# Exact Gradient Computation for Spiking Neural Networks via Forward Propagation

**Jane H. Lee**
Yale University

**Saeid Haghighatshoar**
SynSense

**Amin Karbasi**
Yale University

## Abstract

Spiking neural networks (SNN) have recently emerged as alternatives to traditional neural networks, owing to energy efficiency benefits and capacity to capture biological neuronal mechanisms. However, the classic backpropagation algorithm for training traditional networks has been notoriously difficult to apply to SNN due to the hard-thresholding and discontinuities at spike times. Therefore, a large majority of prior work believes exact gradients for SNN w.r.t. their weights do not exist and has focused on approximation methods to produce surrogate gradients. In this paper, (1) by applying the implicit function theorem to SNN at the discrete spike times, we prove that, albeit being non-differentiable in time, SNNs have well-defined gradients w.r.t. their weights, and (2) we propose a novel training algorithm, called *forward propagation* (FP), that computes exact gradients for SNN. FP exploits the causality structure between the spikes and allows us to parallelize computation forward in time. It can be used with other algorithms that simulate the forward pass, and it also provides insights on why other related algorithms such as Hebbian learning and also recently-proposed surrogate gradient methods may perform well.

## 1 Introduction

Spiking neural networks (SNNs), inspired by biological neuronal mechanisms and sometimes referred to as the third generation of neural networks such as by (Maass, 1997), have garnered considerable attention recently (e.g., see Roy et al. (2019); Panda et al. (2020); Cao et al. (2015); Comsa et al. (2020); Diehl and eCook (2015)) as low-power alternatives. For instance, SNNs have been shown to yield orders of magnitude energy saving over ANNs on emerging neuromorphic hardware (Akopyan et al., 2015; Davies et al., 2018). SNNs have other unique properties, owing to their ability to model biological mechanisms such as dendritic computations with temporally evolving potentials (e.g., Gidon et al. (2020)) or short-term plasticity, which allow them to even outperform ANNs in accuracy in some tasks (e.g., Moraitis et al. (2020)). The power of neuromorphic computing can even be seen in ANNs, e.g., (Jeffares et al., 2022) use rank-coding in ANN inspired by the temporal encoding of information in SNNs. However, due to the discontinuous resetting of the membrane potential in spiking neurons, e.g., in Integrate-and-Fire (IF) or Leaky-Integrate-and-Fire (LIF) type neurons (refer to Burkitt (2006); Kornijcuk et al. (2016)), it is notoriously difficult to calculate gradients and train SNNs by conventional methods. For instance, (Jeffares et al., 2022) use the fact that "spike coding poses difficulties and training that require ad hoc mitigation" and "SNNs are particularly difficult to analyse mathematically" to motivate rank-coding for ANN. As such, many existing works on training SNN do so without exact gradients, which range from heuristic rules like Hebbian learning, (e.g., Kempter et al. (1999); Ruf and Schmitt (2006)) and STDP, (e.g., Lee et al. (2018); Lobov et al. (2020)), to methods like SNN-ANN conversion, (e.g., Rueckauer et al. (2017); Ding et al. (2021); Ho and Chang (2021)), and surrogate gradient approximations, e.g., (Neftci et al., 2019).

In this work, by applying the implicit function theorem (IFT) at the firing times of the neurons in SNN, we first show that under fairly general conditions, gradients of loss w.r.t. network weights are well-defined. We then provide what we call a *forward-propagation* (FP) algorithm which uses the causality structure in network firing times and our IFT-based gradient calculations in order to calculate exact gradients of the loss w.r.t. network weights. We call it forward propagation because intermediate calculations needed to calculate the final gradient are actually done forward in time (or forward in layers for feed-forward networks). Our method was developed independently and in parallel to the recent Forward-Forward algorithm by (Hinton, 2022), which shares the idea that the algorithm computes the nec-

Proceedings of the 26[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

essary quantities for gradients in the forward pass, rather than the usual backpropagation. We highlight the following:

- Our method can be applied in networks with arbitrary recurrent connections (up to self loops) and is agnostic to how the forward pass is implemented. We provide an implementation for computing the firing times in the forward pass, but as long as we can obtain accurate firing times and causality information (for instance, using existing libraries), we can calculate gradients.
- Our method can be seen as an extension of Hebbian learning as it illustrates that the gradient w.r.t. a weight $W_{ji}$ connecting neuron $j$ to neuron $i$ is almost an average of the feeding kernel $y_{ji}$ between these neurons at the firing times. In the context of Hebbian learning (especially from a biological perspective), this is interpreted as the well-known fact that *stronger feeding/activation amplifies the association between the neurons.* (See (Choe, 2013; Gerstner et al., 2014).)
- In our method, the smoothing kernels $y_{ji}$ arise naturally as a result of application of IFT at firing times, resembling the smoothing kernels applied in surrogate gradient methods. As a result, (1) our method sheds some light on why the surrogate gradient methods may work well, and (2) in our method, the smoothing kernels $y_{ji}$ vary according to the firing times between two neurons; thus, they can be seen as an adaptive version of the fixed smoothing kernels used in surrogate methods. See remark 2.

### 1.1 Related Work

A review of learning in deep spiking networks can be found at (Tavanaei et al., 2018; Pfeiffer and Pfeil, 2018; Roy et al., 2019; Wang et al., 2020), with (Roy et al., 2019) discussing also developments in neuromorphic computing in both software (algorithms) and hardware. (Neftci et al., 2019) focuses on surrogate gradient methods, which use smooth activation functions in place of the hard-thresholding for compatibility with usual backpropagation and have been used to train SNNs in a variety of settings, e.g., (Esser et al., 2016; Bellec et al., 2018; Huh and Sejnowski, 2018; Zenke and Ganguli, 2018; Shrestha and Orchard, 2018; Safa et al., 2021).

A number of works explore backpropagation in SNNs (Bohté et al., 2000; Jin et al., 2018; Zhang and Li, 2019). The SpikeProp framework by (Bohté et al., 2000) assumes a linear relationship between the post-synaptic input and the resultant spiking time, which our framework does not rely on. The method in (Jin et al., 2018) and its RSNN version by (Zhang and Li, 2019) are limited to a rate-coded loss that depends on spike counts. The continuous "spike time" representation of spikes in our framework is related to temporal coding, but (Mostafa, 2016) restrict neurons to 1 spike time; our method imposes no such restrictions.

As mentioned in (Wunderlich and Pehle, 2021), applying methods from optimal control theory to compute exact gradients in hard-threshold spiking neural networks has been recognized (Selvaratnam et al., 2000; Kuroe and Ueyama, 2010; Kuroe and Iima, 2006). However, unlike in our setting these works consider a neuron with a two-sided threshold and provide specialized algorithms for specific loss functions. Most related to our work is the recent EventProp (Wunderlich and Pehle, 2021) which derives an algorithm for a continuous-time spiking neural network by applying the adjoint method (which can be seen as generalized backpropagation) together with proper partial derivative jumps. EventProp calculates the gradients by accumulating adjoint variables while computing adjoint state trajectories via simulating another continuous-time dynamical system with transition jumps in a backward pass, but our algorithm computes gradients with just firing time and causality information. In particular, the only time we need to simulate continuous-time dynamics is in the forward pass.

## 2 Spiking Neural Networks

We first describe the precise models we use throughout the paper for the pre-synaptic and post-synaptic behaviors of spiking neurons. We then explain the dynamics of a SNN and the effects of spike generations.

### 2.1 Pre-Synaptic Model

For the ease of presentation, a generic structure of a SNN is illustrated in Fig. 1 on the left. There are many different models to simulate the nonlinear dynamics of a spiking neuron (e.g., see Gerstner et al. (2014)). In this paper, we adopt the Leaky-Integrate-and-Fire (LIF) model which consists of three main steps.

**(i) Synaptic Dynamics.** A generic neuron $i$ is stimulated through a collection of input neurons, its neighborhood $\mathcal{N}_i$. Each neuron $j \in \mathcal{N}_i$ has a synaptic connection to $i$ whose dynamics is modelled by a $1^{\text{st}}$-order low-pass $RC$ circuit that smooths out the Dirac Delta currents it receives from neuron $j$. Since this system is linear and time-invariant (LTI), it can be described by its impulse response

$$h_j^s(t) = e^{-\alpha_j t} u(t),$$

where $\alpha_j = \frac{1}{\tau_j^s}$ and $\tau_j^s = R_j^s C_j^s$ denotes the synaptic time constant of neuron $j$, and $u(t)$ denotes the Heaviside step function. Therefore, the output synaptic current $I_j(t)$ can

be written as

$$I_j(t) = h_j^s(t) \star \sum_{f \in \mathcal{F}_j} \delta(t-f) = \sum_{f \in \mathcal{F}_j} h_j^s(t-f), \quad (1)$$

where $\mathcal{F}_j$ is the set of output firing times from neuron $j$. Note that in Eq. (1) we used the fact that convolution with a Dirac Delta function $h_j^s(t) \star \delta(t-f) = h_j^s(t-f)$, is equivalent to shifts in time.

**(ii) Neuron Dynamics.** The synaptic current of all stimulating neurons is weighted by $W_{ji}$, $j \in \mathcal{N}_i$, and builds the weighted current that feeds the neuron. The dynamic of the neuron can be described by yet another 1st-order low-pass $RC$ circuit with a time constant $\tau_i^n = R_i^n C_i^n$ and with an impulse response

$$h_i^n(t) = e^{-\beta_i t} u(t)$$

where $\beta_i = \frac{1}{\tau_i^n}$. The output of this system is the potential $V_i(t)$.

**(iii) Hard-thresholding and spike generation.** The membrane potential $V_i(t)$ is compared with the firing threshold $\theta_i$ of neuron $i$ and a spike (a Delta current) is produced by neuron when $V_i(t)$ goes above $\theta_i$. Also, after spike generation, the membrane potential is reset/dropped immediately by $\theta_i$.

### 2.2 Post-Synaptic Kernel Model

We call the model illustrated in the left of Fig. 1 the pre-synaptic model, as the spiking dynamics of the stimulating neurons $\mathcal{N}_i$ of a generic neuron $i$ appear before the synapse. In this paper, we will work with a modified but equivalent model in which we combine the synaptic and neuron dynamics, and consider the effect of spiking dynamics directly on the membrane potential after it is smoothed out by the synapse and neuron low-pass filters. We call this the post-synaptic/kernel model of SNN.

To derive this model, we simply use the fact that the only source of non-linearity in SNN is hard-thresholding during the spike generation. In particular, SNN dynamics from the stimulating neuron $j \in \mathcal{N}_i$ until the membrane potential $V_i(t)$ is completely linear and can be described by the joint impulse response

$$\begin{aligned} h_{ji}(t) = h_j^s(t) \star h_i^n(t) &= \int_{-\infty}^{\infty} h_j^s(\tau) h_i^n(t-\tau) d\tau \\ &= \int_0^t e^{-\alpha_j \tau} e^{-\beta_i (t-\tau)} d\tau = \frac{e^{-\alpha_j t} - e^{-\beta_i t}}{\beta_i - \alpha_j} u(t). \end{aligned} \quad (2)$$

Therefore the whole effect of spikes $\mathcal{F}_j$ of neurons $j \in \mathcal{N}_i$ on the membrane potential can be written in terms of the kernel

$$y_{ji}(t) = \sum_{f \in \mathcal{F}_j} h_{ji}(t-f). \quad (3)$$

We call this model post-synaptic since the effect of dynamic of neuron $j \in \mathcal{N}_i$ on $V_i(t)$ is considered after being processed by the synapse and even the neuron $i$. Using the linearity and applying super-position for linear systems, we can see that the effect of all spikes coming for all stimulating neurons $\mathcal{N}_i$, can be written as

$$V_i^\circ(t) = \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(t), \quad (4)$$

where $W_{ji}$ is the weight from neuron $j$ to $i$. We used $V_i^\circ(t)$ to denote the contribution to the membrane potential $V_i(t)$ after neglecting the potential reset due to hard-thresholding and spike generation. Fig. 1 (right) illustrates the post-synaptic model for the SNN.

**Remark 1.** Our motivation for using this equivalent model comes from the fact that even though the spikes are not differentiable functions, the effect of each stimulating neuron $j \in \mathcal{N}_i$ on neuron $i$ is written as a well-defined and (almost everywhere) differentiable kernel. $\lozenge$

**Remark 2** (Connection with the surrogate gradients)**.** Intuitively speaking, and as we will show rigorously in the following sections, the kernel model derived here immediately shows that SNNs have an intrinsic smoothing mechanism for their abrupt spiking inputs, through the low-pass impulse response $h_{ji}(t)$ between their neurons. As a result, one does not need to introduce any additional artificial smoothing to derive surrogate gradients by modifying the neuron model in the backward gradient computation path. We will use this inherent smoothing to prove that SNNs indeed have well-defined gradients. Interestingly, our derivation of the exact gradient based on this inherent smoothing property intuitively explains that even though surrogate gradients are not exact, they may be close to and yield a similar training performance as the exact gradients. $\lozenge$

### 2.3 SNN Full Dynamics

In the post-synaptic kernel model, we specified the effect of spikes from stimulating neurons in (4). To have a full picture of the SNN dynamics, we need to specify also the effect of spike generation. The following completes this.

**Theorem 1.** Let $i$ be a generic neuron in SNN and let $\mathcal{N}_i$ be the set of its stimulating neurons. Let $h_i^n(t)$ and $h_j^s(t)$ be the impulse response of the neuron $i$ and synapse $j \in \mathcal{N}_i$, respectively, and let $h_{ji}(t) = h_i^n(t) \star h_j^s(t)$ as in (2). Then the membrane potential of the neuron $i$ for all times $t$ is given by

$$V_i(t) = V_i^\circ(t) - \sum_{f \in \mathcal{F}_i} \theta_i h_i^n(t-f), \quad (5)$$

where $y_{ji}(t) = \sum_{g \in \mathcal{F}_j} h_{ji}(t-g)$ denotes the smoothed kernel between the neuron $i$ and $j \in \mathcal{N}_i$, and $\theta_i$ denotes the spike generation threshold of the neuron $i$. $\square$

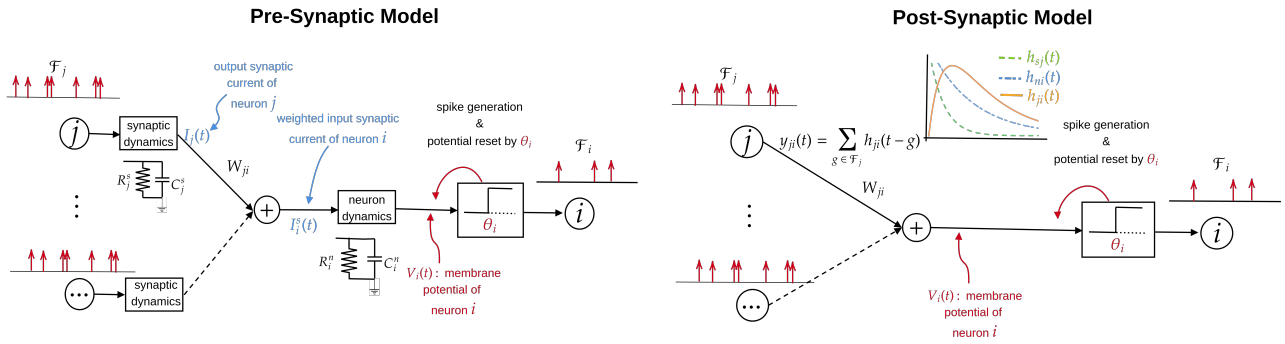



Figure 1: **(Left)** Generic structure of SNN: (i) spikes $\mathcal{F}_j$ coming from input neuron $j$ pass through the synaptic RC circuit with time constant $\tau_j^s = R_j^s C_j^s$ and build synaptic current $I_j(t)$, (ii) synaptic current $I_j(t)$ are weighted by $W_{ji}$ and build the input current $\sum_j W_{ji} I_j(t)$, (iii) this current is filtered through neuron $i$ as an RC circuit with time constant $\tau_i^n = R_i^n C_i^n$ and produces membrane potential $V_i(t)$, (iv) potential $V_i(t)$ is compared with the threshold $\theta_i$, producing a spike when it passes above $\theta_i$, then (v) $V_i(t)$ is dropped by $\theta_i$ immediately after spike generation. **(Right)** Post-synaptic kernel model of SNN. In this model neuron $j \in \mathcal{N}_i$ stimulates neuron $i$ through the smooth kernel $y_{ji}(t) = \sum_{g \in \mathcal{F}_i} h_{ji}(t-g)$ rather than the abrupt spiking signal $\sum_{g \in \mathcal{F}_j} \delta(t-g)$. Thm. 1 shows equivalence of both models in computing membrane potentials.

It is also worth noting here that through Sec. 2.1 and equations (2), (3), (4), and the above (5), we can describe the membrane potential as a weighted sum (and difference) of exponentials, for which it is easy to compute partial derivatives (which show up in (10)). We provide an intuitive proof here, but offer a more rigorous proof in A.1.

*Proof.* We use the following simple result/computation-trick from circuit theory that in an RC circuit, abrupt dropping of the potential of the capacitor by $\theta_i$ at a specific firing time $f \in \mathcal{F}_i$ can be mimicked by adding a voltage source $-\theta_i u(t-f)$ series with the capacitor. If we do this for all the firing times of the neuron, we obtain a linear RC circuit with two inputs: (i) weighted synaptic current coming from the neurons $\mathcal{N}_i$, (ii) voltage sources $\{-\theta_i u(t-f) : f \in \mathcal{F}_i\}$. (See Fig. 2.)

The key observation is that although this new circuit is obtained after running the dynamics of the neuron and observing its firing times $\mathcal{F}_i$, as far as the membrane potential $V_i(t)$ is concerned, the two circuits are equivalent. Interestingly, after this modification, the new circuit is a completely linear circuit and we can apply the super-position principle for linear circuits to write the response of the neuron as the summation of: (i) the response $V_i^{(1)}(t)$ due to the weighted synaptic current $I_i^s(t)$ in the input (as in the previous circuit), and (ii) the response $V_i^{(2)}(t)$ due to Heaviside voltage sources $\{-\theta_i u(t-f) : f \in \mathcal{F}_i\}$. From (4), $V_i^{(1)}(t)$ is simply given by $V_i^{(1)}(t) = \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(t)$.

The response of an RC circuit to a Heaviside voltage function $-\theta_i u(t-f)$ is given by $-\theta_i h_i^n(t-f)$ where $h_i^n(t)$ is the impulse response of the neuron $i$ as before. We also used the time invariance property (for shift by $f$) and a well-known result from circuit theory (Thevenin-Norton

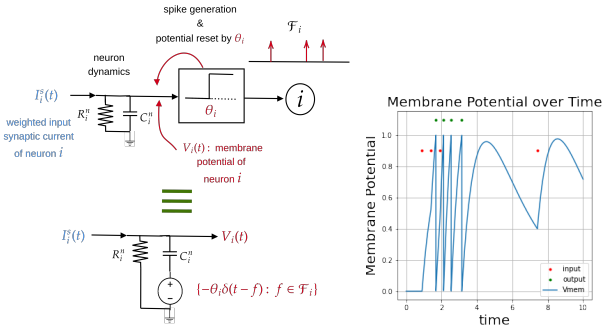



Figure 2: **(Left)** Equivalence of response for: (i) a nonlinear neuron with weighted synaptic currents $I(t)$ and spike generation, and (ii) a linear neuron with input $I(t)$ and Heaviside voltages $\{-\theta_i u(t-f) : f \in \mathcal{F}_i\}$. **(Right)** Example membrane potential over time using Eq. (5)

theorem) that for an RC circuit the impulse response due to a Delta current source is the same as the impulse response due to a Heaviside voltage source. The response to all Heaviside voltage functions, from super-position principle, is simply given by $V_i^{(2)}(t) = -\theta_i \sum_{f \in \mathcal{F}_i} h_i^n(t-f)$. Therefore, we obtain that

$$V_i(t) = V_i^{(1)}(t) + V_i^{(2)}(t) = \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(t) - \sum_{f \in \mathcal{F}_i} \theta_i h_i^n(t-f). \quad (6)$$

This completes the proof. □

# 3 Exact Gradient Computation via Implicit Function Theorem

The Implicit Function Theorem (IFT) will be our main tool for proving the existence of gradients for SNNs. Full statement of the theorem and examples are given in A.2. The reason we need to use IFT lies in the fact that the output of a neuron $i$, which is a spike generated at time $t$, is described by the *implicit* function $V_i(t) = \theta_i$, whereas in usual ANN the input-output relationship is explicit, e.g., $f(\text{input}) = \text{output}$.

## 3.1 Loss Formulation in SNNs

In most settings, we feed the network with an input signal consisting of a collection of spikes within a given time interval and record the firing times $\mathcal{F} = \sqcup_i \mathcal{F}_i$ (disjoint union) of all the spikes produced by neurons $i$ and also the potential of the output layer $V_o(t)$. Here, we consider a quite generic loss function of the form

$$\mathcal{L} = \ell_{\mathcal{F}}(\mathcal{F}; W) + \int_0^T \ell_V(V_o(t), \mathcal{F}; W)\text{dt}, \qquad (7)$$

where $\ell_{\mathcal{F}}$ and $\ell_V$ are assumed to be differentiable functions of all their arguments, with $\ell_{\mathcal{F}}$ the part of the loss that depends on firing times $\mathcal{F}$, and $\ell_V$ the part that depends on membrane potential at the output layer, respectively. Note the second term $\ell_V(V_o(t), \mathcal{F}; W)$ is typically relevant in regression tasks, where in those cases we always assume that the output layer is linear without any firing and potential reset. The first term, in contrast, typically happens in classification tasks.

**Theorem 2.** Let $\mathcal{L}$ be the generic loss function as defined before in (7). Then,

(i) loss $\mathcal{L}$ depends only on the spike firing times $\mathcal{F}$ and the weights $W$, i.e., $\mathcal{L} = \mathcal{L}(\mathcal{F}, W)$,

(ii) $\mathcal{L}(\mathcal{F}, W)$ is a differentiable function of $\mathcal{F}$ and $W$ if $\ell_V(V_o(t), \mathcal{F}; W)$ and $\ell_{\mathcal{F}}(\mathcal{F}; W)$ are differentiable functions of all their arguments $(V_o(t), \mathcal{F}; W)$,

(iii) loss $\mathcal{L}$ has well-defined gradients w.r.t. the weights $W$ if the spike firing times $\mathcal{F}$ are differentiable w.r.t. the weights $W$.

*Proof.* (i) Note that in our post-synaptic kernel model derived in Section 2.2, the membrane potential of the output layer $V_o(t)$ can be written (in a more expanded form) as

$$V_o(t) = \sum_{j \in \mathcal{N}_o} W_{jo} \sum_{g \in \mathcal{F}_j} h_{jo}(t - g). \qquad (8)$$

Note that we dropped the term $-\theta_o \sum_{f \in \mathcal{F}_o} h_o^n(t - f)$ due to potential reset because we always assume that the output neuron is linear in regression tasks where $V_o(t)$ appears directly in the loss. It is also seen that $V_o(t)$ at each time $t$ is a function of all the firing times $\mathcal{F}$ and also weights $W$.

(ii) Since $\ell_{\mathcal{F}}$ is assumed to be a differentiable function of $\mathcal{F}$ and $W$, we need to verify only the differentiability of the integral expression in (7). First note that $h_{jo}(t)$ is a differentiable function except at $t = 0$ where, albeit being non-differentiable, it has finite left and right derivatives. This implies that $V_o(t)$ in (8) is differentiable at all $t$ except at the firing times of its stimulating neuron $\mathcal{N}_o$, where at those points it has finite left and right derivatives. Therefore, we may write

$$\frac{\partial}{\partial \mathcal{F}} \int_0^T \ell_V(V_o(t), \mathcal{F}; W)\text{dt} = \int_0^T \frac{\partial \ell_V}{\partial V_o}(V_o(t), \mathcal{F}; W) \frac{\partial V_o(t)}{\partial \mathcal{F}} + \int_0^T \frac{\partial \ell_V}{\partial \mathcal{F}}(V_o(t), \mathcal{F}; W)\text{dt}.$$

Since $\ell_V$ is assumed to be a differentiable function of $\mathcal{F}$, the second integral is well-defined. Also, $\ell_V$ is differentiable w.r.t. $V_o$. And $V_o(t)$, being a weighted combination of terms $h_{ji}(t - g)$ with $g \in \sqcup_{j \in \mathcal{N}_o} \mathcal{F}_j$, is a differentiable function of firing times $\mathcal{F}$ except perhaps at finitely many points $t \in \sqcup_{j \in \mathcal{N}_o} \mathcal{F}_j$ where at those points it may be discontinuous but has finite left and right derivatives. This implies that the first integral is also well-defined.

(iii) Since from (ii), the loss $\mathcal{L} = \mathcal{L}(\mathcal{F}; W)$ is a differentiable function of both $\mathcal{F}$ and $W$, we have that

$$\frac{\partial \mathcal{L}}{\partial W} = \mathcal{L}_1 \frac{\partial \mathcal{F}}{\partial W} + \mathcal{L}_2$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ denote the partial derivative of $\mathcal{L}$ w.r.t. its 1st and 2nd argument, and where we used the fact that from (ii) both $\mathcal{L}_1$ and $\mathcal{L}_2$ are well-defined. It is seen that the gradients of loss w.r.t. $W$ exist provided that the firing times $\mathcal{F}$ are differentiable w.r.t. the weights. This completes the proof. □

Theorem 2 implies that to prove the existence of the gradients w.r.t. to the weights, which is needed for training the SNN, it is sufficient to prove that the firing times $\mathcal{F}$ are differentiable w.r.t. the weights $W$. We will prove this in the next section by applying the IFT.

## 3.2 Differentiability of Firing Times w.r.t. Weights

Let us consider a generic neuron $i$ and let us write the set of equations for firing times of $i$ by using (5) as:

$$V_i(f) = \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(f) - \theta_i \sum_{m < f} h_i^n(f - m) - \theta_i = 0, \qquad (9)$$

where with some abuse of notation we use $f$ both for the firing time and its label $(i, f) \in \mathcal{F} = \sqcup_l \mathcal{F}_l$. We can write the equations for all the firing times as $\mathbb{V}(\mathcal{F}, W) = \mathbf{0}$ where $\mathbb{V} : \mathbb{R}^{\mathsf{F}} \times \mathbb{R}^{\mathsf{W}} \to \mathbb{R}^{\mathsf{F}}$ is the nonlinear mapping connecting the $\mathsf{F}$ firing times and $\mathsf{W}$ weight parameters.

**Theorem 3.** Let $\mathbf{P}$ be a permutation matrix sorting the firing times in $\mathcal{F}$ in an ascending order. Then, $\frac{\partial \mathbb{V}}{\partial \mathcal{F}} = \mathbf{P}^T \mathbf{L} \mathbf{P}$ where $\mathbf{L}$ is an $\mathsf{F} \times \mathsf{F}$ lower triangular matrix. Moreover, $\mathbf{L}$ has strictly positive diagonal elements $\mathbf{L}_{kk} > 0$ for all $k = 1, 2, \ldots, \mathsf{F}$.

Proof provided in A.3. Informally, this is since the corresponding equation for firing time $f$ can only have contributions from firing times $g < f$. By sorting the firing time equations in ascending time order, this results in a lower triangular structure for partial derivatives w.r.t. firing times. The potential $V_i(t)$ when it fires at $t = f$ should have a positive derivative, since the potential needs to increase to surpass the firing threshold.

Using Theorem 3, we can now prove that the conditions of IFT (Theorem 4) are always fulfilled for firing time equations. This give us explicit formulas for the gradients of the network firing times w.r.t. network weights (Theorem 5). This is summarized in the following.

**Theorem 4.** Let $\mathbb{V}(\mathcal{F}, W) = \mathbf{0}$ be the set of equations corresponding to the firing times. Then the $\mathsf{F} \times \mathsf{F}$ Jacobian matrix $\frac{\partial \mathbb{V}}{\partial \mathcal{F}}$ is non-singular. Moreover, the firing times $\mathcal{F}$ can be written as a differentiable function of the weights $W$.

*Proof.* The first part result follows from Theorem 3:

$$\begin{aligned}\det\left(\frac{\partial \mathbb{V}}{\partial \mathcal{F}}\right) &= \det(\mathbf{P}^T \mathbf{L} \mathbf{P}) = \det(\mathbf{P}) \det(\mathbf{L}) \det(\mathbf{P}^T) \\ &= \det(\mathbf{L}) = \prod_k \mathbf{L}_{kk} > 0,\end{aligned}$$

where we used the fact that $\det(\mathbf{P}) = 1$ for any permutation matrix $\mathbf{P}$. The second part follows from Implicit Function Theorem: $\mathbb{V}(\mathcal{F}, W)$ is a differentiable function of the firing times and weights and $\frac{\partial \mathbb{V}}{\partial \mathcal{F}}$ is non-singular, thus, firing times $\mathcal{F}$ can be written as a differentiable function of the weights. □

**Remark 3.** Using Theorem 3 and 4 and applying the IFT, we have that $\frac{\partial \mathbb{V}}{\partial \mathcal{F}} \times \frac{\partial \mathcal{F}}{\partial W} = -\frac{\partial \mathbb{V}}{\partial W}$. After suitable sorting of the firing times $\mathcal{F}$ (thus, setting the required permutation matrix $\mathbf{P}$ to the identity matrix), this can be written as

$$\mathbf{L} \frac{\partial \mathcal{F}}{\partial W} = -\frac{\partial \mathbb{V}}{\partial W}, \tag{10}$$

where $\mathbf{L}$ is a lower diagonal matrix. As a result, one can solve for the derivatives $\frac{\partial \mathcal{F}}{\partial W}$ recursively, so no matrix inversion is needed. ◊

**Remark 4.** Our results hold for both feed-forward and recurrent networks since it is derived using only the causality relation between the firing times. ◊

**Remark 5.** The matrix $\frac{\partial \mathbb{V}}{\partial W}$ depends only on the values of kernels at the firing times. More specifically, let $f$ be a firing times of neuron $i$ and let $j \in \mathcal{N}_i$ be one of the feeding neurons of neuron $i$. Then, $\frac{\partial \mathbb{V}(f)}{\partial W_{ji}} = y_{ji}(f)$. Moreover, $\frac{\partial \mathbb{V}(f)}{\partial W_{kl}} = 0$ if $l \neq i$ or $k \notin \mathcal{N}_i$. ◊

**Theorem 5.** (Existence of gradients w.r.t. weights) Let $\mathcal{L}$ be a generic loss function for training a SNN as in (7) with $\ell_V$ and $\ell_{\mathcal{F}}$ being differentiable w.r.t. their arguments. Then, $\mathcal{L}$ has well-defined gradients w.r.t. weights.

*Proof.* From Theorem 2, $\mathcal{L}$ has well-defined gradients w.r.t. weights if the firing times $\mathcal{F}$ are differentiable w.r.t. weights, which follows from Theorem 4 by applying the IFT. This completes the proof. □

**Generalization.** We presented our results in the context of exponential kernels (to be able to compare with (Wunderlich and Pehle, 2021)) where we showed that the response of the neuron membrane potential to the input and output spikes can be represented with the exponential feeding and refractory kernels $h_{ji}(t) = \frac{e^{-\alpha_j t} - e^{-\beta_i t}}{\beta_i - \alpha_j} u(t)$ and $-\theta_i h_i(t) = -\theta_i e^{-\alpha_j t} u(t)$. The more generic model for the neuron is the Spike Response Model (SRM) by (Gerstner, 1995) where the membrane potential and output spikes can be written as

$$V_i(t) = K_i^{in}(t) \star \sum_{j \in \mathcal{N}_i} W_{ji} s_j^{in}(t) + K_i^{ref}(t) \star s_i^{out}(t),$$
$$s_i^{out}(t) = u(V_i(t) - \theta_i)$$

where $s_j^{in}(t)$ and $s_i^{out}(t)$ denote the input and output spikes and where $\theta_i$ is the firing threshold. Our method based on IFT is still applicable as far as $K_i^{in}(t)$ and $K_i^{ref}(t)$ are differentiable functions. Also, we need the additional condition that $K_i^{in}(0^+) = 0$ to avoid sudden jumps due to the input spikes so that we can still write the membrane potential at any firing time $f$ as the equality condition

$$\begin{aligned}V_i(f) = \sum_{j \in \mathcal{N}} W_{ji} \sum_{g \in \mathcal{F}_j} K_i^{in}(f - g) \\ - \sum_{e \in \mathcal{F}_i : e < f} K_i^{ref}(f - e) = \theta_i.\end{aligned} \tag{11}$$

These conditions are definitely satisfied for $K_j^{in}(t) = h_{ji}(t)$ and $K_i^{out}(t) = -\theta_i h_i(t)$. By applying the IFT to the differentiable equations (11) corresponding to all the spike firing times, we can find the gradient of the firing times w.r.t. to the weight parameters.

## 4 Implementation

Results from Section 3 also prescribe a natural algorithm.

### 4.1 Causality Graph

By Eq. (5), calculating the membrane potential at any given time just relies on keeping track of causal inputs from the previous (feeding) neuron(s). To efficiently calculate partial derivatives, we will keep track of this information while calculating network outputs. A detailed explanation on a small example is given in A.4.

**Algorithm 1** Firing Time Computation

**Input:** Firing times $\mathcal{F} = \sqcup_j \mathcal{F}_j$ from neighbors $j \in \mathcal{N}_i$ and weights $W_{ji}$. Hyperparameters $\alpha, \beta, \theta_i$.
Initialize $A, B = 0, t_{ref} = 0$.
Initialize empty queue.
**for** $f$ (sorted) **in** $\mathcal{F}$ (where $f$ from neighbor $j$) **do**
- Append $f$ to queue.
- Update $A \leftarrow A \cdot e^{-\alpha(f-t_{ref})} + W_{ji}/(\beta-\alpha)$ and $B \leftarrow B \cdot e^{-\beta(f-t_{ref})} + W_{ji}/(\alpha-\beta)$.
- Update $t_{ref} \leftarrow f$.
- Solve for $t$: $Ae^{-\alpha t} + Be^{-\beta t} = \theta_i$. Add $t$ to output firing times.
- Update $A \leftarrow A \cdot e^{-\alpha(t-t_{ref})}$ and $B \leftarrow e^{-\beta(t-t_{ref})} - \theta_i$.
- Update $t_{ref} \leftarrow t$.

Add entire queue as causal edges to $t$.
**end for**
**Return** Causal graph and firing times.

### 4.2 Forward spike time computation

Simulating an SNN in the forward pass and computing the firing times of its neurons requires solving the Euler integration corresponding to the differential equation of the synapse and membrane potentials. This is usually done approximately by quantizing time into small steps and iteratively updating potentials. There are several libraries such as snnTorch by (Eshraghian et al., 2021a) that implement this. Our gradient computation can also use these methods where the firing times are computed.

Here, we propose another method that uses the impulse response (kernel) representation of the corresponding differential equations derived in (2) and (5) to compute the firing times exactly without any need for time quantization. The main idea behind this method is that for exponential synaptic and membrane impulse responses, one can always write the membrane potential of a neuron over a time interval $[t_0, t_1]$ at which the neuron receives no spikes at its input as $Ae^{-\alpha t} + Be^{-\beta t}$ where $A$ and $B$ are some suitable coefficients and where $\alpha, \beta$ are the inverse synaptic and membrane time constants (common to all neurons), respectively. Thus the next firing time can be found by computing the time $t$, if there is any, at which this curve intersects the horizontal line $\theta$. Once this firing time is computed, we update $A$, $B$ and the search interval $[t_0, t_1]$ depending on whether the neuron receives any spikes before this firing time, and so on. This is summarzied in Algorithm 1.

**Remark 6.** Note that one can calculate partial derivatives immediately after solving for the firing time and computing the causality graph. In feed-forward networks, these calculations for neurons in the same layer can be done in parallel since the firing times of neurons in the same layer will not affect each other.

### 4.3 Forward propagation for gradient computation

The forward propagation algorithm (Algorithm 2) emerges from Theorems 3 and 4. We can calculate partial derivatives of the loss after calculating the partial derivatives of the network firing times w.r.t. network weights, which are in turn calculated by applying the implicit function theorem with appropriate partial derivatives of the firing time equations.

**Algorithm 2** Forward Propagation

**Input:** Network output firing times $\mathcal{F} = \sqcup_i \mathcal{F}_i$ for all $i$ and causal graph (e.g., by Alg. 1). Hyperparameters for network and loss.
Initialize matrices $\mathbf{L}$, $\frac{\partial \mathcal{F}}{\partial W}$, and $\frac{\partial \mathbb{V}}{\partial W}$.
**for** $f$ (sorted) **in** $\mathcal{F}$ **do**
Calculate partial derivatives of the firing time equation for $f$ output by neuron $i$: $V_i(f) - \theta_i = 0$.
- Use causal information and Equation (5) to fully describe $V_i(f)$.
- **Update L.** Calculate $\frac{\partial}{\partial f_{j\to i}}(V_i(f) - \theta_i)$ for each $f_{j\to i}$ in the causal graph for $f$.
- **Update L.** Calculate $\frac{\partial}{\partial f}(V_i(f) - \theta_i)$.
- **Update $\frac{\partial \mathbb{V}}{\partial W}$.** Calculate $\frac{\partial}{\partial W_{ji}}(V_i(f) - \theta_i)$ for all weights $W_{ji}$ attached to neuron $i$.
- **IFT Step.** Solve Equation (10) via back substitution to update $\frac{\partial \mathcal{F}}{\partial W}$.

**end for**
Calculate $\frac{\partial \mathcal{L}}{\partial W}$ using final $\frac{\partial \mathcal{F}}{\partial W}$ via Eq. (10).

Again, due to the lower triangular structure of matrix $\mathbf{L}$ (Theorem 3), we can iteratively solve the linear system (10) of IFT equations without a full matrix inversion. This costs $O(|\mathcal{F}|^2|\mathcal{W}|)$ in time, using $(1 + 2 + 3 + ... + |\mathcal{F}|) \times$ ( up to $|\mathcal{W}|$) operations to solve for the $|\mathcal{F}| \times |\mathcal{W}|$ Jaccobian matrix. The memory cost is $O(|\mathcal{F}||\mathcal{W}|)$ to store the solutions and one of the Jacobians, where $O(|\mathcal{F}||\mathcal{W}|)$ is always needed for storing the gradients.

## 5 Simulation

### 5.1 XOR Task

To investigate whether the network can robustly learn to solve the XOR task as in (Mostafa, 2016), we reproduced

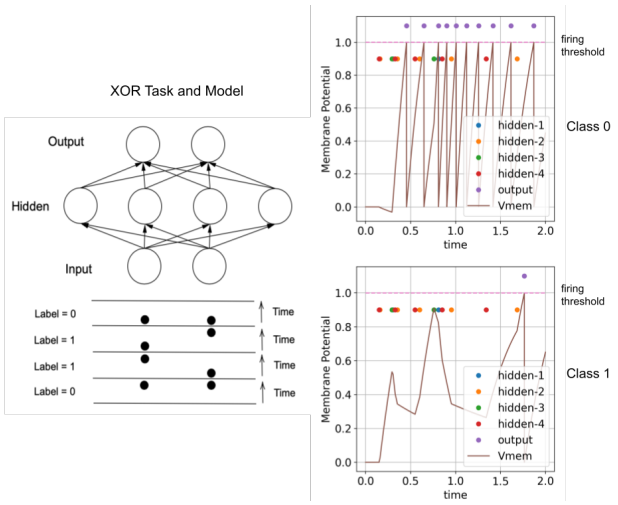



Figure 3: **(Left)** Model for XOR task. **(Right)** Given the input $(0, 0)$, output neurons have different voltage traces. Note that each output neuron has the same input firing times, from each of the 4 hidden layer neurons, but the network is able to learn weights that push the output neuron corresponding to label '1' to spike later, and the one corresponding to label '0' (true label) to spike earlier.

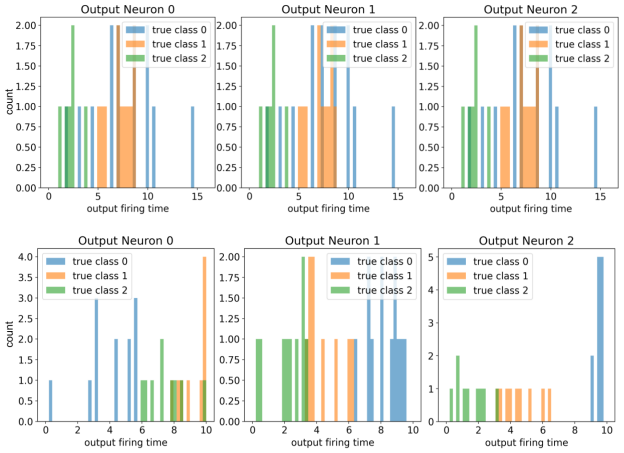



Figure 4: A histogram of the first output firing times of each label neuron, given unseen test data. **(Top)** At random initialization, firing times look the same across all label neurons. **(Bottom)** After training, the firing times are clearly separated into the 3 classes, and all test examples belonging to the same class as the corresponding label neuron fires earlier than in the other label neurons.

most experiment settings by coding each of the input spikes as 0.0 (early spike) or 2.0 (late spike), which feed into 4 hidden neurons, which in turn feed into 2 output neurons. We use a cross-entropy loss based on first spike times of the output neurons (so the label neuron should fire sooner than the other). For each of 1000 different random weight initializations, we trained until convergence with learning rate 0.1. Unlike in (Mostafa, 2016), we consider one iteration of training to be just 1 full batch, rather than 100. Across all 1000 trials, the maximum steps to converge was 98, with the average being 17.52 steps. Compare this to maximum 61 training iterations (each iteration seeing 100 full batches of the four input patterns), with average 3.48 iterations in (Mostafa, 2016). Fig. 3 illustrates the model implementing the XOR task, as well as a post-training simulation of membrane potentials for input $(0, 0)$.

### 5.2 Iris Dataset

We also trained SNN using FP on the Iris dataset to demonstrate learning from data with real-valued features. Note one class is linearly separable from the other 2; the latter are not linearly separable from each other (Anderson, 1936; Fisher, 1936). We encoded the input features with a scheme similar to (Liu et al., 2017), but modified to where each real-valued feature $n_i$ is transformed into a firing time via the transformation $T \cdot (1 - \frac{n_i - \min(n_i)}{\max(n_i) - \min(n_i)})$, where $T$ is the maximum time horizon and the min/max of a feature is taken over the whole dataset. After training a small 4-10-3 network, we achieve 100% test accuracy (compare to 93.3% for MT-1 (4-25-1) and 96.7% for an MLP ANN (4-25-3) in (Liu et al., 2017)). Again, the network is able to learn weights to push the true label output neurons to fire earlier than the others, since our loss function is minimized when all the correct label neurons fire before other output neurons. An illustration of this effect is shown in Fig. 4.

### 5.3 Yin-Yang Dataset

We also implemented FP to train SNN on the Yin-Yang dataset which is a two-dimensional and non-linearly separable dataset (Kriener et al., 2022). The Yin-Yang dataset requires a multi-layer model, as a shallow classifier achieves around 64% accuracy, thus it requires a hidden layer and backpropagation (or forward-propagation in our case) for gradient-based learning to achieve higher accuracy, as noted also in (Wunderlich and Pehle, 2021).

We used a loss based on the earliest spike times of the 3 output neurons, as in (Wunderlich and Pehle, 2021; Göltz et al., 2021) defined as

$$\mathcal{L} = -\frac{1}{N_{\text{batch}}} \left[ \sum_{i=1}^{N_{\text{batch}}} \log \left( \frac{e^{-f_{i,l(i)}/\tau_0}}{\sum_{j=1}^{3} e^{-f_{i,j}/\tau_0}} \right) + \gamma \left( e^{f_{i,l(i)}/\tau_1} - 1 \right) \right],$$

where $f_{i,j}$ is the first spike time of neuron $j$ for the $i^{\text{th}}$ example and $l(i)$ is the index of the correct label for the $i^{\text{th}}$ example. The second term is a regularization term which encourages earlier spike times for the true label neuron, its

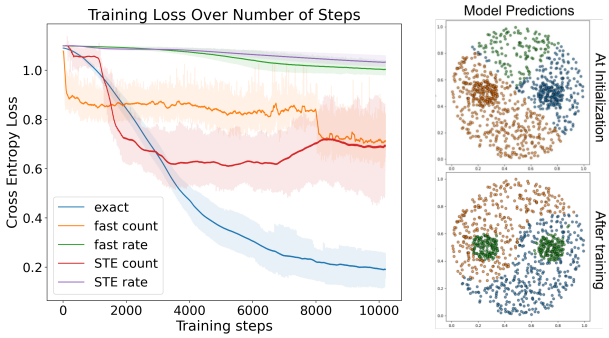



Figure 5: **(Left)** Comparison to surrogate gradients. The plot shows the change in training loss over time for training SNN with exact and surrogate gradients, the fast sigmoid function and straight-through estimator each with a count-based and spike-rate cross entropy loss. **(Right)** A comparison of model predictions at random initialization, versus after training.

influence on the total loss controlled by $\gamma$. Note that $\gamma = 0$ corresponds to usual cross entropy.

**Comparing to surrogate methods.** First, to compare training with surrogate gradient methods, we used the snnTorch library by (Eshraghian et al., 2021a) to train equivalent models[1], using the same hyperparameters and initializations, but with surrogate gradients. Fig. 5 (left) compares training with exact gradient (our method) with using the fast sigmoid (Zenke and Ganguli, 2018) surrogate function and the straight-through estimator (Bengio et al., 2013), with both count-based cross entropy loss and a spike rate cross entropy loss. (See footnote.) All models at initialization have around 30-36% accuracy and cross entropy loss around 1.09-1.1, but at the end of 300 epochs of training, using exact gradients results in faster loss reduction (as one might expect).

**Evaluation.** After repeating the experiment with 10 random initializations, a 2-layer SNN model trained with FP obtains a test accuracy with mean **95.0**(0.83)%, comparable to (Göltz et al., 2021) reporting 95.9(0.7)%. It is worth noting that training only involved using the exact gradients for SGD, without employing other heuristics in (Göltz et al., 2021), which include a flat weight bump (increase weights a fixed amount) whenever the proportion of non-spiking neurons is above a certain threshold, among others.

[1]Many surrogate methods are usually not compatible with training using temporal losses, as noted also by (Eshraghian et al., 2021a) that often the first spike time is non-differentiable with respect to the spikes themselves. To fairly compare to surrogate methods, instead we used both a spike count-based cross entropy loss and a spike rate cross entropy loss. The former calculates cross entropy from the number of spikes emitted by output neurons, and the latter accumulates cross entropy loss at each time step.

These experiments offer a proof of concept that the network is able to learn by using exact gradients. We hope our work will provide a rigorous foundation for improving training libraries for SNNs. Additional details on experiments presented in this section can be found in A.5.

## 6 Discussion

Our framework offers an alternative view of the differentiability of SNN w.r.t. network weights and provides a new algorithm, forward-propagation (FP) to calculate gradients of SNN by accumulating information in the forward pass of the network. Our results apply generally to networks with arbitrary recurrent connections, and the ideas can be generalized to other Spike Response Models (SRM). Our gradient method can be used with other algorithms that can simulate the forward pass dynamics, and the FP algorithm dependence on just the causal graph of firing times allows for self-contained formulas which can be often be computed in parallel, e.g., in feed-forward networks. The operations used to compute gradients via FP are also simple and require solving a lower triangular linear system, which can be done quickly.

An interesting by-product of our framework is the fact that our formulas resemble surrogate gradient methods and Hebbian learning. For instance, (Zenke and Ganguli, 2018) uses the negative half of the sigmoid function to smooth out the discrete spiking behavior. Our framework captures a natural smoothing exponential kernel already present in the exact version. (See Remark 2.) Further, the way the smooth kernels $y_{ij}(t)$ between two neurons $i$ and $j$ that appear in the gradient computation resembles Hebbian learning where if there are more spikes from $i$ to $j$ the kernel $y_{ij}(t)$ becomes larger, thus, causing the gradient w.r.t. the connecting weight $W_{ij}$ to become larger. This has a Hebbian flavor where more firing/activation causes the connecting weight $W_{ij}$ to be rewarded (for negative gradient) or punished (for positive gradients) more strongly. These relationships can be of their own interest.

### Acknowledgements

Amin Karbasi acknowledges funding in direct support of this work from NSF (IIS-1845032), ONR (N00014- 19-1-2406), and the AI Institute for Learning-Enabled Optimization at Scale (TILOS). Jane Lee was partially supported by NSF (IIS-1845032) and a GFSD fellowship sponsored by the NSA.

### References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.-J., Taba, B., Beakes, M., Brezzo, B., Kuang, J. B., Manohar, R., Risk, W. P., Jackson, B., and 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.

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509.

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. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation.

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

Burkitt, A. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95:1–19.

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66.

Choe, Y. (2013). *Hebbian Learning*, pages 1–5. Springer New York, New York, NY.

Comsa, I., Fischbacher, T., Potempa, K., Gesmundo, A., Versari, L., and Alakuijala, J. (2020). Temporal coding in spiking neural networks with alpha synaptic function. pages 8529–8533.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.-H., Wild, A., Yang, Y., and Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Diehl, P. and eCook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9.

Ding, J., Yu, Z., Tian, Y., and Huang, T. (2021). Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *ArXiv*, abs/2105.11654.

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., and Lu, W. D. (2021a). Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*.

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., and Lu, W. D. (2021b). Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*.

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., Berg, D. J., McKinstry, J. L., Melano, T., Barch, D. R., di Nolfo, C., Datta, P., Amir, A., Taba, B., Flickner, M. D., and Modha, D. S. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

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

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

Gidon, A., Zolnik, T. A., Fidzinski, P., Bolduan, F., Papoutsi, A., Poirazi, P., Holtkamp, M., Vida, I., and Larkum, M. E. (2020). Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87.

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., Dold, D., Kungl, A. F., Senn, W., Schemmel, J., Meier, K., and Petrovici, M. A. (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times.

Hinton, G. (2022). The forward-forward algorithm: Some preliminary investigations.

Ho, N.-D. and Chang, I.-J. (2021). Tcl: an ann-to-snn conversion with trainable clipping layers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 793–798.

Huh, D. and Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Jeffares, A., Guo, Q., Stenetorp, P., and Moraitis, T. (2022). Spike-inspired rank coding for fast and accurate recurrent neural networks. In *International Conference on Learning Representations*.

Jin, Y., Zhang, W., and Li, P. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. In Bengio, S., Wallach, H., Larochelle, H.,

Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Kempter, R., Gerstner, W., and van Hemmen, L. (1999). Hebbian learning and spiking neurons. *Phys. Rev. E*, 59.

Kornijcuk, V., Lim, H., Seok, J. Y., Kim, G., Kim, S. K., Kim, I., Choi, B. J., and Jeong, D. S. (2016). Leaky integrate-and-fire neuron circuit based on floating-gate integrator. *Frontiers in Neuroscience*, 10.

Kriener, L., Göltz, J., and Petrovici, M. A. (2022). The yin-yang dataset.

Kuroe, Y. and Iima, H. (2006). A learning method for synthesizing spiking neural oscillators. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 3882–3886.

Kuroe, Y. and Ueyama, T. (2010). Learning methods of recurrent spiking neural networks based on adjoint equations approach. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

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. *Frontiers in Neuroscience*, 12.

Liu, T., Liu, Z., Lin, F., Jin, Y., Quan, G., and Wen, W. (2017). Mt-spike: A multilayer time-based spiking neuromorphic architecture with temporal error backpropagation. pages 450–457.

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. *Frontiers in Neuroscience*, 14.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671.

Moraitis, T., Sebastian, A., and Eleftheriou, E. (2020). Optimality of short-term synaptic plasticity in modelling certain dynamic environments.

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

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *ArXiv*, abs/1901.09948.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9.

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. *Frontiers in Neuroscience*, 14:653.

Pfeiffer, M. and Pfeil, T. (2018). Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12.

Roberts, P. D. (2013). *Synaptic Dynamics: Overview*, pages 1–4. Springer New York.

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

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11.

Ruf, B. and Schmitt, M. (2006). *Hebbian learning in networks of spiking neurons using temporal coding*, pages 380–389.

Safa, A., Catthoor, F., and Gielen, G. G. (2021). Convsnn: A surrogate gradient spiking neural framework for radar gesture recognition. *Software Impacts*, 10:100131.

Selvaratnam, K., Kuroe, Y., and Mori, T. (2000). Learning methods of recurrent spiking neural networks.

Shrestha, S. B. and Orchard, G. (2018). Slayer: Spike layer error reassignment in time. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2018). Deep learning in spiking neural networks. *Neural Networks*.

Wang, X., Lin, X., and Dang, X. (2020). Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks*, 125:258–280.

Wunderlich, T. and Pehle, C. (2021). Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11:12829.

Zenke, F. and Ganguli, S. (2018). SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation*, 30(6):1514–1541.

Zhang, W. and Li, P. (2019). Spike-train level backpropagation for training deep recurrent spiking neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

# A Appendix

## A.1 Alternative Proof of Theorem 1

**Proof (ii)**: Here we provide a more rigorous proof based on induction on the number of firing times $\mathsf{F}_i := |\mathcal{F}_i|$ of the neuron $i$.

We first check the base of the induction. If there are no firing times, i.e., $\mathcal{F}_i = \emptyset$ and $F_i = 0$, then there is no source of non-linearity and the neuron is a fully linear system. Thus, the response of the neuron to the input weighted synaptic current $I_i^s(t)$ is given, as in (4), by

$$V_i(t) = \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(t),$$

which yields the desired result since, for $\mathcal{F}_i = \emptyset$, the second term $-\sum_{f \in \mathcal{F}_i} \theta_i h_i^n(t-f)$ is zero. This confirms the base of induction for $\mathsf{F}_i = 0$.

Now let us assume that $\mathcal{F}_i \neq \emptyset$ and the neuron $i$ has fired at least once ($\mathsf{F}_i \geq 1$). Here, we can still check that result holds for all time $t \in [0, f_1)$ before the first firing time $f_1$ because before the first firing time the circuit is completely linear (thus, the first term) and the second term is equal to zero as $h_i^n(t-f_1) = e^{\beta_i(t-f_1)}u(t-f_1)$ is equal to zero for all $t < f_1$ (due to causality and the fact that $u(t-f_1) = 0$ for $t < f_1$).

Now we prove that if the result is true for $t \in [0, f^k)$ it remains true for $t \in [f_k, f_{k+1})$ where we denote the $k$-th and $(k+1)$-th firing times by $f_k$ and $f_{k+1}$ and apply the convention that $f_k = \infty$ for $k > F_i$.

To prove this, we first note that the weighted synaptic current (see, e.g., Fig. 2) coming from the neurons $\mathcal{N}_i$ is given by

$$I_i^s(t) = \sum_{j \in \mathcal{N}_i} W_{ji} \sum_{g \in \mathcal{F}_i} h_j^s(t-g)$$

for all times $t \geq 0$. Also, note that since synapses are always linear, this is true independent of whether there is any firing and potential drop at the neuron $i$. At the firing time $f_k$ the value of potential drops to $V_i^{(k)} = V_i(f_k) - \theta_i$. Thus, to prove the result, we need to find and verify the response of the neuron to the synaptic current $I_i^s(t)$ for $t \in [f_k, f_{k+1})$ starting from the initial value $V_i^{(k)}$. Here again we note that starting from $f_k$ the system is again linear until the next firing time $f_{k+1}$. Thus, we can again apply the super position principle for linear systems to decompose the response into two parts: (a) response to the initial condition $V_i^{(k)}$ and (b) response to the input synaptic current $I_i^s(t)$.

From the linearity and time-invariance of RC circuits, (a) is simply given by

$$\begin{aligned} V_i^{(a)}(t) &= V_i^{(k)} h_i^n(t-f_k) \\ &= V_i^{(k)} e^{-\beta_i(t-f_k)} u(t-f_k) \\ &= V_i(f_k) e^{-\beta_i(t-f_k)} u(t-f_k) - \theta_i h_i^n(t-f_k), \end{aligned}$$

where $h_i^n(t) = e^{-\beta_i t}u(t)$ is the impulse response of the neuron $i$.

The response to the synaptic current in the time interval $t \in [f_k, f_{k+1})$ is also given by

$$\begin{aligned}
V_i^{(b)}(t) &\overset{(i)}{=} I_i^s(t)u(t-f_k) \star h_i^n(t) \\
&= \int_0^\infty I_i^s(\lambda)u(\lambda-f_k)h_i^n(t-\lambda)d\lambda \\
&\overset{(ii)}{=} \int_{f_k}^t I_i^s(\lambda)h_i^n(t-\lambda)d\lambda \\
&= \int_0^t I_i^s(\lambda)h_i^n(t-\lambda)d\lambda - \int_0^{f_k} I_i^s(\lambda)h_i^n(t-\lambda)d\lambda \\
&= I_i^s(t) \star h_i^n(t) - \int_0^{f_k} I_i^s(\lambda)e^{-\beta_i(t-\lambda)}d\lambda \\
&= I_i^s(t) \star h_i^n(t) - e^{-\beta_i(t-f_k)} \int_0^{f_k} I_i^s(\lambda)e^{-\beta_i(f_k-\lambda)}d\lambda \\
&= I_i^s(t) \star h_i^n(t) - I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} \times e^{-\beta_i(t-f_k)},
\end{aligned}$$

where in $(i)$ we multiplied $I_i^s(t)$ with $u(t-f_k)$ to remove the effect of the synaptic current before $f_k$ (since, due to causality, it cannot affect the neuron potential in the time interval $t \in [f_k, f_{k+1})$), where in $(ii)$ we used the fact that, due to causality, $h_{ni}(t-\lambda) = 0$ for $\lambda > t$, and that $u(\lambda - f_k)$ is zero for $\lambda < f_k$.

From the induction hypothesis applied to $f_k \in [0, f_k]$, we have that

$$\begin{aligned}
V_i(f_k) &= I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} - \theta_i \sum_{l=1}^{k-1} h_i^n(f_k - f_l) \\
&= I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} - \theta_i \sum_{l=1}^{k-1} h_i^n(f_k - f_l) \\
&= I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} - \theta_i \sum_{l=1}^{k-1} e^{-\beta_i(f_k-f_l)} \\
&= I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} - \theta_i e^{\beta_i(t-f_k)} \sum_{l=1}^{k-1} e^{-\beta_i(t-f_l)} \\
&= I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} - \theta_i e^{\beta_i(t-f_k)} \sum_{l=1}^{k-1} h_i^n(t-f_l).
\end{aligned}$$

Therefore, after simplification, we obtain that

$$I_i^s(t) \star h_i^n(t)\Big|_{t=f_k} \times e^{-\beta_i(t-f_k)} \tag{12}$$

$$= V_i(f_k)e^{-\beta_i(t-f_k)} + \theta_i \sum_{l=1}^{k-1} h_i^n(t-f_l). \tag{13}$$

Replacing in (12), therefore, we obtain

$$V_i^{(b)}(t) = I_i^s(t) \star h_i^n(t) \tag{14}$$

$$- V_i(f_k)e^{-\beta_i(t-f_k)} - \theta_i \sum_{l=1}^{k-1} h_i^n(t-f_l). \tag{15}$$

Applying the super position principle, we have

$$\begin{aligned}
V_i(t) &= V_i^{(a)}(t) + V_i^{(b)}(t) \\
&= I_i^s(t) \star h_i^n(t) - \theta_i \sum_{l=1}^{k-1} h_i^n(t - f_l) - \theta_i h_i^n(t - f_k) \\
&= I_i^s(t) \star h_i^n(t) - \theta_i \sum_{l=1}^{k} h_i^n(t - f_l) \\
&= \Big( \sum_{j \in \mathcal{N}_i} W_{ji} \sum_{g \in \mathcal{F}_i} h_j^s(t - g) \Big) \star h_i^n(t) - \theta_i \sum_{l=1}^{k} h_i^n(t - f_l) \\
&= \sum_{j \in \mathcal{N}_i} W_{ji} \sum_{g \in \mathcal{F}_i} h_{ji}(t - g) - \theta_i \sum_{l=1}^{k} h_i^n(t - f_l) \\
&= \sum_{j \in \mathcal{N}_i} W_{ji} y_{ji}(t) - \theta_i \sum_{f \in \mathcal{F}_i} h_i^n(t - f),
\end{aligned}$$

where in the last equation we used the fact that $h_i^n(t - f) = 0$ for $t \in [f_k, f_{k+1})$ and for $f > f_{k+1}$. This validates the result for $t \in [f_k, f_{k+1})$, and verifies the induction. This completes the proof.

### A.2 Implicit Function Theorem

In many problem in machine learning, statistics, control theory, mathematics, etc. we use a collection of variables to track/specify the state of an algorithm, a dynamical system, etc. However, in practice, these variables are not completely free and are connected to each other via specific constraints. In such cases, we are always interested to know the functional relation between these variables, namely, how changing some variables affect the others (sensitivity analysis). IFT theorem provides a rigorous method for these types of analyses when the variables are connected through differentiable equality constraints, as illustrated in the following theorem.

**Theorem 6** (Implicit Function Theorem)**.** Let $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ be a differentiable function and let $\mathcal{Z} = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^m : \phi(x, y) = 0\}$ be the zero-set of $\phi$. Suppose that $\mathcal{Z} \neq \emptyset$ and let $(x_0, y_0) \in \mathcal{Z}$ be an arbitrary point. Also, let $\frac{\partial \phi}{\partial y}\phi(x_0, y_0)$ be the $m \times m$ matrix of partial derivatives w.r.t. $y$ and assume that it is non-singular, i.e., $\det\big(\frac{\partial \phi}{\partial y}(x_0, y_0)\big) \neq 0$. Then,

- There is an open neighborhood $\mathcal{N}_x$ around $x_0$ and an open neighborhood $\mathcal{N}_{\mathrm{y}}$ around $y_0$ such that $\frac{\partial \phi}{\partial y}\phi(x, y)$ is non-singular for all $(x, y) \in \mathcal{N} := \mathcal{N}_{\mathrm{x}} \times \mathcal{N}_{\mathrm{y}}$ (including of course the original $(x_0, y_0)$.
- There is a function $\psi : \mathcal{N}_{\mathrm{x}} \to \mathcal{N}_{\mathrm{y}}$ such that $(x, \psi(x))$ belongs to the zero set $\mathcal{Z}$, namely, $\phi(x, \psi(x)) = 0$, for all $x \in \mathcal{N}_{\mathrm{x}}$; therefore, the variables $y$ in $\mathcal{N}_{\mathrm{y}}$ can be written as a function $y = \psi(x)$ of the variables $x$ in $\mathcal{N}_{\mathrm{x}}$.
- $\psi$ is a differentiable function of $x$ for $x \in \mathcal{N}_{\mathrm{x}}$ and

$$\frac{\partial \phi}{\partial y} \times \frac{\partial \psi}{\partial x} + \frac{\partial \phi}{\partial x} = 0, \tag{16}$$

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

$$\frac{\partial \psi}{\partial x} = -\Big(\frac{\partial \phi}{\partial y}\Big)^{-1} \times \frac{\partial \phi}{\partial x}. \tag{17}$$

**Example 1.** Fig. 6 illustrates the zero-set $\mathcal{Z} = \{(x, y) : \phi(x, y) = 0\}$ of a function $\phi : \mathbb{R}^2 \to \mathbb{R}$. To investigate the conditions of the implicit function theorem, we first note that the gradient of $\phi$ denoted by $\nabla\phi = (\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y})$ is always orthogonal to the level-set (here the zero-set) of $\phi$. Thus, by observing the orthogonal vector to curve, we can verify if $\frac{\partial \phi}{\partial x}$ or $\frac{\partial \phi}{\partial y}$ are non-singular (non-zero in the scalar case we consider here). We investigate several cases:

- Point $C$: gradient vector does not exist, so the assumptions of the IFT are not fulfilled. One can also see that at $C$ one cannot write neither $x$ as a function of $y$ nor $y$ as a function of $x$.

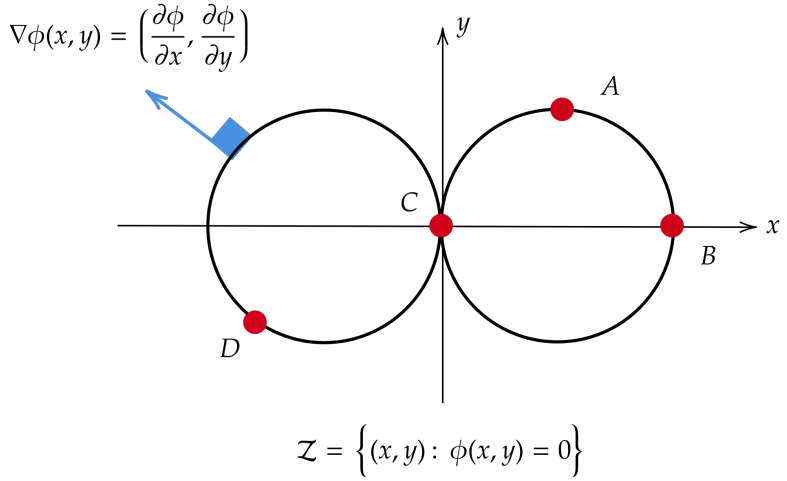



Figure 6: Illustration of the implicit function theorem.

- Point $A$: gradient vector has zero horizontal and non-zero vertical component , i.e., $\frac{\partial\phi}{\partial x} = 0$ and $\frac{\partial\phi}{\partial y} \neq 0$. Thus, from IFT, in a local neighborhood of $A$, one should be able to write only $y$ as a differentiable function of $x$.

- Point $B$: gradient has zero horizontal component. And, only $x$ can be written as differentiable function of $y$.

- Point $D$: gradient has non-zero horizontal and vertical components. So, in a local neighborhood of $D$, one may write both $x$ and $y$ as a differentiable function of the another.

### A.3 Proof of Theorem 3

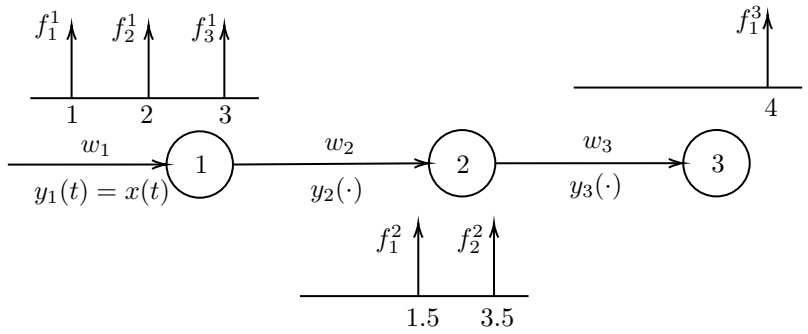
*Proof.* We note that due to causality (future firing times cannot affect past ones), the equation corresponding to a specific firing time $f \in \mathcal{F}$ can only have contribution from firing times less than $f$. In other words, $\frac{\partial V_f}{\partial g} = 0$ for all $g < f$. Letting $\mathbf{P}$ be the permutation matrix sorting the firing times, therefore, the Jaccobian matrix of the sorted firing times given by $\mathbf{P}\frac{\partial \mathbb{V}}{\partial \mathcal{F}}\mathbf{P}^T$ should be a lower triangular matrix $\mathbf{L}$. This yields the first part $\frac{\partial \mathbb{V}}{\partial \mathcal{F}} = \mathbf{P}^T\mathbf{L}\mathbf{P}$. To check the second part, let $k$ be the index of a specific firing time $f$ in the sorted version. Let us denote the neuron corresponding to the firing $f$ by $i$. Then, we have that

$$\mathbf{L}_{kk} = \frac{\partial \mathbb{V}_f}{\partial f} = \frac{d}{df} V_i(f)\Big|_{\text{all other firing times fixed}} = V_i'(t)\Big|_{t=f^-} > 0$$

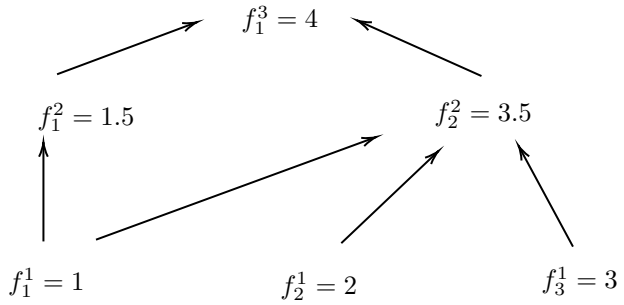
which is equal to the left time derivative the potential $V_i(t)$ when it passes through the threshold $\theta_i$ at time $t = f$. It is worthwhile to mention that that since $V_i(f)$ is a differentiable function of $f$, it has both left and right derivatives and they are equal. However, this derivative is equal to only the left derivative of the potential. Note that this derivative should be strictly positive otherwise the potential will not surpass the firing threshold $\theta_i$ and no firing time will happen. This completes the proof. □

### A.4 Example: Causality and Differentiability

In order to track the effects of previous layers' firing times on a current neuron $i$, we can map which firing times of a previous neuron cause the firing of a neuron that it feeds into, and so on through the network. Consider the following simple example of a simple 3 neuron feed-forward network with 1 input dimension:




For simplicity, we will assume all neurons have the same parameters $\alpha, \beta, \theta$. Let $w_1, w_2, w_3$ be the weights corresponding to the inputs to neurons 1, 2, and 3, respectively. Suppose that neuron 1 had firing times at $f_1^1 = 1, f_2^1 = 2$, and $f_3^1 = 3$. Neuron 2 fired at $f_1^2 = 1.5$ and $f_2^2 = 3.5$. Finally neuron 3 fired at $f_1^3 = 4$. The input $x(t)$ causes neuron 1 to fire. Then note that the only firing times that could cause neuron 2 to fire at $f_1^2 = 1.5$ had to occur before $t = 1.5$. This is only $f_1^1 = 1$. After neuron 2 fires at $f_1^2$, its next firing time $f_2^2 = 3.5$ is affected by $f_1^1$, $f_2^1$ and $f_3^1$. And similarly, $f_1^2$ and $f_2^2$ affects $f_1^3$. This corresponds to the following causality diagram:




The arrows only point up to one level, which allows us to compute the necessary partial derivatives while computing the forward pass for the current layer (i.e., layer by layer). Note that while this simple example is for the reset to zero regime, where the membrane potential resets completely to 0 and all inputs in-between firing times accumulate until the next time the neuron fires, this kind of diagram can similarly be constructed for other regimes. For instance, if there is a time delay before inputs can start increasing the membrane potentials again, to decide the causal edges for a current firing time for a neuron we would have to look for input firing times that occurred at least "time delay" seconds after the current neuron's previous firing time.

We will use equations (2), (5), and (6) to define the following system. Since all neurons share the same parameters $\alpha, \beta$, we can simplify some notation and refer to the joint impulse response coming into a neuron as $h_{s+n}$ which corresponds to equation (2) and the impulse response for just the membrane potential dynamics as $h_n$ which corresponds to the $h_i^n$ term in equation (5). Explicitly,

$$h_{s+n}(t) = \frac{e^{-\alpha t} - e^{-\beta t}}{\beta - \alpha} u(t)$$
$$h_n(t) = e^{-\beta t} u(t)$$

The firing time equations are explicitly given by the following:

$$w_1 \cdot \sum_{t:x(t)=1 \wedge t<f_1^1} h_{s+n}(f_1^1 - t) = \theta \qquad \text{Eq. for } f_1^1$$

$$w_1 \cdot \sum_{t:x(t)=1 \wedge t<f_2^1} h_{s+n}(f_2^1 - t) - \theta \cdot h_n(f_2^1 - f_1^1) = \theta \qquad \text{Eq. for } f_2^1$$

$$w_1 \cdot \sum_{t:x(t)=1 \wedge t<f_3^1} h_{s+n}(f_3^1 - t) - \theta \cdot \big(h_n(f_3^1 - f_1^1) + h_n(f_3^1 - f_2^1)\big) = \theta \qquad \text{Eq. for } f_3^1$$

$$w_2 \cdot h_{s+n}(f_1^2 - f_1^1) = \theta \qquad \text{Eq. for } f_1^2$$

$$w_2 \cdot \big(h_{s+n}(f_2^2 - f_1^1) + h_{s+n}(f_2^2 - f_2^1) + h_{s+n}(f_2^2 - f_3^1)\big) - \theta \cdot h_n(f_2^2 - f_1^2) = \theta \qquad \text{Eq. for } f_2^2$$

$$w_3 \cdot \big(h_{s+n}(f_1^3 - f_1^2) + h_{s+n}(f_1^3 - f_2^2)\big) = \theta \qquad \text{Eq. for } f_1^3$$

Now, all 6 equations are equations of the network weights $(w_1, w_2, w_3)$ and the 6 firing times $(f_1^1, f_2^1, f_3^1, f_1^2, f_2^2, f_1^3)$. Here, we invoke the implicit function theorem which will allow us to express firing times as a function of the weights.

We just need to check that the Jacobian of the above 6 equations (treated as a vector valued function) differentiated w.r.t. the 6 firing times is invertible. It turns out the causality structure will ensure that the Jacobian is always lower triangular once you sort by firing times. For feed-forward networks, this is also true if you sort by firing times by layer (since firing times within the same layer do not affect each other, and the firing times of deeper layers do not affect earlier ones). This Jacobian looks like

$$\begin{array}{l} \\ V_1(f_1^1) - \theta = 0 \\ V_1(f_2^1) - \theta = 0 \\ V_1(f_3^1) - \theta = 0 \\ V_2(f_1^2) - \theta = 0 \\ V_2(f_2^2) - \theta = 0 \\ V_3(f_1^3) - \theta = 0 \end{array} \begin{array}{c} \begin{array}{cccccc} f_1^1 & f_2^1 & f_3^1 & f_1^2 & f_2^2 & f_1^3 \end{array} \\ \begin{pmatrix} x & & & & & \\ x & x & & & & \\ x & x & x & & & \\ x & & & x & & \\ x & x & x & x & x & \\ & & & x & x & x \end{pmatrix} \end{array}$$

where $x$ is marked for each equation there is a nontrivial derivative w.r.t. the corresponding variable. The lower triangular structure occurs because of the way later firing times cannot occur in the equations for earlier ones.

Invertibility holds as long as the diagonal elements are non-zero. Since each equation is equal to the membrane potential at the firing threshold, the derivative of the membrane potential w.r.t. its firing time is the equal to the derivative of the membrane potential w.r.t. $t$ evaluated at the firing time, which is strictly positive because the potential is increasing at firing time.

The Jacobian with respect to the network weights requires no special structure, but we can similarly calculate the partial derivatives of the above 6 equations now with respect to weights (to get a 6 by 3 matrix). Using (10), if we multiply the negative inverse of the Jacobian with respect to firing times (a 6 by 6 matrix) and the Jacobian with respect to weights (a 6 by 3 matrix), we get all partial derivatives of the 6 firing times with respect to all the weights (a 6 by 3 matrix).

### A.5 Experiment details

To save space in the main paper, we include more details about our experiments here.

**XOR Task** The following hyperparameters of the 2-4-2 network were used for the XOR task. No special tuning of hyperparameters was used, and the network reliably converged to 100% accuracy using parameters in Table 1.

**Iris Dataset** We did a grid search over hyperparameters to find a network suitable for Iris classification. There were several networks which achieved 100% test accuracy. One such set of hyperparameters is given in Table 2.

**Yin-Yang Dataset** The following Table 3 describes the hyperparameters used for training the SNN on the Yin-Yang dataset. The hyperparameters were chosen by a manual search through several combinations of the architecture and the parameters shown in the table. The final experiments were done on machines part of an internal cluster with 48 CPU and 5 GB memory, which results in training over 300 full epochs through the entire training dataset of 5000 examples and evaluation on the entire test dataset of 1000 examples completing in approximately 3 hours.

Code for all experiments can be found at `https://github.com/janehjlee/exact_snn_forward_prop`.

Table 1: Hyperparameters for XOR Task

| SYMBOL | DESCRIPTION | VALUE |
|---|---|---|
| $\alpha = \frac{1}{\tau_s}$ | Inverse synaptic time constant | 1.0 |
| $\beta = \frac{1}{\tau_n}$ | Inverse membrane time constant | 0.99 |
| $\theta$ | Threshold | 1.0 |
| $T$ | Maximum time | 2.0 |
| $t_{\text{early}}$ | Minimum time | 0.0 |
| | Hidden sizes | [4] |
| | Hidden weights mean | [3.0] |
| | Hidden weights stdev | [1.0] |
| | Output weights mean | 2.0 |
| | Ouput weights stdev | 0.1 |
| | Optimizer | Adam |
| $\beta_1$ | Adam parameter | 0.9 |
| $\beta_2$ | Adam parameter | 0.999 |
| $\epsilon$ | Adam parameter | $1e-8$ |
| $\eta$ | Learning rate | 0.1 |
| $\gamma$ | Regularization factor | 0.2 |
| $\tau_0$ | First loss time constant | 0.1 |
| $\tau_1$ | Second loss time constant | 1.0 |

Table 2: Hyperparameters for Iris Classification

| SYMBOL | DESCRIPTION | VALUE |
|---|---|---|
| $\alpha = \frac{1}{\tau_s}$ | Inverse synaptic time constant | 1.0 |
| $\beta = \frac{1}{\tau_n}$ | Inverse membrane time constant | 0.9 |
| $\theta$ | Threshold | 1.0 |
| $T$ | Maximum time | 16.0 |
| $t_{\text{early}}$ | Minimum time | 0.0 |
| | Hidden sizes | [10] |
| | Hidden weights mean | [3.0] |
| | Hidden weights stdev | [1.0] |
| | Output weights mean | 2.0 |
| | Ouput weights stdev | 0.1 |
| | Optimizer | Adam |
| $\beta_1$ | Adam parameter | 0.9 |
| $\beta_2$ | Adam parameter | 0.999 |
| $\epsilon$ | Adam parameter | $1e-8$ |
| $\eta$ | Learning rate | 0.05 |
| $\gamma$ | Regularization factor | 0.1 |
| $\tau_0$ | First loss time constant | 1.0 |
| $\tau_1$ | Second loss time constant | 1.0 |

Table 3: Hyperparameters for Yin-Yang Simulations

| SYMBOL | DESCRIPTION | VALUE |
|---|---|---|
| $\alpha = \frac{1}{\tau_s}$ | Inverse synaptic time constant | 0.999 |
| $\beta = \frac{1}{\tau_n}$ | Inverse membrane time constant | 1.0 |
| $\theta$ | Threshold | 1.0 |
| $T$ | Maximum time | 2.0 |
| $t_{\text{early}}$ | Minimum time | 0.15 |
| $t_{\text{bias}}$ | Bias time | 0.9 |
| | Hidden sizes | [150] |
| | Hidden weights mean | [1.5] |
| | Hidden weights stdev | [0.8] |
| | Output weights mean | 2.0 |
| | Ouput weights stdev | 0.1 |
| | Minibatch size | 150 |
| | Epochs | 300 |
| | Optimizer | Adam |
| $\beta_1$ | Adam parameter | 0.9 |
| $\beta_2$ | Adam parameter | 0.999 |
| $\epsilon$ | Adam parameter | $1e-8$ |
| $\eta$ | Learning rate | 0.0005 |
| $\gamma$ | Regularization factor | 0.005 |
| $\tau_0$ | First loss time constant | 0.2 |
| $\tau_1$ | Second loss time constant | 1.0 |