A Differentiable Point Process with Its Application to Spiking Neural Networks

Hiroshi Kajino [1]

Abstract

This paper is concerned about a learning algorithm for a probabilistic model of spiking neural networks (SNNs). Jimenez Rezende & Gerstner (2014) proposed a stochastic variational inference algorithm to train SNNs with hidden neurons. The algorithm updates the variational distribution using the score function gradient estimator, whose high variance often impedes the whole learning algorithm. This paper presents an alternative gradient estimator for SNNs based on the path-wise gradient estimator. The main technical difficulty is a lack of a general method to differentiate a realization of an arbitrary point process, which is necessary to derive the path-wise gradient estimator. We develop a differentiable point process, which is the technical highlight of this paper, and apply it to derive the path-wise gradient estimator for SNNs. We investigate the effectiveness of our gradient estimator through numerical simulation.

1. Introduction

A spiking neural network (SNN) is an artificial neural network (ANN) where neurons communicate with each other using *spikes* rather than *real values* as the conventional ANNs do. The conventional ANN is a special case of SNN where information is encoded into the firing rate of neurons (which we call the rate coding) and the rate serves as the communication currency. This specification facilitates developing learning algorithms for ANNs, leading to the recent great success of deep neural networks. On the other hand, in the community of neuroscience, experimental evidence on biological neurons indicates that the rate coding alone cannot explain the whole brain activity (Bothe, 2004) and more precise modeling of neural coding is anticipated. Since there still exist performance gaps between the rate-based ANNs and biological neural networks (*i.e.*, brains) in terms of inference capability and energy efficiency, this raises the following question: how much of the current performance gaps can be attributed to this difference on neural coding? This open problem motivates us to study SNNs.

One of the major obstacles towards answering it is a lack of practical learning algorithms for SNNs, which discourages us from empirical investigation. While there exist a number of attempts to develop learning algorithms, most of them have more or less limited applicability. We consider a practical learning algorithm should at least be (i) theoretically grounded, (ii) empirically confirmed to work well, and (iii) easy to simulate (fewer hyperparameters, less computation time, etc.)[1]. For example, theoretical aspects of the algorithms based on spike-timing-dependent plasticity (Chapter 19 (Gerstner et al., 2014)) are not well understood. For another example, simulating learning algorithms for continuous-time deterministic SNNs requires the step-size parameter of time-axis discretization when the dynamics of a neuron is described by differential equations (*e.g.*, (Huh & Sejnowski, 2018)). The step-size parameter brings about the trade-off between the simulation quality and computation time, which makes the simulation more intricate. These examples illustrate that even major approaches do not satisfy all the requirements above, and therefore, there still exists much room for improvement.

Among a number of approaches, we employ as a foundation a probabilistic formulation of SNNs (Pfister et al., 2006), which models spike trains (temporal sequence of spikes emitted from neurons) as a realization of a multivariate point process. It is easier for us to start from it than others because it already satisfies requirements (i) and (iii), which are more intrinsic properties than requirement (ii). In fact, learning algorithms are formalized by maximum likelihood estimation, and its exact simulation has no trade-off hyperparameter as will be explained in Section 2.2. Therefore, the remaining concern is its empirical performance.

One of the state-of-the-art learning algorithms for probabilistic SNNs is the work by Jimenez Rezende & Gerstner (2014). The authors propose a stochastic variational inference algorithm for SNNs with hidden neurons. Since spike trains of hidden neurons are unobservable and it is intractable to compute the marginal likelihood, an evidence

[1]IBM Research - Tokyo, Tokyo, Japan. Correspondence to: Hiroshi Kajino <kajino@jp.ibm.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]Biological plausibility is of another great interest, but is not mentioned because it is not mandatory for engineering purposes.

lowerbound (ELBO) is instead used as the objective function (Section 4.2). The key factor for optimizing ELBO is the way we estimate the gradient of ELBO. The authors employed the score function gradient estimator, also known as the REINFORCE estimator, which is widely applicable but is often reported to suffer from its high variance.

Our main idea is to substitute a path-wise gradient estimator for the score function gradient estimator. The path-wise gradient estimator tends to have lower variance than the score function gradient estimator (Mohamed et al., 2019), but it is not widely applicable (and is not applicable to SNNs) because it requires a sample from the variational distribution to be differentiable. Our contribution is that we develop a differentiable point process (Section 3) and apply it to derive the path-wise gradient estimator for SNNs (Section 4.2.2).

We empirically investigate the effectiveness of the proposed learning algorithm in Section 5. We will confirm that (i) the proposed gradient estimator has lower variance than the existing one and (ii) this lower variance contributes to improve the performance of the learning algorithm. By comparing the performance of the proposed and existing ones, we obtain experimental results supporting these hypotheses. Therefore, we conclude that our path-wise gradient estimator improves empirical performance of SNNs.

One of the limitations of our learning algorithm as compared to the existing algorithm (Jimenez Rezende & Gerstner, 2014) is computation time. Since our algorithm generates more hidden spikes than the existing one does, our algorithm requires more computation time. We empirically examine the computational overhead of our algorithm against the existing one, and find that our algorithm requires 2.8 times more computation time than the existing one.

Notation. Let $[N] = \{1, 2, \dots, N\}$. For any vector $\mathbf{x}$, its d-th element is represented by x_d. $\left[x_d\right]_{d\in[D]}$ denotes a D-dimensional vector whose d-th element is x_d. For any vector $\mathbf{x} \in \mathbb{R}^D$ and scalar $c \in \mathbb{R}$, $\begin{bmatrix}\mathbf{x}^\top & c\end{bmatrix}^\top$ denotes the $(D+1)$-dimensional vector concatenating $\mathbf{x}$ and c. Let $\mathbb{R}_{\geq 0} = \{x \geq 0\}$ and $\mathbb{R}_{>0} = \{x > 0\}$. Let $\mathbb{1}^D = \{\mathbf{1}_d\}_{d\in[D]}$ be the set of D-dimensional one-hot vectors, where $\mathbf{1}_d \in \{0,1\}^D$ is the one-hot vector whose d-th element is 1 and the others are 0. For any set A, let $\mathrm{conv}(A)$ be its convex hull, let $\mathrm{conv}_0(A) := \mathrm{conv}(A \cup \{0\})$. Let $\mathrm{Cat}(\mathbf{p})$ be the categorical distribution with parameter $\mathbf{p} \in \mathrm{conv}(\mathbb{1}^D)$, whose random variable takes $\mathbf{1}_d \in \mathbb{1}^D$ with probability p_d. Let $U[a, b]$ denote the uniform distribution over $[a, b]$. For any expectation operator $\mathbb{E}_p$, let $\hat{\mathbb{E}}_p$ be its Monte-Carlo approximation using an i.i.d. sample from p.

2. Preliminaries

This section introduces temporal point processes along with their parameter estimation and sampling methods.

2.1. Point Processes

A point process (Daley & Vere-Jones, 2003) is a probabilistic model of an event collection. It is called a *temporal point process* when the event collection evolves in time. This paper only deals with a temporal point process, and therefore, we refer to it as a point process. We assume that point processes are *simple*, *i.e.*, no events coincide.

2.1.1. Univariate Point Process

Assume we observe a sequence of $N \in \mathbb{N}$ discrete events during time interval $[0, T]$, and let $\mathcal{T}$ denote such an observation. $\mathcal{T}$ can be represented by a series of event time stamps $\{t_n \in [0, T]\}_{n\in[N]}$ as well as the information that we observe no event during $[0, t_1)$, $\{(t_n, t_{n+1})\}_{n=1}^{N-1}$, and $(t_N, T]$. Let $\mathcal{T}^{\leq t_n}$ represents a partial observation of $\mathcal{T}$ up to and including time t_n. One way of modeling $\mathcal{T}$ is to specify the probability density function of the event time stamp t_{n+1} given the collection of its past events $\mathcal{T}^{\leq t_n}$, which we describe, $f(t \mid \mathcal{T}^{\leq t_n})$. Note that the probability density function must satisfy $f(t \mid \mathcal{T}^{\leq t_n}) = 0$ for $t \leq t_n$ and $\int_{t_n}^{\infty} f(t \mid \mathcal{T}^{\leq t_n})\mathrm{d}t = 1$. The cumulative distribution function can be defined accordingly: $F(t \mid \mathcal{T}^{\leq t_n}) = \int_{t_n}^{t} f(s \mid \mathcal{T}^{\leq t_n})\mathrm{d}s = \Pr[t_{n+1} \in (t_n, t) \mid \mathcal{T}^{\leq t_n}]$.

Another way of modeling it is to specify the conditional intensity function, which is related to the distributions as,

$$\lambda(t \mid \mathcal{T}^{\leq t_n}) = \begin{cases} \dfrac{f\left(t \mid \mathcal{T}^{\leq t_n}\right)}{1 - F\left(t \mid \mathcal{T}^{\leq t_n}\right)} & (t > t_n), \\ 0 & (t \leq t_n). \end{cases} \tag{1}$$

In the following, let t_n denote an arbitrary event time stamp and we only specify the conditional intensity function for $t > t_n$, because its value for $t \leq t_n$ is trivially 0. Observing that $\lambda(t \mid \mathcal{T}^{\leq t_n})\mathrm{d}t = \Pr[t_{n+1} \in [t, t + \mathrm{d}t] \mid t_{n+1} \notin (t_n, t), \mathcal{T}^{\leq t_n}]$ holds as $\mathrm{d}t \to +0$ (Rasmussen, 2018), the conditional intensity function represents how likely the event occurs at time t given that we have observed n events so far and no event has been observed during (t_n, t).

A point process is more often specified by the conditional intensity function than the time interval distribution. Let $\mathcal{PP}(\lambda)$ be the point process with the conditional intensity function λ. Corollary 1, which is an immediate consequence of Proposition 2, states the conditions under which the conditional intensity function uniquely specifies a point process.

Corollary 1. *A conditional intensity function λ uniquely defines a point process if it satisfies the following conditions for any observation of discrete events $\mathcal{T}^{\leq t_n}$ and any $t > t_n$:*

1. *$\lambda(t \mid \mathcal{T}^{\leq t_n})$ is non-negative and integrable on any interval starting at t_n,*
2. *$\int_{t_n}^{t} \lambda(s \mid \mathcal{T}^{\leq t_n})\mathrm{d}s \to \infty$ as $t \to \infty$, and*

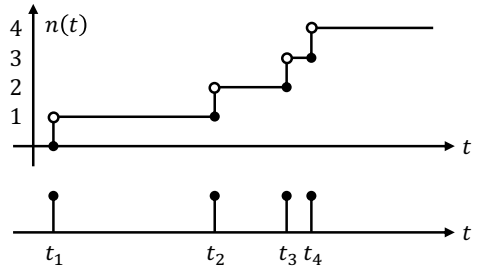

Figure 1. Realization of a temporal point process (bottom) and its corresponding left-continuous counting process (top).

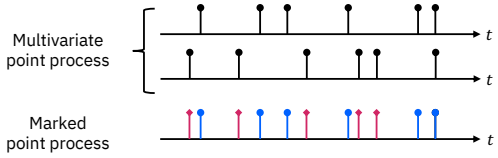

Figure 2. Illustration of a multivariate point process (top) and its equivalent marked point process (bottom).

3. $\int_{t_n}^{t} \lambda(s \mid \mathcal{T}^{\leq t_n})\mathrm{d}s$ *is right continuous w.r.t.* t.

The log-likelihood of observation $\mathcal{T}$ on $\mathcal{PP}(\lambda)$ is given as,

$$\log p(\mathcal{T}) = \sum_{t \in \mathcal{T}} \log \lambda \left(t \mid \mathcal{T}^{\leq t_{n(t)}}\right) - \Lambda^{[0,T]}(\mathcal{T}), \quad (2)$$

where let $\Lambda^{[0,T]}(\mathcal{T}) = \int_0^T \lambda(t \mid \mathcal{T}^{\leq t_{n(t)}})\mathrm{d}t$ be the integrated conditional intensity function, also known as the *compensator*, which accounts for no-event periods, and let $n(t) : \mathbb{R}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be the left-continuous[2] counting process of the observation $\mathcal{T}$, which counts the number of events up to but not including time t. The latest event time stamp at time t can be denoted by $t_{n(t)} \in [0, t)$. Figure 1 illustrates a realization of a point process and its counting representation. A typical procedure of modeling $\mathcal{T}$ is to design a parametric model of the conditional intensity function that satisfies the conditions of Corollary 1 and train it by maximizing the log-likelihood function (Equation (2)).

2.1.2. Multivariate Point Process

A multivariate point process is a set of mutually dependent point processes and can be defined via a *marked point process*, in which each event is associated with a *mark*. We call a marked point process whose mark belongs to set X, an *X-marked point process*. Let $\mathcal{T}_X$ denote an observation of an X-marked point process, which contains a series of event time stamps and marks, $\{(t_n, \mathbf{p}_n) \in [0, T] \times X\}_{n \in [N]}$. As illustrated in Figure 2, a D-variate point process can be defined by a $\mathbb{1}^D$-marked point process, where each mark $\mathbf{p}_n$ indicates which dimension the event belongs to. For example, if $\mathbf{p}_n = \mathbf{1}_1$, the n-th event occurs at the first dimension. In Figure 2, blue-circle and red-diamond marks correspond to the first and the second dimensions respectively.

Letting $f(t, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n})$ be the probability density function of each event $(t_{n+1}, \mathbf{p}_{n+1})$ given its past events $\mathcal{T}_{\mathbb{1}^D}^{\leq t_n}$, the conditional intensity function can be defined similarly:

$$\lambda\left(t, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}\right) = \frac{f\left(t, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}\right)}{1 - F\left(t \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}\right)}, \quad (3)$$

where $F(t \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}) = \int_{t_n}^{t} \mathrm{d}s \sum_{\mathbf{p} \in \mathbb{1}^D} f(s, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n})$. The conditional intensity function represents how likely event $(t, \mathbf{1}_d)$ occurs: $\lambda(t, \mathbf{1}_d \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n})\mathrm{d}t = \Pr[t_{n+1} \in [t, t + \mathrm{d}t], \mathbf{p}_{n+1} = \mathbf{1}_d \mid t_{n+1} \notin (t_n, t), \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}]$. Proposition 2 states conditions under which the conditional intensity function uniquely specifies a marked point process. See Appendix B for its proof. Let $\mathcal{MPP}(\lambda)$ be the multivariate point process with the conditional intensity function λ.

Proposition 2. *Let X be a set. A conditional intensity function λ uniquely defines an X-marked point process if it satisfies the following conditions for any $\mathcal{T}_X^{\leq t_n}$ and $t > t_n$:*

1. $\lambda(t, \mathbf{p} \mid \mathcal{T}_X^{\leq t_n}) \geq 0$ *and integrable w.r.t.* $\mathbf{p}$ *and w.r.t.* t *on any interval starting at* t_n,
2. $\int_{t_n}^{t} \mathrm{d}s \int_X \mathrm{d}\mathbf{p} \lambda(s, \mathbf{p} \mid \mathcal{T}_X^{\leq t_n}) \to \infty$ *as* $t \to \infty$, *and*
3. $\int_{t_n}^{t} \mathrm{d}s \int_X \mathrm{d}\mathbf{p} \lambda(s, \mathbf{p} \mid \mathcal{T}_X^{\leq t_n})$ *is right continuous in* t.

The log-likelihood of observation $\mathcal{T}_{\mathbb{1}^D}$ is written as:

$$\log p(\mathcal{T}_{\mathbb{1}^D}) = \sum_{(t, \mathbf{p}) \in \mathcal{T}_{\mathbb{1}^D}} \log \lambda \left(t, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_{n(t)}}\right) - \Lambda^{[0,T]}(\mathcal{T}_{\mathbb{1}^D}), \quad (4)$$

where let $\Lambda^{[0,T]}(\mathcal{T}_{\mathbb{1}^D}) = \int_0^T \sum_{\mathbf{p} \in \mathbb{1}^D} \lambda(t, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n(t)})\mathrm{d}t$ be the compensator. Since its analytical form is not available for a general conditional intensity function, we resort to Monte-Carlo approximation to estimate the compensator. In specific, we draw M examples, $\{t_m\}_{m \in [M]}$, from $U[0, T]$ and approximate it as,

$$\Lambda^{[0,T]}(\mathcal{T}_{\mathbb{1}^D}) \approx \frac{T}{M} \sum_{m=1}^{M} \sum_{\mathbf{p} \in \mathbb{1}^D} \lambda \left(t_m, \mathbf{p} \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_{n(t_m)}}\right). \quad (5)$$

2.2. Sampling Algorithms

This section introduces sampling algorithms for a point process given a conditional intensity function. A notable

[2] A counting process is usually defined to be right-continuous. We introduce the left-continuous one so as to represent the integrand of the compensator concisely. See Appendix A for details.

Algorithm 1 Thinning algorithm for $\mathcal{MPP}$

Input: Conditional intensity function λ and upperbound $\bar{\lambda}$
Output: Realization of $\mathcal{MPP}(\lambda)$

1: $\mathcal{S} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$
2: **while true do**
3: Sample $s \sim \mathcal{PP}(\bar{\lambda} \mid \mathcal{S})$
4: **if** $s > T$ **then**
5: break
6: Sample $\begin{bmatrix} \mathbf{p} \\ r \end{bmatrix} \sim \mathrm{Cat}\left(\boldsymbol{\pi}_{\bar{\lambda}} \circ \boldsymbol{\lambda}\left(s \mid \mathcal{T}\right)\right)$
7: **if** $r \neq 1$ **then**
8: $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s, \mathbf{p})\}$
9: $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
10: **return** $\mathcal{T}$

feature of the algorithms is that they can exactly simulate point processes without any approximation. This indicates that there exists no hyperparameter controling the trade-off between computational cost and accuracy of the simulation, which greatly facilitates simulating SNNs.

2.2.1. Homogeneous Poisson Process

The simplest point process is the homogeneous Poisson process whose conditional intensity function is constant; $\lambda(t \mid \mathcal{T}^{\leq t_n}) = \lambda$ for any $\mathcal{T}^{\leq t_n}$. It is straightforward to sample from it because the interval between two successive events τ is independently and identically distributed according to the exponential distribution, $f(\tau; \lambda) = \lambda \mathrm{e}^{-\lambda\tau}$.

2.2.2. General Point Process

It is not straightforward to sample from a general point process when a closed-form expression of the inter-event time distribution is not available. This is true for many point processes including SNNs. Among several sampling methods, the thinning algorithm (Lewis & Shedler, 1979; Ogata, 1981) allows us to sample from such a point process without knowing the closed-form expression. For other sampling algorithms, please refer to Section 6.

The main idea is to generate a sequence of time stamps from a homogeneous Poisson process with sufficiently high intensity (which we call the *base process*) and then to thin some of the events so that the sequence follows the given point process. Algorithm 1 describes it for the multivariate case, where let $\boldsymbol{\lambda}\left(t \mid \mathcal{T}\right) = \left[\lambda\left(t, \mathbf{1}_d \mid \mathcal{T}\right)\right]_{d \in [D]}$, and let $\boldsymbol{\pi}_{\bar{\lambda}} \circ$ be an operator that receives a D-dimensional vector $\boldsymbol{\lambda}$ and returns $\frac{1}{\bar{\lambda}} \begin{bmatrix} \boldsymbol{\lambda} \\ \bar{\lambda} - \|\boldsymbol{\lambda}\|_1 \end{bmatrix}$.

It first generates a new time stamp s from the homogeneous Poisson process with intensity $\bar{\lambda}$ (line 3). Then it decides whether or not to accept the event, and if accepting, decides which dimension the event is assigned to (lines 6-8); s is rejected if $r = 1$, *i.e.*, with probability $1 - \frac{1}{\bar{\lambda}} \sum_{\mathbf{p} \in \mathbb{1}^D} \lambda(s, \mathbf{p} \mid \mathcal{T})$, and s is accepted as the event from the d-th dimension ($d \in [D]$) if $p_d = 1$, *i.e.*, with probability $\lambda\left(s, \mathbf{1}_d \mid \mathcal{T}\right) / \bar{\lambda}$,

Intuitively, the correctness of Algorithm 1 is understood as follows. Assuming we have sampled $\mathcal{T}_{\mathbb{1}^D}^{\leq t_n}$, at any time $t > t_n$, the probability that the algorithm generates the event with mark $\mathbf{1}_d$ in interval $[t, t + \mathrm{d}t]$ is,

$$
\begin{aligned}
&\Pr\left[t_{n+1} \in [t, t + \mathrm{d}t], \mathbf{p}_{n+1} = \mathbf{1}_d \mid \mathcal{T}_{\mathbb{1}^D}^{<t}\right] \\
=& \underbrace{\bar{\lambda}\mathrm{d}t}_{\substack{\text{Prob. that the base process} \\ \text{generates the event in } [t, t+\mathrm{d}t]}} \underbrace{\cdot \lambda\left(t, \mathbf{1}_d \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}\right) / \bar{\lambda}}_{\substack{\text{Prob. that } t \text{ is} \\ \text{assigned the } d\text{-th mark}}} \\
=& \lambda\left(t, \mathbf{1}_d \mid \mathcal{T}_{\mathbb{1}^D}^{\leq t_n}\right) \mathrm{d}t,
\end{aligned}
$$

where let $\mathcal{T}_{\mathbb{1}^D}^{<t}$ denote the event $t_{n+1} \notin (t_n, t)$ and $\mathcal{T}_{\mathbb{1}^D}^{\leq t_n}$. This shows that the output follows $\mathcal{MPP}(\lambda)$. For its formal proof, please refer to Reference (Ogata, 1981).

3. Differentiable Point Process

We present the key building block of our method called a *differentiable point process*, whose realization is differentiable with respect to its parameters. Differentiability plays an essential role when designing a learning algorithm for latent variable models as will be discussed in Section 4.2.

The key idea is that the output of Algorithm 1 becomes differentiable if we replace the categorical distribution in line 6 with a reparameterizable distribution such as the concrete distribution, also known as the Gumbel-softmax distribution (Maddison et al., 2017; Jang et al., 2017). We first review the concrete distribution (Section 3.1), and then we present the differentiable point process (Section 3.2).

3.1. Concrete Distribution

The concrete distribution has been developed as a reparameterizable substitute for the categorical distribution. The idea comes from the Gumbel-max trick, which enables us to sample from the categorical distribution. Letting $\boldsymbol{\pi} \in \mathbb{R}_{\geq 0}^D$ be an unnormalized parameter of the categorical distribution, the Gumbel-max trick first samples $u_d \sim U[0, 1]$ for each $d \in [D]$, and then outputs $\mathbf{1}_{d^\star}$ where $d^\star = \arg\max_{d \in [D]} \log \pi_d - \log(-\log u_d)$. The output is known to be distributed according to $\mathrm{Cat}(\boldsymbol{\pi}/\|\boldsymbol{\pi}\|_1)$. While the Gumbel-max trick successfully divides the sampling procedure into random sampling from the fixed distribution and a parameterized transformation of it, which is necessary to be differentiable, the gradient of its realization with respect to $\boldsymbol{\pi}$ is non-informative, because a small variation to $\boldsymbol{\pi}$ does not change the gradient.

The concrete distribution is defined by relaxing the range of

Algorithm 2 Thinning algorithm for $\partial\mathcal{PP}$

Input: Conditional intensity function λ, its upperbound $\bar{\lambda}$, and temperature $\tau > 0$.
Output: Realization of $\partial\mathcal{PP}\left(\lambda, \bar{\lambda}, \tau\right)$

$\mathcal{S} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset$
while true do
  Sample $s \sim \mathcal{PP}(\bar{\lambda} \mid \mathcal{S})$
  if $s > T$ **then**
    break
  Sample $\begin{bmatrix}\mathbf{p} \\ r\end{bmatrix} \sim \mathrm{Concrete}_\tau\left(\boldsymbol{\pi}_{\bar{\lambda}} \circ \boldsymbol{\lambda}\left(s \mid \mathcal{T}\right)\right)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s, \mathbf{p})\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
return $\mathcal{T}$

the random variable from $\mathbb{1}^D$ to its convex hull $\mathrm{conv}(\mathbb{1}^D)$ so that its gradient is more informative. Accordingly, the argmax operator in the Gumbel-max trick is replaced with the softmax operator with temperature $\tau > 0$. Since softmax becomes equivalent to argmax as $\tau \to 0$, the concrete distribution also becomes equivalent to the categorical distribution as $\tau \to 0$. Let $g_\tau(\mathbf{p}; \boldsymbol{\pi})$ denote the probability density function of the concrete distribution with temperature τ and unnormalized parameter $\boldsymbol{\pi} \in \mathbb{R}^D_{>0}$.

3.2. Multivariate Differentiable Point Process

We present a constructive definition of a differentiable point process in Definition 3.

Definition 3. *Assume the conditional intensity function $\lambda(t, \mathbf{p} \mid \mathcal{T}^{\le t_n}_{\mathbb{1}^D})$ can be computed with an observation of a $\mathrm{conv}_0(\mathbb{1}^D)$-marked point process. Let $\bar{\lambda}$ be a constant satisfying $\bar{\lambda} > \sum_{\mathbf{p}\in\mathbb{1}^D} \lambda(t, \mathbf{p} \mid \mathcal{T}^{\le t_n}_{\mathrm{conv}_0(\mathbb{1}^D)})$ for any $\mathcal{T}^{\le t_n}_{\mathrm{conv}_0(\mathbb{1}^D)}$ and $t > t_n$, and $\tau > 0$ be temperature. The differentiable point process $\partial\mathcal{PP}\left(\lambda, \bar{\lambda}, \tau\right)$ is defined as a $\mathrm{conv}_0(\mathbb{1}^D)$-marked point process constructed by Algorithm 2.*

Algorithms 1 and 2 are different in two ways. First, all events from the base process are accepted in Algorithm 2, while some are rejected in Algorithm 1. Second, in Algorithm 1, the mark is defined over $\mathbb{1}^D$, while in Algorithm 2, it is defined over $\mathrm{conv}_0(\mathbb{1}^D)$; each mark is associated with amplitude that is continuous w.r.t. the model parameter.

The differentiable point process as defined above can be understood as a marked point process (Proposition 4).

Proposition 4. *The differentiable point process $\partial\mathcal{PP}(\lambda, \bar{\lambda}, \tau)$ is a $\mathrm{conv}_0(\mathbb{1}^D)$-marked point process with conditional intensity function,*

$$\lambda_\partial\left(t, \mathbf{p} \mid \mathcal{T}^{\le t_n}_{\mathrm{conv}_0(\mathbb{1}^D)}; \boldsymbol{\lambda}, \bar{\lambda}, \tau\right) = \bar{\lambda} \cdot g_\tau\left(\begin{bmatrix}\mathbf{p} \\ 1 - \|\mathbf{p}\|_1\end{bmatrix}; \boldsymbol{\pi}_{\bar{\lambda}} \circ \boldsymbol{\lambda}\left(t \mid \mathcal{T}^{\le t_n}_{\mathrm{conv}_0(\mathbb{1}^D)}\right)\right).$$

We can confirm the differentiability of a realization of $\partial\mathcal{PP}$ (Proposition 5). We can also confirm that in the limit of $\tau \to 0$, the differentiable point process becomes equivalent to the original point process (Proposition 6). See Appendix C for their formal statements and proofs.

As discussed by Maddison et al. (2017), the concrete distribution often suffers from underflow and we have to implement it in the logarithmic scale. Our implementation also suffers from the same issue, and we provide a numerically stable implementation idea in Appendix E.

4. Learning Algorithm for SNNs

We present a learning algorithm for spiking neural networks (SNNs) based on the differentiable point process. We first define a probabilistic model of SNNs (Section 4.1) and then will present our learning algorithm, highlighting the difference from the existing one (Section 4.2).

4.1. Probabilistic Model of Spiking Neural Networks

We employ the standard probabilistic model in the literature (Pfister et al., 2006). Let D be the number of neurons, let $\mathcal{N} = \mathbb{1}^D$ be the set of neurons, each of which is indexed by a one-hot vector, and let $\mathcal{T}_\mathcal{N}$ be spike trains emitted from SNN during time interval $[0, T]$. We assume that $\mathcal{T}_\mathcal{N}$ is a realization of an $\mathcal{N}$-marked point process.

We define the conditional intensity function based on a *spike response model* (SRM) (Gerstner et al., 2014). SRM assumes that the d-th spiking neuron is driven by its internal state called a membrane potential,

$$u_d\left(t \mid \mathcal{T}^{\le t_n}_\mathcal{N}\right) = \bar{u}_d + \sum_{(t', \mathbf{p}) \in \mathcal{T}^{\le t_n}_\mathcal{N}} \mathbf{f}_d(t - t') \cdot \mathbf{p}, \tag{6}$$

where $\mathbf{f}_d(s) = \left[f_{d',d}(s)\right]_{d' \in [D]}$ is a vector of filter functions from all of the neurons to the d-th neuron. In specific, $f_{d',d}(s)$ describes the time course of the membrane potential of neuron d in response to a spike emitted by neuron d' at time $s = 0$. We assume $f_{d,d}(s) \le 0$ for all $d \in \mathcal{N}$. This assumption allows us to reproduce the resetting behavior of a biological neuron; the membrane potential is reset to a lower level after the neuron fires. We also assume that $f_{d',d}(s) = 0$ for $s < 0$. This assumption ensures that future events have no influence on past events.

Then, the conditional intensity function is defined by,

$$\lambda^{\mathrm{SNN}}(t, \mathbf{p} \mid \mathcal{T}^{\le t_n}_\mathcal{N}) = \mathbf{p} \cdot \sigma(\mathbf{u}(t \mid \mathcal{T}^{\le t_n}_\mathcal{N})), \tag{7}$$

where $\sigma\colon \mathbb{R}^D \to \mathbb{R}^D_{\ge 0}$ is element-wisely non-decreasing and differentiable[3] and let $\mathbf{u}(t \mid \mathcal{T}^{\le t_n}_\mathcal{N}) = \left[u_d(t \mid \mathcal{T}^{\le t_n}_\mathcal{N})\right]_{d \in [D]}$.

[3] We use the sigmoid function multiplied by amplitude $a > 0$ element-wisely as σ, for which $\bar{\lambda}$ is easy to derive.

As the membrane potential of one neuron increases, the neuron is more likely to fire and generate a spike.

For numerical simulation, we assume that the filter functions are parameterized by weights $\{w_{d',d,l} \in \mathbb{R}\}_{l=1}^{L}$ as,

$$f_{d',d}(s) = \begin{cases} \sum_{l=1}^{L} w_{d',d,l} \cdot \kappa(s - s_l) & (s \geq 0), \\ 0 & (s < 0), \end{cases} \tag{8}$$

where $\{s_l \in \mathbb{R}\}_{l=1}^{L}$ are fixed and $\kappa(s) = \max\{\frac{3}{4}(1-s^2), 0\}$ is the Epanechnikov kernel. We chose this kernel because the bounded support of the kernel allows us to ignore events that occurred more than a certain period ago for membrane potential computation. Let $\theta = \{\bar{u}_d \in \mathbb{R}\}_{d=1}^{D} \cup \{w_{d',d,l} \in \mathbb{R} \mid l \in [L]\}_{d,d'=1}^{D}$ denote the set of model parameters.

4.2. Learning Algorithms

Assume some of the neurons are hidden and their spike trains are unobservable. Let $\mathcal{O} \subset \mathcal{N}$ and $\mathcal{H} = \mathcal{N} \backslash \mathcal{O}$ be the sets of observable and hidden neurons, respectively. Accordingly, the spike trains of all of the neurons are divided into observable and hidden ones: $\mathcal{T}_{\mathcal{N}} = \mathcal{T}_{\mathcal{O}} \cup \mathcal{T}_{\mathcal{H}}$. We consider an estimation procedure for the model parameters of SNN, θ, given a set of observed spike trains $\{\mathcal{T}_{\mathcal{O},n}\}_{n=1}^{N}$.

Letting $p(\mathcal{T}_{\mathcal{N}}; \theta) = p(\mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}; \theta)$ be the joint distribution of the observable and hidden spike trains, the parameter θ is estimated by maximum likelihood estimation:

$$\underset{\theta}{\text{maximize}} \quad \sum_{n=1}^{N} \ell\left(\theta; \mathcal{T}_{\mathcal{O},n}\right)$$

where $\ell(\theta; \mathcal{T}_{\mathcal{O}}) = \log \int p(\mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}; \theta) \mathrm{d}\mathcal{T}_{\mathcal{H}}$ is the marginalized log-likelihood function. Since it is intractable to compute it, we substitute its lower bound called an *evidence lower bound* (ELBO) for the marginalized log-likelihood function as the objective function:

$$\begin{aligned} \underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}}) &= \mathbb{E}_{q(\mathcal{T}_{\mathcal{H}};\phi)} \left[\log p(\mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}; \theta) - \log q(\mathcal{T}_{\mathcal{H}}; \phi)\right], \\ &\equiv \mathbb{E}_{q(\mathcal{T}_{\mathcal{H}};\phi)} \underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}), \end{aligned} \tag{9}$$

where $q(\mathcal{T}_{\mathcal{H}}; \phi)$ is an arbitrary distribution called a *variational distribution*, parameterized by ϕ. We specifically assume that the variational distribution is modeled by SNN driven by both observable and hidden spike trains. In the following, we omit the index of data n for ease of presentation and consider ELBO using a single observation $\mathcal{T}_{\mathcal{O}}$.

Since there exists no closed-form solution to the maximization problem, we resort to stochastic gradient ascent methods, resulting in Algorithm 3. The basic procedure to train SNN is to choose one realization $\mathcal{T}_{\mathcal{O}}$ from the data set randomly, and update θ and ϕ so as to maximize Equation (9). In the following, we present both an existing approach and our novel approach to compute the gradients, $\frac{\partial \underline{\ell}}{\partial \theta}$ and $\frac{\partial \underline{\ell}}{\partial \phi}$.

Algorithm 3 Generic learning algorithm

Input: Observation $\mathcal{T}_{\mathcal{O}}$, learning rate $\{\alpha_k\}_{k=1}^{K}$.
Output: Model parameters θ, ϕ.

1: Initialize θ, ϕ
2: **for** $k = 1, \ldots, K$ **do**
3: Update $\theta \leftarrow \theta + \alpha_k \frac{\partial \underline{\ell}}{\partial \theta}(\theta, \phi; \mathcal{T}_{\mathcal{O}})$
4: Update $\phi \leftarrow \phi + \alpha_k \frac{\partial \underline{\ell}}{\partial \phi}(\theta, \phi; \mathcal{T}_{\mathcal{O}})$
5: **return** θ, ϕ

4.2.1. Gradient with respect to θ

The gradient with respect to θ is straightforwardly computed by applying Monte-Carlo approximation: $\frac{\partial}{\partial \theta} \underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}}) \approx \hat{\mathbb{E}}_{q(\mathcal{T}_{\mathcal{H}};\phi)} \left[\frac{\partial}{\partial \theta} \log p(\mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}; \theta)\right]$. This can be numerically calculated with the help of automatic differentiation tools.

4.2.2. Gradient with respect to ϕ

The gradient with respect to ϕ is more involved. In Equation (9), the expectation operator depends on ϕ and we cannot exchange $\frac{\partial}{\partial \phi}$ and $\mathbb{E}_{q(\mathcal{T}_{\mathcal{H}};\phi)}$. There are at least two approaches to computing the gradient in this situation (Mohamed et al., 2019). One approach is to rely on *the score function gradient estimator*, also known as the REINFORCE estimator (Williams, 1992). While it is widely applicable to a variety of models, it is often reported that the gradient estimator has high variance. Another approach is *the path-wise gradient estimator*, which makes use of the reparameterization trick (Kingma & Welling, 2014). While its variance is often reported to be lower than that of the score function gradient estimator (Mohamed et al., 2019), its application is limited because the probability distribution q must be reparameterizable.

In the literature of SNNs, the score function gradient estimator with respect to ϕ has been developed by Jimenez Rezende & Gerstner (2014). Our contribution is to develop a path-wise gradient estimator for SNNs based on a differentiable point process presented in Section 3.

Score function gradient estimator. Jimenez Rezende & Gerstner (2014) used the score function gradient estimator for computing the gradient with respect to ϕ: $\frac{\partial \underline{\ell}}{\partial \phi}(\theta, \phi; \mathcal{T}_{\mathcal{O}}) \approx \hat{\mathbb{E}}_{q(\mathcal{T}_{\mathcal{H}};\phi)}[\frac{\partial \log q(\mathcal{T}_{\mathcal{H}};\phi)}{\partial \phi}(\underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathcal{H}}) - 1)]$. While this is an unbiased estimator of the gradient, its high variance is often problematic. We employ the variational distribution with the conditional intensity function,

$$\lambda_q(t, \mathbf{p} \mid \mathcal{T}_{\mathcal{N}}^{<t_n}; \phi) = \mathbf{p} \cdot \sigma(\mathbf{u}(t \mid \mathcal{T}_{\mathcal{N}}^{<t_n}; \phi)), \tag{10}$$

for any $\mathbf{p} \in \mathcal{H}$. In particular, we use shared parameters for the model and the variational distribution, *i.e.*, we set $\phi = \theta$ as we observe it improves the performance.

Path-wise gradient estimator. We propose a path-wise gra-

dient estimator for SNNs. Our main idea is to employ the differentiable point process, $\partial\mathcal{PP}(\lambda_q(t, \mathbf{p} \mid \mathcal{T}_{\mathcal{N}}; \phi); \bar{\lambda}, \tau)$, as the variational distribution, where λ_q is defined in Equation (10). This allows us to differentiate a Monte-Carlo approximation of ELBO (Equation (9)) using automatic differentiation tools:

$$\frac{\partial \underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}})}{\partial \phi} \approx \frac{\partial \hat{\mathbb{E}}_{\partial\mathcal{PP}} \underline{\ell}(\theta, \phi; \mathcal{T}_{\mathcal{O}}, \mathcal{T}_{\mathrm{conv}_0(\mathcal{H})}(\phi))}{\partial \phi}. \quad (11)$$

The main technical issue in applying the differentiable point process is that its realization $\mathcal{T}_{\mathrm{conv}_0(\mathcal{H})}(\phi)$ is incompatible with the SNN model defined by Equations (6) and (7). The model assumes that a mark $\mathbf{p}$ is a one-hot vector, while a mark of a differentiable point process belongs to $\mathrm{conv}_0(\mathcal{H})$. We address this by devising a *differentiable spiking neural network* (∂SNN), which can handle a mark in $\mathrm{conv}_0(\mathcal{H})$, while keeping the conditional intensity function proper.

Let $\bar{\mathcal{N}} = \mathcal{O} \cup \mathrm{conv}_0(\mathcal{H})$ be the set of marks for ∂SNN. We define the membrane potential of neuron $d \in \mathcal{N}$ as,

$$u_d\left(t \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}\right) = \bar{u}_d + \sum_{(t', \mathbf{p}) \in \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}} \mathbf{f}_d(t - t') \cdot \mathbf{p}, \quad (12)$$

and the conditional intensity of ∂SNN for $\mathbf{p} \in \bar{\mathcal{N}}$ as,

$$\begin{aligned} &\lambda^{\partial\mathrm{SNN}}\left(t, \mathbf{p} \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}; \bar{\lambda}, \tau\right) \quad (13) \\ &= \sum_{\mathbf{1}_d \in \mathcal{O}} \delta(\mathbf{p} - \mathbf{1}_d) \lambda^{\mathrm{SNN}}\left(t, \mathbf{p} \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}\right) \\ &\quad + \mathbb{I}[\mathbf{p} \in \mathrm{conv}_0(\mathcal{H})] \lambda_\partial\left(t, \mathbf{p}_{\mathcal{H}} \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}; \boldsymbol{\lambda}_{\mathcal{H}}, \bar{\lambda}, \tau\right) \end{aligned}$$

where $\boldsymbol{\lambda}_{\mathcal{H}}\left(t \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n}\right) = \sigma\left(\left[u_d(t \mid \mathcal{T}_{\bar{\mathcal{N}}}^{\leq t_n})\right]_{d \in \mathcal{H}}\right)$, $\mathbb{I}[\cdot]$ is the indicator function, and $\mathbf{p}_{\mathcal{H}} = \left[p_d\right]_{d \in \mathcal{H}}$.

It is necessary to confirm that (i) the conditional intensity function can be calculated using past events whose marks are in $\bar{\mathcal{N}}$ and (ii) the conditional intensity function satisfies all of the conditions listed in Proposition 2 for $\mathcal{X} = \bar{\mathcal{N}}$. The first requirement immediately follows from Equations (12) and (13). In Appendix D, we provide the formal statement and proof of the second requirement (Proposition 7). We also confirm that ELBO is differentiable (Proposition 8) and that the differentiable SNN becomes equivalent to the vanilla SNN in the limit of $\tau \to 0$ (Proposition 9).

5. Empirical Studies

Let us investigate the effectiveness of our gradient estimator through numerical simulation. Our hypothesis is that (i) the path-wise gradient estimator will have lower variance than the score function estimator and (ii) lower variance will improve the predictive performance. We design two experiments (Sections 5.1 and 5.2) to verify these two hypotheses. We additionally compare computation cost of the learning algorithms using each of the gradient estimators in Section 5.3. All the experiments are conducted on IBM Cloud[4], and the code is publicly available (Kajino, 2021).

Table 1. Configuration of SNN generating a synthetic data set.

Network size	$D = 6$, $\lvert\mathcal{O}\rvert = 2$, $\lvert\mathcal{H}\rvert = 4$
Activation/filter functions	$a = 5$, $L = 2$, $s_1 = 0$, $s_2 = 10$
$\partial\mathcal{PP}$	$\tau = 0.3$, $\bar{\lambda} = 20$
# of samplings	100 (Eq. (5)), 1 (Eq. (9))

Data set. We use a synthetic data set generated by the vanilla SNN (Equation (7)). Table 1 summarizes its configuration. We set $\bar{\lambda} = a|\mathcal{H}| = 20$, which is the tightest upper-bound because we use the sigmoid activation function with amplitude a. The weights are randomly sampled: biases from $U[-1, 1]$, off-diagonal kernel weights from $U[-5, 5]$, and diagonal kernel weights from $U[-5, -0.1]$.

Methods compared. Since our objective is to highlight the performance gap between our path-wise gradient estimator (∂SNN) and the score function gradient estimator (SNN), we use the same hyperparameters and initialization for both of them as much as possible. We initialize their parameters randomly using the same random seed so that both of them have random but the same initial parameters. We also set their hyperparameters as Table 1. The temperature is the only hyperparameter that impacts the performance gap. In preliminary experiments, we observe no significant impact for $\tau \in [0.1, 0.5]$, and we only report the result at $\tau = 0.3$.

5.1. Variance of the Gradient Estimators

First, let us study the variance of the gradient estimators.

Protocol. We generate a single random parameter setting and use it to generate a synthetic data set consisting of 10 examples of length 50. Then, we compute the gradient estimators using the whole data set 1000 times, which yields 1000 gradient estimates for each method. Finally, we compute the standard deviations of each element of the gradients, and report the mean of the standard deviations.

Result. The mean standard deviation of ∂SNN was 66.3, whereas that of SNN was 2.49×10^3. This clearly demonstrates that the variance of our estimator tends to be lower than that of the existing estimator.

5.2. Predictive Performance

The second experiment studies the predictive performance of the models learned by each of the methods compared.

[4]Intel Xeon Gold 6248 2.50GHz 48 cores and 192GB memory.

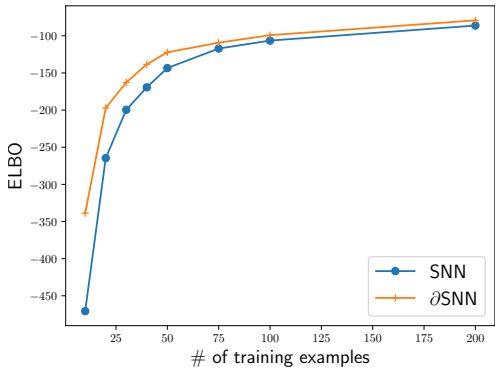

Figure 3. Predictive performance of SNN and ∂SNN.

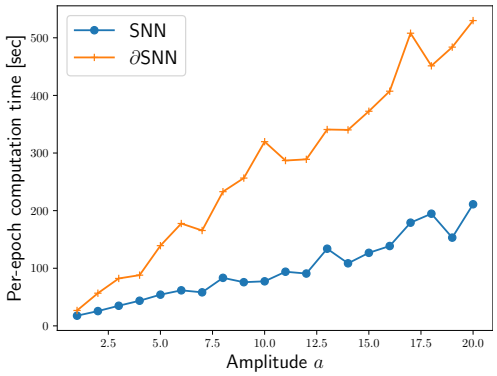

Figure 4. Per-epoch computation time of SNN and ∂SNN.

Protocol. We generate 24 random parameter settings, and consistently use them in this experiment. We aim to evaluate the performance gap between SNN and ∂SNN in different sizes of training sets. To this end, we execute the following, varying the size as $N_{\text{train}} = 10, 20, 30, 40, 50, 75\ 100, 200$, and for each parameter setting.

We generate training/test sets consisting of $N_{\text{train}}/100$ examples of length 50 respectively. SNN and ∂SNN are trained on the training set using AdaGrad (Duchi et al., 2011) with initial learning rate 0.05 for 10 epochs. We evaluate the predictive performance by computing ELBO (Equation (9)) on the test set. For fair comparison, we evaluate the performance of ∂SNN by transferring its parameters to the vanilla SNN[5]. By repeating this over 24 parameter settings, we obtain 24 ELBO scores. We report their mean as the performance of each method for each N_{train}.

Result. Figure 3 summarizes the experimental results. It clearly shows that ∂SNN consistently outperforms SNN especially in the small-sample regime, which supports the benefit of our low-variance estimator.

5.3. Computational Overhead

The last experiment studies computation overhead of ∂SNN over SNN. The computation time depends on the number of spikes, and the number of (hidden) spikes is proportional to a, the amplitude of the non-linearlity σ that maps the membrane potential into the conditional intensity function. In general, ∂SNN generates more hidden spikes than SNN because the thinning algorithm for the differentiable point process does not reject any of the candidate spikes. Therefore, we expect that ∂SNN requires more computation time than SNN. The purpose of this experiment is to measure the computational overhead of ∂SNN over SNN.

Protocol. We generate a single parameter setting, and generate a training set of 10 examples of length 50. We then set up both SNN and ∂SNN with amplitude $a = 1, 2, \ldots, 20$, resuting in 40 models to be trained. For each model, we measure the computation time of running 100 epochs, and obtain per-epoch computation time by averaging them.

Result. Figure 4 summarizes the experimental results. As is expected, ∂SNN requires 2.8 times more computation time than SNN on average. This result can be used as a reference for users to decide which gradient estimator to be employed. If a user can afford this overhead, our path-wise gradient estimator is recommended; otherwise, please consider to use the score function gradient estimator.

Note that we can improve the computation time of our method by introducing an adaptive upperbound $\bar{\lambda}$ in Algorithm 3, if it is a tigher upperbound than the fixed upperbound. We leave this improvement as future work.

6. Related Work

The present work is related to the communities of SNNs and point processes. Let us discuss our contributions to them.

6.1. Spiking Neural Networks

The most relevant work is the stochastic variational learning algorithm for SNNs (Jimenez Rezende & Gerstner, 2014). As discussed in Section 4.2.2, the difference is the gradient estimator. The authors used the score function gradient estimator, because the path-wise gradient estimator (which became popular by VAE (Kingma & Welling, 2014)) was not popular at that time and the reparameterization trick for point processes was not trivial. Our contribution is to develop a differentiable point process that enables us to derive the path-wise gradient estimator.

Less relevant but still worth mentioning are the line of work in learning algorithms for deterministic SNNs, where a neuron fires when the membrane potential exceeds a threshold. Although our technique cannot directly contribute to them,

[5]For better transfer, we decrease τ geometrically by ratio 0.95 at every epoch, which slightly improves the performance.

we believe it is worthwhile to compare the pros and cons of these different approaches for further development. Of a number of approaches proposed so far (Neftci et al., 2019), we introduce two inspiring studies.

SpikeProp (Bohte et al., 2000) is one of the earliest attempts to develop a learning algorithm for deterministic SNNs. SpikeProp uses backpropagation to minimize the difference between the target firing times $\{t_n^\star\}_{n=1}^N$ and the actual firing times $\{t_n\}_{n=1}^N$ of the network, *i.e.*, $\sum_{n=1}^N |t_n^\star - t_n|^2$. The gradient is approximated by assuming a linear relationship between the firing time and the membrane potential, which is valid only for a small learning rate.

Huh & Sejnowski (2018) propose a differentiable alternative to the threshold-based spike generation, which facilitates gradient computation. They employ a soft-threshold mechanism, and therefore, is differentiable without approximation. Another important contribution is that their model can handle not only spike trains but also a real-valued time-series. They use a readout network that maps spike trains from/into a real-valued time-series. This end-to-end formulation is significant towards practical applications of SNNs, and probabilistic SNNs should be equipped with this feature.

One interesting feature of probabilistic SNNs including our method is that both inference and learning algorithms can be executed naturally in an event-based manner without any discretization of time axis. This is in contrast to deterministic SNNs, where many learning algorithms require us to discretize the continuous-time dynamics for simulation.

6.2. Differentiable Point Processes

Our differentiable point process is significant in the community of point processes in that it largely expands the applicability of the reparameterization trick for point processes. Let us review the approaches to differentiable point processes, and discuss their pros and cons.

There are mainly three approaches to sample from point processes, and each of them can be used as a basis of differentiable point processes. The first approach (Shchur et al., 2020a) is to model the inter-event time conditioned on the past history by a log-normal mixture model, instead of modeling the conditional intensity function. Since it is straightforward to develop a reparameterizable sampling algorithm for the mixture model, the resultant point process is also reparameterizable. The second one is the inverse method (Rasmussen, 2018), which utilizes the fact that the inverse of the compensator $\Lambda^{[0,t]}$ can convert a unit-rate Poisson process into the point process with the corresponding conditional intensity function. Shchur et al. (2020b) propose a reparameterization trick based on the inverse method. The third one is the thinning algorithm, as we presented.

Of these three approaches, it is interesting to compare the second and the third approaches. When applying the inverse method (Shchur et al., 2020b) to computing ELBO, it is reported that the objective function contains discontinuous points, making optimization difficult. The discontinuity arises because time stamps of a realization are parameterized, and the algorithm involves a discrete decision whether a time stamp is less than T or not for termination. In contrast, Our differentiable point process does not suffer from it because not time stamps but marks are parameterized. In this sense, these two approaches are complementary.

When developing a path-wise gradient estimator for SNNs, only the third approach is feasible. The first approach is difficult to be applied because SNNs are modeled via the conditional intensity function, and the inter-event time distribution is not available in a closed form. The second approach is also difficult due to the lack of a closed-form expression of the inverse of the compensator. Our approach only assumes the existence of an upperbound of the conditional intensity function, and therefore, can be applied to SNNs. The assumption on the existence of a constant upperbound can be relaxed in the same way as Ogata's method (Ogata, 1981), which determines $\bar{\lambda}$ adaptively.

7. Conclusion and Future Work

We develop a path-wise gradient estimator for SNNs based on a differentiable point process. Given the experimental results in Section 5, we conclude that our estimator has lower variance than the existing one, which contributes to improve the learning capability.

Throughout this paper, we only focus on the dependency of the gradient estimator on learning capability, and we have not discussed about its practical applications. In the community of SNNs, however, an increasing number of studies have started to apply SNNs to real-world tasks (Shrestha & Orchard, 2018; Woźniak et al., 2020). One of the major concerns towards applying our method to real-world tasks is a method to convert real-valued data into/from spike trains. While there are a number of information encoding methods for spike trains, it is still an open problem which encoding is preferred. One interesting direction is to empirically and theoretically investigate the performance of different encoding methods and to understand their pros and cons.

Another limitation is the computational overhead as discussed in Section 5.3. While the probabilistic formulation can be simulated by an event-based manner, the gradient computation involves backpropagation through time (BPTT), whose complexity increases proportionally to the number of spikes. In addition to relying on the adaptive upperbound $\bar{\lambda}$, applying online BPTT calculation and its approximation techniques (Williams & Zipser, 1989) to SNNs may be an interesting research direction.

References

Bohte, S. M., Kok, J. N., and Poutré, H. L. SpikeProp: Backpropagation for Networks of Spiking Neurons. In *Proceedings of the 8th European Symposium on Artificial Neural Networks (ESANN 2000)*, pp. 419–424, 2000.

Bothe, S. M. The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, 2:195–206, 2004.

Daley, D. J. and Vere-Jones, D. *An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods*. Springer-Verlag New York, 2003.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 15324435.

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. *Neuronal Dynamics*. Cambridge University Press, 2014.

Huh, D. and Sejnowski, T. J. Gradient Descent for Spiking Neural Networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 1433–1443. Curran Associates, Inc., 2018.

Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the Fifth International Conference on Learning Representations*, 2017.

Jimenez Rezende, D. and Gerstner, W. Stochastic variational learning in recurrent spiking networks. *Frontiers in Computational Neuroscience*, 8:38, 2014. ISSN 1662-5188. doi: 10.3389/fncom.2014.00038.

Kajino, H. diffsnn, 2021. URL https://github.com/ibm-research-tokyo/diffsnn.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *Proceedings of the Second International Conference on Learning Representations*, 2014.

Lewis, P. A. W. and Shedler, G. S. Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979. doi: 10.1002/nav.3800260304.

Mächler, M. Accurately computing $\log(1 - \exp(-|a|))$ assessed by the Rmpfr package. Technical report, 2012.

Maddison, C. J., Mnih, A., and Teh, Y. W. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proceedings of the Fifth International Conference on Learning Representations*, 2017.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte Carlo Gradient Estimation in Machine Learning, 2019.

Neftci, E. O., Mostafa, H., and Zenke, F. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *IEEE Signal Processing Magazine*, 36 (6):51–63, 2019. doi: 10.1109/MSP.2019.2931595.

Ogata, Y. On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31, jan 1981. ISSN 1557-9654. doi: 10.1109/TIT.1981.1056305.

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. Optimal Spike-Timing-Dependent Plasticity for Precise Action Potential Firing in Supervised Learning. *Neural Computation*, 18(6):1318–1348, 2006. doi: 10.1162/neco.2006.18.6.1318.

Rasmussen, J. G. Lecture notes: Temporal point processes and the conditional intensity function. *arXiv preprint arXiv:1806.00221*, 2018.

Shchur, O., Biloš, M., and Günnemann, S. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020a.

Shchur, O., Gao, N., Biloš, M., and Günnemann, S. Fast and Flexible Temporal Point Processes with Triangular Maps. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 73–84. Curran Associates, Inc., 2020b.

Shrestha, S. B. and Orchard, G. SLAYER: Spike Layer Error Reassignment in Time. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

Williams, R. J. and Zipser, D. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280, 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270.

Woźniak, S., Pantazi, A., Bohnstingl, T., and Eleftheriou, E. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0187-0.