Exact Gradients for Stochastic Spiking Neural Networks Driven by Rough Signals

Christian Holberg
Department of Mathematics
University of Copenhagen
c.holberg@math.ku.dk

Cristopher Salvi
Department of Mathematics
Imperial College London
c.salvi@imperial.ac.uk

Abstract

We introduce a mathematically rigorous framework based on rough path theory to model stochastic spiking neural networks (SSNNs) as stochastic differential equations with event discontinuities (Event SDEs) and driven by càdlàg rough paths. Our formalism is general enough to allow for potential jumps to be present both in the solution trajectories as well as in the driving noise. We then identify a set of sufficient conditions ensuring the existence of pathwise gradients of solution trajectories and event times with respect to the network's parameters and show how these gradients satisfy a recursive relation. Furthermore, we introduce a general-purpose loss function defined by means of a new class of signature kernels indexed on càdlàg rough paths and use it to train SSNNs as generative models. We provide an end-to-end autodifferentiable solver for Event SDEs and make its implementation available as part of the `diffrax` library. Our framework is, to our knowledge, the first enabling gradient-based training of SSNNs with noise affecting both the spike timing and the network's dynamics.

1 Introduction

Stochastic differential equations exhibiting event discontinuities (Event SDEs) and driven by noise processes with jumps are an important modelling tool in many areas of science. One of the most notable examples of such systems is that of stochastic spiking neural networks (SSNNs). Several models for neuronal dynamics have been proposed in the computational neuroscience literature with the *stochastic leaky integrate-and-fire* (SLIF) model being among the most popular choices [19, 56]. In its simplest form, given some continuous input current i_t on $[0, T]$, the dynamics of a single SLIF neuron consist of an Ornstein-Uhlenbeck process describing the membrane potential as well as a threshold for spike triggering and a resetting mechanism [33]. In particular, between spikes, the dynamics of the membrane potential v_t is given by the following SDE

$$dv_t = \mu\,(i_t - v_t)\,dt + \sigma dB_t, \tag{1}$$

where $\mu > 0$ is a parameter and B_t is a standard Brownian motion. The neuron spikes whenever the membrane potential v hits the threshold $\psi > 0$ upon which v is reset to 0. Alternatively, one can model the spike times as a Poisson process with intensity $\lambda : \mathbb{R} \to \mathbb{R}_+$ depending on the membrane potential v_t. A common choice is $\lambda(v) = \exp((v - \psi)/\beta)$ [50, 26, 27, 24].

A notorious issue for calibrating Event SDEs such as SSNNs is that the implicitly defined event discontinuities, e.g., the spikes, make it difficult to define derivatives of the solution trajectories and of the event times with respect to the network's parameters using classical calculus rules. This issue is exacerbated when the dynamics are stochastic in which case the usual argument relying on the implicit function theorem, used for instance in [6, 25], is no longer valid.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

1.1 Contributions

In this paper, we introduce a mathematically rigorous framework to model SSNNs as SDEs with event discontinuities and driven by càdlàg rough paths, without any prior knowledge of the timing of events. The mathematical formalism we adopt is that of *rough path theory* [38], a modern branch of stochastic analysis providing a robust solution theory for stochastic dynamical systems driven by noisy, possibly discontinuous, *rough signals*. Although Brownian motion is a prototypical example, these signals can be far more irregular (or rougher) than semimartingales [17, 16, 39].

Equipped with this formalism, we proceed to identify sufficient conditions under which the solution trajectories and the event times are differentiable with respect to the network's parameters and obtain a recursive relation for the exact pathwise gradients in Theorem 3.2. This is a strict generalization of the results presented in [6] and [25] which only deal with ordinary differential equations (ODEs). Furthermore, we define *Marcus signature kernels* as extensions of continuous signature kernels [52] to càdlàg rough paths and show their characteristicness. We then make use of this class of kernels indexed on discontinuous trajectories to define a general-purpose loss function enabling the training of SSNNs as generative models. We provide an end-to-end autodifferentiable solver for Event SDEs (Algorithm 1) and make its implementation available as part of the `diffrax` library [28].

Our framework is, to our knowledge, the first allowing for gradient-based training of a large class of SSNNs where a noise process can be present in both the spike timing and the network's dynamics. In addition, we believe this work is the first enabling the computation of exact gradients for classical SNNs whose solutions are approximated via a numerical solver (not necessarily based on a Euler scheme). In fact, previous solutions are based either on surrogate gradients [46] or follow an optimise-then-discretise approach deriving adjoint equations [56], the latter yielding exact gradients only in the scenario where solutions are available in closed form and not approximated via a numerical solver.[1] Finally, we discuss how our results lead to bioplausible learning algorithms akin to *e-prop* [2].

2 Related work

Neural stochastic differential equations (NSDEs) The intersection between differential equations and deep learning has become a topic of great interest in recent years. A neural ordinary differential equation (NODE) is an ODE of the form $\mathrm{d}y_t = f_\theta(y_t)dt$ started at $y_0 \in \mathbb{R}^e$ using a parametric Lipschitz vector field $f_\theta : \mathbb{R}^e \to \mathbb{R}^e$, usually given by a neural network [5]. Similarly, a neural stochastic differential equation (NSDE) is an SDE of the form $\mathrm{d}y_t = \mu_\theta(y_t)dt + \sigma_\theta(y_t)dB_t$ driven by a d-dimensional Brownian motion B, started at $y_0 \in \mathbb{R}^e$, and with parametric vector field $\mu_\theta : \mathbb{R}^e \to \mathbb{R}^e$ and $\sigma_\theta : \mathbb{R}^e \to \mathbb{R}^{e\times d}$ that are Lip^1 and $\mathrm{Lip}^{2+\epsilon}$ continuous respectively[2]. Rough path theory offers a way of treating ODEs, SDEs, and more generally differential equations driven by signals or arbitrary (ir)regularity, under the unified framework of *rough differential equations* (RDEs) [44, 22]. For an account on applications of rough path theory to machine learning see [4, 14, 51].

Training techniques for NSDEs Training a NSDE amounts to minimising over model parameters an appropriate notion of statistical divergence between a distribution of continuous trajectories generated by the NSDE and an empirical distribution of observed sample paths. Several approaches have been proposed in the literature, differing mostly in the choice of discriminating divergence. SDE-GANs, introduced in [29], use the 1-Wasserstein distance to train a NSDE as a Wasserstein-GAN [1]. Latent SDEs [36] train a NSDE with respect to the KL divergence via variational inference and can be interpreted as variational autoencoders. In [23] the authors propose to train NSDEs non-adversarially using a class of maximum mean discrepancies (MMD) endowed with signature kernels [30, 52]. Signature kernels are a class of characteristic kernels indexed on continuous paths that have received increased attention in recent years thanks to their efficiency for handling path-dependent problems [35, 54, 10, 53, 9, 48, 41]. For a treatment of this topic we refer the interested reader to [4, Chapter 2]. These kernels are not applicable to sample trajectories of SSNNs because of the lack of continuity.

Backpropagation through NSDEs Once a choice of discriminator has been made, training NSDEs amounts to perform backpropagation through the SDE solver. There are several ways to do this. The

[1] The point here is actually a little more subtle. It is in fact possible to obtain exact gradients using the adjoint method as long as one uses reversible (or adjoint) solvers in the forward pass.

[2] These are standard regularity conditions to ensure existence and uniqueness of a strong solution.

first option is simply to backpropagate through the solver's internal operations. This method is known as *discretise-then-optimise*; it is generally speaking fast to evaluate and produces accurate gradients, but it is memory-inefficient, as every internal operation of the solver must be recorded. A second approach, known as *optimise-then-discretise*, computes gradients by deriving a backwards-in-time differential equation, the *adjoint equation*, which is then solved numerically by another call to the solver. Not storing intermediate quantities during the forward pass enables model training at a memory cost that is constant in depth. Nonetheless, this approach produces less accurate gradients and is usually slower to evaluate because it requires recalculating the forward solutions to perform the backward pass. A third way of backpropagating through NDEs is given by *algebraically reversible solvers*, offering both memory and accuracy efficiency. We refer to [28] for further details.

Differential equations with events Many systems are not adequately modelled by continuous differential equations because they experience jump discontinuities triggered by the internal state of the system. Examples include a bouncing ball or spiking neurons. Such systems are often referred to as *(stochastic) hybrid systems* [21, 37]. When the differential equation is an ODE, there is a rich literature on sensitivity analysis aimed at computing derivatives using the implicit function theorem [11, 12]. If, additionally, the vector fields describing the hybrid system are neural networks, [6] show that NODEs solved up until first event time can be implemented as autodifferentiable blocks and [25] derive the corresponding adjoint equations. Nonetheless, none of these works cover the more general setting of SDEs. The only work, we are familiar with, dealing with sensitivity analysis in this setting is [47], although focused on the problem of optimal control.

Training techniques for SNNs Roughly speaking, these works can be divided into two strands. The first, usually referred to as *backpropagation through time* (BPTT), starts with a Euler approximation of the SNN and does backpropagation by unrolling the computational graph over time; it then uses surrogate gradients as smooth approximations of the gradients of the non-differentiable terms. [58, 46, 40]. This approach is essentially analogous to discretise-then-optimise where the backward pass uses custom gradients for the non-differentiable terms. The second strand computes exact gradients of the spike times using the implicit function theorem. These results are equivalent to optimise-then-discretise and can be used to define adjoint equations as in [56] or to derive forward sensitivities [34]. However, we note that, unless solution trajectories and spike times of the SNN are computed exactly, neither method provides the actual gradients of the implemented solver. Furthermore, the BPTT surrogate gradient approach only covers the Euler approximation whereas many auto-differentiable differential equation solvers are available nowadays, e.g. in `diffrax`. Finally, there is a lot of interest in developing bioplausible learning algorithms where weights can be updated locally and in an online fashion. Notable advances in this direction include [2, 57]. To the best of our knowledge, none of these works cover the case of stochastic SNNs where the neuronal dynamics are modeled as SDEs instead of ODEs.

3 Stochastic spiking neural networks as Event SDEs

We shall in this paper be concerned with SDEs where solution trajectories experience jumps triggered by implicitly defined events, dubbed *Event SDEs*. The prototypical example that we come back to throughout is the SNN model composed of SLIF neurons. Here the randomness appears both in the inter-spike dynamics as well as in the firing mechanism. To motivate the general definitions and concepts we start with an informal introduction of SSNNs.

3.1 Stochastic spiking neural networks

To achieve a more bioplausible model of neuronal behaviour, one can extend the simple deterministic LIF model by adding two types of noise: a diffusion term in the differential equation describing inter-spike behaviour [33] and stochastic firing [50, 27]. That is, the potential is modelled by eq. (1). Instead of firing exactly when the membrane potential hits a set threshold, we model the spike times (event times) by an inhomogenous Poisson process with intensity $\lambda : \mathbb{R}^e \to \mathbb{R}_+$ which is assumed to be bounded by some constant $C > 0$. This can be phrased as an Event SDE (note that this is essentially the reparameterisation trick) by introducing the additional state variable s_t satisfying

$$ds_t = \lambda(v_{t-})dt, \quad s_0 = \log u$$

where $u \sim \text{Unif}(0,1)$. The neuron spikes whenever s_t hits 0 from below at which point the membrane potential is reset to a resting level and we sample a new initial condition for s_t. We can denote this first spike time by τ_1 and repeat the procedure to generate a sequence of spike times $\tau_1 < \tau_2 < ...$ In practice, we reinitialize s_t at $\log u - \alpha$ for some $\alpha > 0$. It can then be shown that

$$\mathbb{P}\left(t < \tau_{n+1} | \mathcal{F}_{\tau_n}\right) = \min\left\{1, \exp\left(\alpha - \int_{\tau_n}^{t} \lambda(v_{t-})dt\right)\right\} \quad \text{for } t \in [\tau_n, \tau_{n+1}).$$

It follows that $\tau_{n+1} - \tau_n \geq \alpha/C$ a.s., i.e. α controls the refractory period after spikes, a large value indicating a long resting period.

We can then build a SSNN by connecting such SLIF neurons in a network. In particular, apart from the membrane potential, we now also model the input current of each neuron as affected by the other neurons in the network. Let $K \geq 1$ denote the total number of neurons. We model neuron $k \in [K]$ be the three dimensional vector $y^k = (v^k, i^k, s^k)$ the dynamic of which in between spikes is given by

$$dv_t^k = \mu_1\left(i_t^k - v_t^k\right)dt + \sigma_1 dB_t^k, \quad di_t^k = -\mu_2 i_t^k dt + \sigma_2 dB_t^k, \quad ds_t^k = \lambda(v_t^k; \xi)dt, \tag{2}$$

where B^k is a standard two-dimensional Brownian motion, $\sigma = (\sigma_1, \sigma_2) \in \mathbb{R}^{2\times 2}$, $\mu = (\mu_1, \mu_2) \in \mathbb{R}^2$, and $\lambda(\cdot;\xi) : \mathbb{R} \to \mathbb{R}_+$ is an intensity function. As before, neuron k fires (or spikes) whenever s^k hits zero from below. Apart from resetting the membrane potential, this event also causes spikes to propagate through the network in a such a way that a spike in neuron k will increment the input current of neuron j by w_{kj}. Here $w \in \mathbb{R}^{K\times K}$ is a matrix of weights representing the synaptic weights in the neural network. If one is only interested in specific network architectures such as, e.g., feed-forward, this can be achieved by fixing the appropriate entries in w at 0.

As presented here, there is no way to model information coming into the network. But this would only require a minor change. Indeed, by adding a suitable control term to eq. (2) we can model all relevant scenarios. Since this does not change the theory in any meaningful way (the general theory in Appendix B covers RDEs so an extra smooth control is no issue), we only discuss the more simple model given without any additional input currents.

3.2 Model definition

Definition 3.1 (Event SDE)**.** *Let $N \in \mathbb{N}$ be the number of events. Let $y_0 \in \mathbb{R}^e$ be an initial condition. Let $\mu : \mathbb{R}^e \to \mathbb{R}^e$ and $\sigma : \mathbb{R}^e \to \mathbb{R}^{e\times d}$ be the drift and diffusion vector fields. Let $\mathcal{E} : \mathbb{R}^e \to \mathbb{R}$ and $\mathcal{T} : \mathbb{R}^e \to \mathbb{R}^e$ be an event and transition function respectively. We say that $\left(y, (\tau_n)_{n=1}^N\right)$ is a solution to the Event SDE parameterised by $(y_0, \mu, \sigma, \mathcal{E}, \mathcal{T}, N)$ if $y_T = y_T^N$,*

$$y_t = \sum_{n=0}^{N} y_t^n \mathbf{1}_{[\tau_n, \tau_{n+1})}(t), \quad \tau_n = \inf\left\{t > \tau_{n-1} : \mathcal{E}(y_t^{n-1}) = 0\right\}, \tag{3}$$

with $\mathcal{E}(y_{\tau_n}^n) \neq 0$ and

$$dy_t^0 = \mu(y_t^0)dt + \sigma(y_t^0)dB_t, \quad \textit{started at } \; y_0^0 = y_0, \tag{4}$$

$$dy_t^n = \mu(y_t^n)dt + \sigma(y_t^n)dB_t, \quad \textit{started at } \; y_{\tau_n}^n = \mathcal{T}\left(y_{\tau_n}^{n-1}\right), \tag{5}$$

where B_t is a d-dimensional Brownian motion and (4), (5) are Stratonovich SDEs.

In words, we initialize the system at y_0, evolve it using (4) until the first time τ_1 at which an event happens $\mathcal{E}(y_{\tau_1}^0) = 0$. We then transition the system according to $y_{\tau_1}^1 = \mathcal{T}\left(y_{\tau_1-}^0\right)$ and evolve it according to (5) until the next event is triggered. We note that Definition 3.1 can be generalised to multiple event and transition functions. Also, the transition function can be randomised by allowing it to have an extra argument $u \sim \text{Unif}([0,1])$. As part of the definition we require that there are only finitely many events and that an event is not immediately triggered upon transitioning.

Existence of strong solutions to Event SDEs driven by continuous semimartingales has been studied in [31, Theorem 5.2] and [32]. Under sufficient regularity of μ and σ, a unique solution to (4) exists. We need the following additional assumptions:

Assumption 3.1. There exists $c > 0$ such that for all $s \in (0,T)$ and $a \in \text{im } \mathcal{T}$ it holds that $\inf\{t > s \; : \; \mathcal{E}(y_t) = 0\} > c$ where y_t is the solution to 4 started at $y_s = a$

Assumption 3.2. It holds that $\mathcal{T}(\ker \mathcal{E}) \cap \mathcal{E} = \emptyset$.

Assumptions 3.1 and 3.2 simply ensure that an event cannot be triggered immediately upon transitioning. This holds in most settings of interest. For example, for the usual deterministic LIF neuron im $\mathcal{T} = 0$ and ker $\mathcal{E} = 1$ and the duration of the refractory period is directly linked to c in Assumption 3.1.

Theorem 3.1 (Theorem 5.2, [31])**.** *Under Assumptions 3.1-3.2 and with $\mu \in \text{Lip}^1$ and $\sigma \in \text{Lip}^\gamma$ for $\gamma > 2$, there exists a unique solution $(y, (\tau_n)_{n=1}^N)$ to the Event SDE of Definition 3.1.*

The definitions and results of this section can be extended to differential equations driven by random rough paths, and in particular, to cases where the driving noise exhibits jumps. In the latter case, it is important to note that the resulting Event SDE will exhibit two types of jumps: the ones given apriori by the driving noise and the ones that are implicitly defined through the solution (what we call *events*). In fact, we develop the main theory of Event RDEs in Appendix A in the more general setting of RDEs driven by càdlàg rough paths. The rough path formalism enables a unified treatment of differential equations driven by noise signals of arbitrary (ir-)regularity, and makes all proofs simple and systematic. In particular, it allows us to handle cases where the diffusion term is driven by a finite activity Lévy process (e.g, a homogeneous Poisson processes highly relevant in the context of SNNs).

3.3 Backpropagation

We are interested in optimizing a continuously differentiable loss function L whose input is the solution of a parameterised Event SDE. As for Neural ODEs, the vector fields, μ, σ, and the event and transition functions $\mathcal{E}, \mathcal{T}$, might depend on some learnable parameters θ. We can move the parameters θ of the Event RDE inside the initial condition y_0 by augmenting the dynamics with the additional state variable θ_t satisfying $d\theta_t = 0$ and $\theta_0 = \theta$. Thus, as long as we can compute gradients with respect to y_0, these will include gradients with respect to such parameters. We then require the gradients $\partial_{y_0} L$, if they exist. For this, we need to be able to compute the Jacobians $\partial y_t^n := \partial_{y_0} y_t^n$ of the inter-event flows associated to the dynamics of y_t^n and the derivatives $\partial \tau_n := \partial_{y_0} \tau_n$. We assume that the event and transition functions $\mathcal{E}$ and $\mathcal{T}$ are continuously differentiable.

Apriori, it is not clear under what conditions such quantities exist and even less how to compute them. This shall be the focus of the present section. We will need the following running assumptions.

Assumption 3.3. $\sigma(\mathcal{T}(y)) - \nabla\mathcal{T}(y)\sigma(y) = 0$ for all $y \in \ker \mathcal{E}$.

Assumption 3.4. $\nabla\mathcal{E}(y)\sigma(y) = 0$ for all $y \in \ker \mathcal{E}$.

Assumption 3.5. $\nabla\mathcal{E}(y)\mu(y) \neq 0$ for all $y \in \ker \mathcal{E}$.

Assumption 3.4 and 3.5 ensure that the event times are differentiable. Intuitively, they state that the event condition is hit only by the drift part of the solution. Assumption 3.4 holds for example if the event functions depend only on a smooth part of the system. Assumption 3.3 is what allows us to differentiate through the event transitions.

Theorem 3.2. *Let Assumptions 3.1-3.5 be satisfied and $(y, (\tau_n)_{n=1}^N)$ the solution to the Event SDE parameterized by $(y_0, \mu, \sigma, \mathcal{E}, \mathcal{T}, N)$. Then, almost surely, for any $n \in [N]$, the derivatives $\partial\tau_n$ and the Jacobians ∂y_t^n exist and admit the following recursive expressions*

$$\partial\tau_n = -\frac{\nabla\mathcal{E}(y_{\tau_n}^{n-1})\partial y_{\tau_n}^{n-1}}{\nabla\mathcal{E}(y_{\tau_n}^{n-1})\mu(y_{\tau_n}^{n-1})} \tag{6}$$

$$\partial y_t^n = (\partial_{y_{\tau_n}^n} y_t^n)\left[\nabla\mathcal{T}(y_{\tau_n}^{n-1})\partial y_{\tau_n}^{n-1} - \left(\mu(y_{\tau_n}^n) - \nabla\mathcal{T}(y_{\tau_n}^{n-1})\mu(y_{\tau_n}^{n-1})\right)\partial\tau_n\right]. \tag{7}$$

where ∂y_t^n and $\partial\tau_n$ are the total derivatives of y_n^t and τ_n with respect to the initial condition y_0, $\partial_{y_{\tau_n}^n} y_t^n$ denotes the partial derivative of the flow map of eq. (5) *with respect to its initial condition, and $\nabla\mathcal{T} \in \mathbb{R}^{e\times e}$ and $\nabla\mathcal{E} \in \mathbb{R}^{1\times e}$ are the Jacobians of $\mathcal{T}$ and $\mathcal{E}$.*

Remark 3.1. If the diffusion term is absent we recover the gradients in [6]. In this case, the assumptions of the theorem are trivially satisfied. Note however, that the result, as stated here, is slightly different since we are considering repeated events.

Remark 3.2. The recursive nature of (6) - (7) suggest a way to update gradients in an online fashion by computing the forward sensitivity along with the state of the Event SDE. In traditional machine

learning applications (e.g. NDEs) forward mode automatic differentiation is usually avoided due to the fact that the output dimension tends to be orders of magnitude smaller than the number of parameters [28]. However, for (S)SNNs this issue can be partly avoided as discussed in Section 4.4.

Returning now to the SSNN model introduced in Section 3.1 we find that it is an Event SDE with K different event functions given by $\mathcal{E}_k(y) = s^k$ and corresponding transition functions given by

$$\mathcal{T}_k(y) = \left(\mathcal{T}_k^1(y^1), \dots, \mathcal{T}_k^K(y^K)\right)$$

where $\mathcal{T}_k^j(y^j) = (v^j, i^j + w_{kj}, s^j)$ if $j \neq k$ and $\mathcal{T}_k^k(y^k) = (v^k - v_{reset}, i^k, \log u - \alpha)$ where $v_{reset} > 0$ is a constant determining by what amount the membrane potential is reset. The addition of the constant $\alpha > 0$ controlling the refractory period ensures that Assumption 3.2 and 3.2 are satisfied. Stochastic firing smooths out the event triggering so that Assumption 3.5 and 3.4 hold. Finally, one can check that the combination of constant diffusion terms and the given transition functions satisfies Assumptions 3.3. Note that setting v_t^k exactly to 0 upon spiking would break Assumption 3.3. If one is interested in such a resetting mechanism it suffices to pick a diffusion term $\sigma_1(y^k)$ that satisfies $\sigma(0) = 0$. To sum up, solutions (in the sense of Def. 3.1) of the SSNNs exist and are unique. In addition, the trajectories and spike times are almost surely differentiable satisfying (6) and (7).

3.4 Numerical solvers

Theorem 3.2 gives an expression for the gradients of the event times as well as the Event SDE solution. In practice, analytical expressions for gradients are often not available and one has to resort to numerical solvers. Three solutions suggest themselves:

1. There are multiple autodifferentiable differential equation solvers (such as `diffrax` [28]) that provide differentiable numerical approximations of the flows $\partial_{y_{\tau_n}^n} y_t^n$. We shall write $\text{SDESolve}(y_0, \mu, \sigma, s, t)$ for a generic choice of such a solver. Furthermore, if $\text{RootFind}(y_0, f)$ is a differentiable root finding algorithm (here $f : (y, t) \mapsto \mathbb{R}$ should be differentiable in both arguments and $\text{RootFind}(y_0, f)$ returns $t^* \in \mathbb{R}$ such that $f(y_0, t^*) = 0$), then we can define a differentiable map $E : y_0 \mapsto y^*$ by

$$t^* = \text{RootFind}\left(y_0, \mathcal{E}(\text{SDESolve}\left(\cdot, \mu, \sigma, s, \cdot\right))\right), \quad y^* = \text{SDESolve}\left(y_0, \mu, \sigma, s, t^*\right).$$

   Consequently, $\text{EventSDESolve}(y_0, \mu, \sigma, \mathcal{E}, \mathcal{T}, N)$ can be implemented as subsequent compositions of $\mathcal{T} \circ E$ (see Algorithm 1). This is a discretise-then-optimise approach [28].

2. Alternatively, one can use the formulas (6) and (7) directly as a replacement of the derivatives. This is the approach taken in e.g. [6]. To be precise, one would replace all the derivatives of the flow map (terms of the sort $\partial_{y_{\tau_n}^n} y_t^n$) with the derivatives of the given numerical solver. This approach is a solution between discretise-then-optimise and optimise-then-discretise.

3. Finally, one could apply the adjoint method (or optimise-then-discretise) as done for deterministic SNNs in [56] by deriving the adjoint equations. These adjoint equations define another SDE with jumps which is solved backwards in time. Between events the dynamics are exactly as in the continuous case so one just needs to specify the jumps of the adjoint process. This can be done by referring to (6) and (7).

Remark 3.3. One thing to be careful of with the discretise-then-optimise approach is that the SDE solver will compute time derivatives in the backward pass, although the modelled process is not time differentiable. Assumptions 3.4 and 3.3 should in principle guarantee that these derivatives cancel out (see Appendix B), yet this might not necessarily happen at the level of the numerical solver because of precision issues. This is essentially due to the fact that approximate solutions provided by numerical solvers are in general not *flows*. Thus, when the path driving the diffusion term is very irregular, the gradients can become unstable. In practice we found this could be fixed by setting the gradient with respect to time of the driving Brownian motion to 0 and picking a step size sufficiently small.

Remark 3.4. In the context of SNNs, Algorithm 1 is actually a version of *exact backpropagation through time* (BPTT) of the unrolled numerical solution. Contrary to popular belief, this illustrates that one can compute exact gradients of numerical approximations of SNNs without the need to resort to surrogate gradient functions. Of course, this does not alleviate the so-called *dead neuron problem*. However, this ceases to be a problem when stochastic firing is introduced. In fact, surrogate gradients can be related to stochastic firing mechanisms and expected gradients [20].

Remark 3.5. One the one hand, the EventSDESolve algorithm as presented here scales poorly in the number of events since it requires doing a full SDESolve and an additional RootFind each time an event occurs. This problem becomes especially prevalent for SSNNs with a large number of neurons since in this case an event is triggered every time a single neuron spikes and the inter-spike SDE that needs to be solved is high-dimensional. On the other hand, there are multiple ways to mitigate this issue. Firstly, one could relax the root-finding step and simply trigger a spike as soon as $e \geq 0$ and take this as the spike time. For the backward pass one could then solve the adjoint equations (for which you need need to store the spike times in the forward pass). The resulting algorithm would be similar to the one derived in [55] for deterministic SNNs. Secondly, for special architectures such as a feed-forward network, given the spikes from the previous layer, one could solve the EventSDE for each neuron in the current layer independently of all other neurons. This would imply that a forward (or backward) pass of the entire SSNN scales as $O(KS)$ where S is the cost of the forward (or backward) pass of a single neuron and K is the number of neurons.

Algorithm 1 EventSDESolve

Input $y_0, \mu, \sigma, \mathcal{E}, \mathcal{T}, N, t_0, \Delta t, T$

$y \leftarrow y_0$
$n \leftarrow 0$
$e \leftarrow \mathcal{E}(y)$
while $n < N$ **and** $t_0 < T$ **do**
  while $e < 0$ **do** ▷ We assume for simplicity that $e \leq 0$
    $y_0 \leftarrow y$
    $y \leftarrow \text{SDESolveStep}(y_0, \mu, \sigma, t_0, \Delta t)$
    $t_0 \leftarrow t_0 + \Delta t$
    $e \leftarrow \mathcal{E}(y)$ ▷ Update value of event function
  end while
  $t^*_{n+1} \leftarrow \text{RootFind}\left(y_0, \mathcal{E}(\text{SDESolveStep}\left(\cdot, \mu, \sigma, t_0 - \Delta t, \cdot\right))\right)$ ▷ Find exact event time
  $y^*_{n+1} \leftarrow \text{SDESolveStep}(y_0, \mu, \sigma, t_0 - \Delta_t, t^*_{n+1})$ ▷ Compute state at event time
  $y \leftarrow \mathcal{T}(y^*_{n+1})$ ▷ Apply transition function
  $n \leftarrow n + 1$
end while
Return $(t^*_n)_{n \leq N}, y$

4 Training stochastic spiking neural networks

4.1 A loss function based on signature kernels for càdlàg paths

To train SSNNs we will adopt a similar technique as in [23], where the authors propose to train NSDEs non-adversarially using a class of maximum mean discrepancies (MMD) endowed with signature kernels [52] indexed on spaces of continuous paths as discriminators. However, as we mentioned in the introduction, classical signature kernels are not directly applicable to the setting of SSNNs as the solution trajectories not continuous. To remedy this issue, in Appendix C, we generalise signature kernels to *Marcus signature kernels* indexed on discontinuous (or càdlàg) paths. We note that our numerical experiments only concern learning from spike trains, which are càdlàg paths of bounded variation. Yet, the Marcus signature kernel defined in Appendix C can handle more general càdlàg rough paths.

The main idea goes as follows. If x is a càdlàg path, one can define the *Marcus signature* $S(x)$ in the spirit of Marcus SDEs [42, 43] as the signature of the *Marcus interpolation* of x. The general construction is given in Appendix A. The *Marcus signature kernel* is defined as the inner product $k(x, y) = \langle S(x), S(y) \rangle$ of Marcus signatures $S(x), S(y)$ of two càdlàg paths x, y. As stated in the first part of Theorem C.1, this kernel is characteristic on regular Borel measures supported on compact sets of càdlàg paths. In particular, this implies that the resulting *maximum mean discrepancy* (MMD)

$$d_k(\mu, \nu)^2 = \mathbb{E}_{x,x' \sim \mu} k(x, x') - 2\mathbb{E}_{x,y \sim \mu \times \nu} k(x, x') + \mathbb{E}_{y,y' \sim \nu} k(y, y')$$

satisfies the property $d_k(\mu, \nu)^2 = 0 \iff \mu = \nu$ for any two compactly supported measures μ, ν.

Nonetheless, characteristicness ceases to hold when one considers measures on càdlàg paths that are not compactly supported. In [8] the authors address this issue for continuous paths by using the

so-called *robust signature*. They introduce a *tensor normalization* Λ ensuring that the range of the robust signature $\Lambda \circ S$ remains bounded. The *robust signature kernel* is then defined as the inner product $k_\Lambda(x, y) = \langle \Lambda \circ S(x), \Lambda \circ S(y) \rangle$. This normalization can be applied analogously to the *Marcus signature* resulting in a *robust Marcus signature kernel*. In the second part of Theorem C.1, we prove characteristicness of k_Λ for possibly non-compactly supported Borel measures on càdlàg paths. The resulting MMD is denoted by d_{k_Λ}.

There are several ways of evaluating signature kernels. The most naive is to simply truncate the signatures at some finite level and then take their inner product. Another amounts to solve a path-dependent wave equation [52]. Our experiments are compatible with both of these methods.

Given a collection of observed càdlàg trajectories $\{x^i\}_{i=1}^m \sim \mu^{\text{true}}$ sampled from an underlying unknown target measure μ^{true}, we can train an Event SDE by matching the generated càdlàg trajectories $\{y^i\}_{i=1}^n \sim \mu^\theta$ using an unbiased empirical estimator of d_k (or d_{k_Λ}), i.e. minimising over the parameters θ of the Event SDE the following loss function

$$\mathcal{L} = \frac{1}{m(m-1)} \sum_{j \neq i} k(x^i, x^j) - \frac{2}{mn} \sum_{i,j} k(x^i, y^j) + \frac{1}{n(n-1)} \sum_{j \neq i} k(y^i, y^j).$$

In the context of SSNNs, the observed and generated trajectories x^i's and y^i's correspond to spike trains, which are càdlàg paths of bounded variation.

4.2 Input current estimation

The first example is the simple problem of estimating the constant input current $c > 0$ based on a sample of spike trains in the single SLIF neuron model,

$$dv_t = \mu(c - v_t)dt + \sigma dB_t, \quad ds_t = \lambda(v_t)dt,$$

where $\lambda(v) = \exp(5(v - 1)$, $\mu = 15$ and σ varies. Throughout we fix the true $c = 1.5$ and set $v_{reset} = 1.4$ and $\alpha = 0.03$. We run stochastic gradient descent for 1500 steps for two choices of the diffusion constant σ. The loss function is the signature kernel MMD between a simulated batch and the sample of spike trains.[3]. The test loss is the mean absolute error between the first three average spike times. Results are given in Fig. 1. For additional details regarding the experiments, we refer to Appendix E.

In all cases backpropagation through Algorithm 1 is able to learn the underlying input current after around 600 steps up to a small estimation error. In particular, the convergence is fastest for the largest sample size and the true c is recovered for both levels of noise.

4.3 Synaptic weight estimation

Next we consider the problem of estimating the weight matrix in a feed-forward SSNN with input dimension 4, 1 hidden layer of dimension 16, and output dimension 2. The rest of the parameters are fixed throughout. We run stochastic gradient descent for 1500 steps with a batch size of 128 and for a sample size of $256, 512$, and 1024 respectively. Learning rate is decreased from 0.003 to 0.001 after 1000 steps. The results are given in Fig. 2 in Appendix E. For a sample size of 512 and 1024 we are able to reach a test loss of practically 0, that is, samples from the learned model and the underlying model are more or less indistinguishable. Also, in all cases the estimated weight matrix approaches the true weight matrix. Interestingly, for the largest sample size, the model reaches the same test loss as the model trained on a sample size of 512, but their estimated weight matrices differ significantly.

4.4 Online learning

In the case of SSNNs, equations (6)-(7) lead to a formula for the forward sensitivity where any propagation of gradients between neurons only happens at spike times and only between connected neurons (see Proposition D.1). Since the forward sensitivities are computed forward in time together with the solution of the SNN, gradients can be updated online as new data appears. As a result, between spikes of pre-synaptic neurons, we can update the gradient flow of the membrane potential

[3]For simplicity we only compute an approximation of the true MMD by truncating the signatures at depth 3 and taking the average across the batch/sample size.

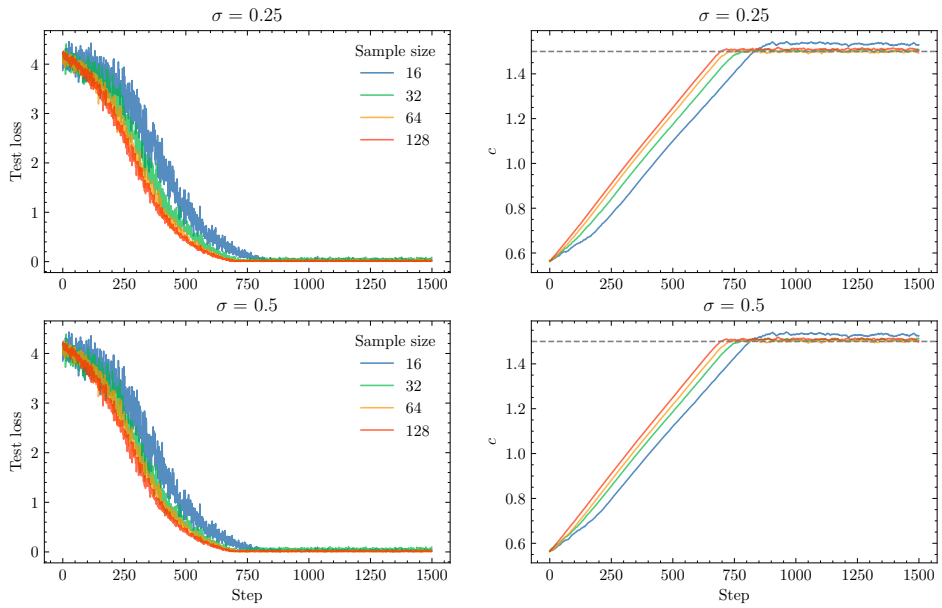

Figure 1: Test loss and c estimate across four sample sizes and for two levels of noise σ. On the left: MAE for the three first average spike times on a hold out test set. On the right: estimated value of c at the current step.

and input current of each neuron using information exclusively from that neuron. For general network structures and loss functions, however, this implies that each neuron needs to store on the order of K^2 gradient flows (one for each weight in the network).

On the other hand, if the adjacency matrix of the weight matrix forms a directed acyclic graph (DAG), three-factor Hebbian learning rules like those in [57, 2] are easily derived from Proposition D.1. For simplicity, consider the SNN consisting of deterministic LIF neurons and let N_t^k denote the spike train of neuron k, i.e., N_t^k is càdlàg path equal to the number of spikes of neuron k at time t. We let $\tau^k(t)$ (or τ^k for short) denote the last spike of neuron k before time t. We shall assume that the instantaneous loss function L_t depends only on the most recent spike times $\tau^1, \dots, \tau^K$. Then,

$$\partial_{w_{jk}} L_t = \partial_{\tau^k} L_t \frac{a^{jk}_{\tau^k}}{\mu_1(v^k_{\tau^k} - i^k_{\tau^k})}$$

where a_t^{jk} is the *eligibility trace* and the first term can be viewed as a *global modulator*, that is, a top-down learning signal propagating the error from the output neurons.[4] The eligibility trace satisfies

$$da_t^{jk} = \mu_1 \left(b_t^{jk} - a_t^{jk} \right) dt + \frac{v_{reset} a_t^{jk}}{\mu_1(i_t^k - v_t^k)} dN_t^k, \quad db_t^{jk} = -\mu_2 b_t^{jk} + dN_t^j,$$

where the dN terms are to be understood in the Riemann-Stieltjes sense. In other words, the eligibility trace can be updated exclusively from the activity of the pre- and post-synaptic neurons. We note the similarity to the results derived in [2] only our result gives the exact gradients with no need to introduce surrogate gradient functions. A similar equation for deterministic SNNs was derived in [49] (see, in particular, Chapter 5). For general network structures one can use the eligibility traces as proxies for the the true derivatives $\partial_{w_{ij}} \tau^k$.

5 Conclusion

We introduced a mathematical framework based on rough path theory to model SSNNs as SDEs exhibiting event discontinuities and driven by càdlàg rough paths. After identifying sufficient

[4] Note that in the case of stochastic SNNs this term is not necessarily well-defined since semi-martingales are in general not differentiable wrt. time.

conditions for differentiability of solution trajectories and event times, we obtained a recursive relation for the pathwise gradients in Theorem 3.2, generalising the results presented in [6] and [25] which only deal with the case of ODEs. Next, we introduced Marcus signature kernels as extensions of continuous signature kernels from [52] to càdlàg rough paths and used them to define a general-purpose loss function on the space of càdlàg rough paths to train SSNNs where noise is present in both the spike timing and the network's dynamics. Based on these results, we also provided an end-to-end autodifferentiable solver for SDEs with event discontinuities (Algorithm 1) and made its implementation available as part of the `diffrax` repository. Finally, we discussed how our results lead to bioplausible learning algorithms akin to *e-prop* [2] but in the context of spike time gradients.

The primary objective of the paper was to lay out the theoretical foundations of gradient-based learning with stochastic SNNs. Although we provided an initial implementation, which is well-suited for low dimensional examples, a robust version that scales to a high number of neurons is beyond the scope of the paper. Examples that require a much higher number of neurons than the two examples already discussed will be hard to handle with the discretize-then-optimize approach for the reasons given in Remark 3.5.

We think there are still many interesting research directions left to explore. For instance, it would be of interest to implement the adjoint equations or to use reversible solvers and compare the results. Similarly, since our Algorithm 1 differs from the usual approach with surrogate gradients even in the deterministic setting, questions remain on how these methods compare for training SNNs. Furthermore, it would be interesting to understand to what extent the inclusion of different types of driving noises in the dynamics of SSNNs would be beneficial for learning tasks compared to deterministic SNNs. Finally, it remains to be seen whether the discussion in Section 4.4 could lead to a bio-plausible learning algorithm with comparable performance to state-of-the-art backpropagation methods and implementable on neuromorphic hardware.

Acknowledgements. Christian Holberg gratefully acknowledges financial support from Novo Nordisk Foundation through Grant NNF20OC0062958 and from Independent Research Fund Denmark | Natural Sciences through Grant 9040-00215B.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[2] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

[3] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[4] Thomas Cass and Cristopher Salvi. Lecture notes on rough paths and applications to machine learning. *arXiv preprint arXiv:2404.06583*, 2024.

[5] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[6] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations*, 2020.

[7] Ilya Chevyrev and Peter K Friz. Canonical rdes and general semimartingales as rough paths. *The Annals of Probability*, 47(1):420–463, 2019.

[8] Ilya Chevyrev and Harald Oberhauser. Signature moments to characterize laws of stochastic processes. *Journal of Machine Learning Research*, 23(176):1–42, 2022.

[9] Nicola Muca Cirone, Maud Lemercier, and Cristopher Salvi. Neural signature kernels as infinite-width-depth-limits of controlled resnets. In *International Conference on Machine Learning*, pages 25358–25425. PMLR, 2023.