x_{i,j}\right). \tag{A.1}$$

where $j_{\max}$ is the width of the image in pixels. Neurons with a pixel value $x_{i,j} = 0$ do not produce any spikes to further limit the number of events to process. For convolutional SNNs, the same temporal encoding was used, but the shape of the input was set as a three-dimensional tensor corresponding to the image with a single channel.

A.3 Implementation of Fast & Deep. To reproduce Fast & Deep with TTFS models, we constrained the number of firings allowed per neuron to one in our implementation and used a TTFS softmax cross-entropy loss function, as described in (Göltz et al., 2021).

A.4 Architectures and Parameters. For the MNIST EMNIST and SHD data sets, we trained both TTFS and unconstrained fully connected SNNs with a batch size of 50 and a maximum number of spikes per neuron of 30 for the unconstrained SNNs. Output spike targets were set to 15 for the target label and 3 for the others. We used a learning rate of $\lambda = 0.003$ for image classification and $\lambda = 0.001$ for the SHD data set. No data augmentation was used with full connected networks.

For Fashion-MNIST, we implemented a three-layer, fully connected network composed of two hidden layers of 400 neurons each and a 10 neuron

output layer. We allowed a maximum number of spikes per neuron of 5 for the hidden layers and 20 for the output layer. We also set the target spike counts to 15 for the true class and 3 for the others. We used a batch size of 5 with a learning rate of $\lambda = 0.0005$, a learning rate decay factor of 0.5 every 10 epochs, and a minimum rate of 0.0001.

The weight kernels of convolution neurons were shared within each layer, as in rate-based CNNs. Convolution allows the detection of spatially correlated features and therefore makes networks invariant to translations. In contrast to fully connected SNNs, the translation invariances of CSNNs allow the networks to detect objects at different locations in space. We used a six-layer CSNN composed of two spiking convolutional layers of 15 5×5 and 40 5×5 filters, respectively, each followed by 2×2 spike aggregation pooling layer (i.e., the spike trains of input neurons are aggregated into a single spike train). The spikes of the last pooling layer are finally sent to two successive fully connected layers of sizes 300 and 10, respectively. Each layer allows an increasing number of spikes per neuron, starting from a single spike for the first convolutional layer, 3 for the second layer, 10 for the fully connected layer, and 30 spikes per neuron for the output layer. We also set the output spike targets to 30 for the true label and 3 for the others. The CSNN was also trained with data augmentation. In this case, we used elastic distortions (Simard et al., 2003) to transform the MNIST training images. We finally trained the networks for 100 epochs with a batch size of 20, a learning rate of $\lambda = 0.003$, a decay factor of 0.5 every 10 epochs, and a minimum rate of 0.0001.

In all our experiments, we used the Adam optimizer with the values of β_1, β_2, and ϵ set as in the original paper (Kingma & Ba, 2015). Initial weights were randomly drawn from a uniform distribution $U[a, b]$. Networks trained on image classification used the same base time constant of $\tau_s = 0.130$. For the SHD data set, we used a time constant of $\tau_s = 0.100$. All thresholds were manually tuned to find the best-performing networks for each method and data set. Thresholds were then kept fixed during training. Finally, we did not use any regularization or synaptic scaling techniques in any of our experiments to provide a fair comparison between TTFS and unconstrained SNNs.

Appendix B: Fully Connected SNNs on MNIST

Table 2: Performance of Several Methods on the MNIST Data Set.

Method	Architecture	Test Accuracy
TTFS		
Fast & Deep (Göltz et al., 2021)	350	$97.1 \pm 0.1\%$
Wunderlich & Pehle (Wunderlich & Pehle, 2021)	350	$97.6 \pm 0.1\%$
Alpha Synapses (Comsa et al., 2020)	340	97.96%
S4NN (Kheradpisheh & Masquelier, 2020)	400	$97.4 \pm 0.2\%$
BS4NN (Kheradpisheh et al., 2021)	600	97.0%
Mostafa (Mostafa, 2016)	800	97.2%
STDBP (Zhang et al., 2022)	800	98.5%
Fast & Deep (Göltz et al., 2021) (our implementation)	**800**	**$97.83 \pm 0.08\%$**
Unconstrained		
eRBP (Neftci et al., 2017)	2×500	97.98%
Lee et al. (Lee et al., 2016)	800	98.71%
HM2-BP (Jin et al., 2018)	800	$98.84 \pm 0.02\%$
This work	**800**	**98.88 ± 0.02%**

Note: Results for Fast & Deep and our method are highlighted in bold.

Appendix C: Fully Connected SNNs on EMNIST

Table 3: Performance of Several Methods on the EMNIST Data Set.

Method	Architecture	Test Accuracy
TTFS		
Fast & Deep (Göltz et al., 2021) (our implementation)	**800**	**$83.34 \pm 0.27\%$**
Unconstrained		
eRBP (Neftci et al., 2017)	2×200	78.17%
HM2-BP (Jin et al., 2018)	2×200	$84.31 \pm 0.10\%$
HM2-BP (Jin et al., 2018)	800	$85.41 \pm 0.09\%$
This work	**800**	**$85.75 \pm 0.06\%$**

Note: Results for Fast & Deep and our method are highlighted in bold.

Appendix D: Fully Connected SNNs on Fashion MNIST

Table 4: Performances of Several Methods on the Fashion-MNIST Data Set.

Method	Architecture	Test Accuracy
TTFS		
S4NN (Kheradpisheh & Masquelier, 2020)	1000	88.0%
BS4NN (Kheradpisheh et al., 2021)	1000	87.3%
STDBP (Zhang et al., 2022)	1000	88.1%
Fast & Deep (Göltz et al., 2021) (our implementation)	**2 × 400**	**88.28 ± 0.41%**
Unconstrained		
HM2-BP (Jin et al., 2018)	2 × 400	88.99%
TSSL-BP (Zhang & Li, 2020)	2 × 400	89.75 ± 0.03%
ST-RSBP[a] (Zhang & Li, 2019)	2 × 400	90.00 ± 0.14%
This work	**2 × 400**	**90.19 ± 0.12%**

Note: Results for Fast & Deep and our method are highlighted in bold.
[a]The trained model has recurrent connections.

Appendix E: Fully Connected SNNs on SHD

Table 5: Performance of Several Methods on the Spiking Heidelberg Digits (SHD) Data Set.

Method	Architecture	Test Accuracy
TTFS		
Fast & Deep (Göltz et al., 2021) (our implementation)	**128**	**47.37 ± 1.65%**
Unconstrained		
Cramer et al. (Cramer et al., 2022)	128	48.10 ± 1.6%
Cramer et al.[a] (Cramer et al., 2022)	128	71.4 ± 1.9%
This work	**128**	**66.79 ± 0.66%**

Note: Results for Fast & Deep and our method are highlighted in bold.
[a]The trained model has recurrent connections.

Appendix F: Convolutional SNNs on MNIST

Table 6: Network Architectures used in Table 7.

Network Name	Architecture
Net1	32C5-P2-16C5-P2-10
Net2	12C5-P2-64C5-P2-10
Net3	15C5-P2-40C5-P2-300-10
Net4	20C5-P2-50C5-P2-200-10
Net5	32C5-P2-32C5-P2-128-10
Net6	16C5-P2-32C5-P2-800-128-10

Note: 15C5 represents a convolution layer with 15 5×5 filters and P2 represents a 2×2 pooling layer.

Table 7: Performance of Several Methods on the MNIST Data Set with Spiking Convolutional Neural Networks.

Method	Architecture	Test Accuracy
TTFS		
Zhou et al.[a] (Zhou et al., 2021)	Net1	99.33%
STDBP[a] (Zhang et al., 2022)	Net6	99.4%
Fast & Deep (our implementation)	**Net3**	**$99.22 \pm 0.05\%$**
Fast & Deep[a] (our implementation)	**Net3**	**$99.46 \pm 0.01\%$**
Unconstrainned		
Lee et al.[a] (Lee et al., 2016)	Net4	99.31%
HM2-BP[a] (Jin et al., 2018)	Net3	$99.42\% \pm 0.11\%$
TSSL-BP (Zhang & Li, 2020)	Net3	$99.50 \pm 0.02\%$
ST-RSBP[a] (Zhang & Li, 2019)	Net2	$99.50 \pm 0.03\%$
ST-RSBP[a] (Zhang & Li, 2019)	Net3	$99.57 \pm 0.04\%$
This work	**Net3**	**$99.38 \pm 0.04\%$**
This work[a]	**Net3**	**$99.60 \pm 0.03\%$**

Note: The network topologies are given in Table 6. [a]The network has been trained using data augmentation.

Code Availability

The code produced in this work will be made available at: https://github.com/Florian-BACHO/bats

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., . . . Modha, D. S. (2015). TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557. 10.1109/TCAD.2015.2474396

Blouw, P., Choo, X., Hunsberger, E., & Eliasmith, C. (2019). Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop*.

Bohté, S. M., Kok, J. N., & Poutré, H. L. (2000). Spikeprop: Backpropagation for networks of spiking neurons. In *Proceedings of ESANN*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., . . . Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 1877–1901). Curran.

Chen, Y., Mai, Y., Feng, R., & Xiao, J. (2022). An adaptive threshold mechanism for accurate and efficient deep spiking convolutional neural networks. *Neurocomputing*, *469*, 189–197. 10.1016/j.neucom.2021.10.080

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). Emnist: Extending MNIST to handwritten letters. In *Proceedings of the 2017 International Joint Conference on Neural Networks* (pp. 2921–2926).

Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., & Alakuijala, J. (2020). Temporal coding in spiking neural networks with alpha synaptic function: Learning with backpropagation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*.

Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2022). The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(7), 2744–2757. 10.1109/TNNLS.2020.3044364

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., . . . Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. 10.1109/MM.2018.112130359

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *Bert: Pre-training of deep bidirectional transformers for language understanding*. arXiv:1810.04805.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proceedings of the 2015 International Joint Conference on Neural Networks* (pp. 1–8).

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., & Tian, Y. (2021). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 2661–2671).

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 051001. 10.1088/1741-2560/13/5/051001

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press.

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., . . . Petrovici, M. A. (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature Machine Intelligence*, *3*(9), 823–835.

Guo, W., Yantır, H. E., Fouda, M. E., Eltawil, A. M., & Salama, K. N. (2020). Towards efficient neuromorphic hardware: Unsupervised adaptive neuron pruning. *Electronics*, *9*(7).

Hendy, H., & Merkel, C. (2022). Review of spike-based neuromorphic computing for brain-inspired vision: Biology, algorithms, and hardware. *Journal of Electronic Imaging*, *31*(1), 1–25. 10.1117/1.JEI.31.1.010901

Jin, Y., Zhang, W., & Li, P. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems*, *31*. Curran.

Kheradpisheh, S. R., & Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, *30*. 10.1142/S0129065720500276

Kheradpisheh, S. R., Mirsadeghi, M., & Masquelier, T. (2021). BS4NN: Binarized spiking neural networks with temporal coding and learning. *Neural Processing Letters*, *54*(2), 1255–1273. 10.1007/s11063-021-10680-x

Kim, J., Park, J., Joo, S., & Jung, S.-O. (2020). Efficient hardware implementation of STDP for AER-based large-scale SNN neuromorphic system. In *Proceedings of the 35th International Technical Conference on Circuits/Systems, Computers and Communications* (pp. 1–4).

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *Proceedings of the 3rd International Conference on Learning Representations*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems*, *25*. Curran.

LeCun, Y., Cortes, C., & Burges, C. (2010). MNIST handwritten digit database. ATT Labs. http://yann.lecun.com/exdb/mnist, 2.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*, 508.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., & Gu, S. (2021). Differentiable spike: Rethinking gradient-descent for training spiking neural networks. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), *Advances in neural information processing systems*, *34* (pp. 23426–23439). Curran.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671. 10.1016/S0893-6080(97)00011-7

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with deep reinforcement learning*. arXiv:1312.5602.

Mostafa, H. (2016). Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7).

Neftci, E. O., Augustine, C., Paul, S., & Detorakis, G. (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, *11*, 324. 10.3389/fnins.2017.00324

Okuta, R., Unno, Y., Nishino, D., Hido, S., & Loomis, C. (2017). CuPy: A numPy-compatible library for NVIDIA GPU calculations. In *Proceedings of Workshop on*

Machine Learning Systems in the Thirty-First Annual Conference on Neural Information Processing Systems.

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., . . . Furber, S. B. (2013). SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, *48*(8), 1943–1953. 10.1109/JSSC.2013.2259038

Park, S., Lee, D., & Yoon, S. (2021). Noise-robust deep spiking neural networks with temporal information. In *Proceedings of the 58th ACM/IEEE Design Automation Conference* (pp. 373–378). IEEE.

Schmitt, S., Klaehn, J., Bellec, G., Gruebl, A., Güttler, M., Hartel, A., . . . Meier, K. (2017). *Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system*. arXiv:1703.01909

Shrestha, S. B., & Orchard, G. (2018). SLAYER: Spike layer error reassignment in time. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems*, *31*. Curran.

Simard, P., Steinkraus, D., & Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition* (pp. 958–963).

Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *26*. Curran.

Taunyazov, T., Sng, W., See, H. H., Lim, B., Kuan, J., Ansari, A. F., Tee, B. C., & Soh, H. (2020). *Event-driven visual-tactile sensing and learning for robots*. arXiv:2009.07083.

Thorpe, S., Fize, D., & Marlot, C. (1996). Speed of processing in the human visual system. *Nature*, *381*, 520–522. 10.1038/381520a0

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. In I. Guyon, Y. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*, *30*. Curran.

Wu, Y., Deng, L., Li, G., Zhu, J., & Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, *12*, 331.

Wunderlich, T. C., & Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, *11*(1). 10.1038/s41598-021-91786-z

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms.

Yan, Y., Chu, H., Jin, Y., Huan, Y., Zou, Z., & Zheng, L. (2022). Backpropagation with sparsity regularization for spiking neural network learning. *Frontiers in Neuroscience*, *16*.

Yin, B., Corradi, F., & Bohté, S. M. (2021). Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nature Machine Intelligence*, *3*(10), 905–913. 10.1038/s42256-021-00397-w

Yin, R., Kim, Y., Li, Y., Moitra, A., Satpute, N., Hambitzer, A., & Panda, P. (2023). *Workload-balanced pruning for sparse spiking neural networks*. arXiv:2302.06746

Zambrano, D., Nusselder, R., Scholte, H. S., & Bohté, S. M. (2019). Sparse computation in adaptive spiking neural networks. *Frontiers in Neuroscience*, *12*. 10.3389/fnins.2018.00987

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., . . . Li, H. (2022). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(5), 1947–1958. 10.1109/TNNLS.2021.3110991

Zhang, W., & Li, P. (2019). Spike-train level backpropagation for training deep recurrent spiking neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, *32*. Curran.

Zhang, W., & Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, *33* (pp. 12022–12033). Curran.

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S. T., & Sanyal, A. (2021). Temporal-coded deep spiking neural network with easy training and robust performance. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(12), 11143–11151. 10.1609/aaai.v35i12.17329

Received March 1, 2023; accepted June 3, 2023.