2nd Symposium on Advances in Approximate Bayesian Inference, 2019 1–28

Scalable Gradients and Variational Inference for Stochastic Differential Equations

Xuechen Li LXUECHEN@CS.TORONTO.EDU
Google Research

Ting-Kam Leonard Wong TKL.WONG@UTORONTO.CA
University of Toronto

Ricky T. Q. Chen RTQICHEN@CS.TORONTO.EDU
University of Toronto, Vector Institute

David Duvenaud DUVENAUD@CS.TORONTO.EDU
University of Toronto, Vector Institute

Abstract

We derive reverse-mode (or adjoint) automatic differentiation for solutions of stochastic differential equations (SDEs), allowing time-efficient and constant-memory computation of pathwise gradients, a continuous-time analogue of the reparameterization trick. Specifically, we construct a backward SDE whose solution is the gradient and provide conditions under which numerical solutions converge. We also combine our stochastic adjoint approach with a stochastic variational inference scheme for continuous-time SDE models, allowing us to learn distributions over functions using stochastic gradient descent. Our latent SDE model achieves competitive performance compared to existing approaches on time series modeling.

1. Introduction

Deterministic dynamical systems can often be modeled by ordinary differential equations (ODEs). For training, a memory-efficient implementation of the adjoint sensitivity method (Chen et al., 2018) effectively computes gradients through ODE solutions with constant memory cost. Stochastic differential equations (SDEs) are a generalization of ODEs which incorporate instantaneous noise into their dynamics (Arnold, 1974; Øksendal, 2003). They are a natural fit for modeling phenomena governed by small and unobserved interactions.

In this paper, we generalize the adjoint method to dynamics defined by SDEs resulting in an approach which we call the *stochastic adjoint sensitivity method*. Building on theoretical advances by Kunita (2019), we derive a memory-efficient adjoint method whereby we simultaneously reconstruct the original trajectory and evaluate the gradients by solving a backward SDE (in the sense of Kunita (2019)) whose formulation we detail in Section 3. Computationally, in order to retrace the original trajectory during the backward pass, we need to reuse noise samples generated in the forward pass. In Section 4, we give an algorithm that allows arbitrarily-precise querying of a Brownian motion realization at any time point, while only storing a single random seed. Overall, this results in a constant-memory algorithm that approximates the gradient arbitrarily well as step size reduces by computing vector-Jacobian

products a constant number of times per-iteration. See Table 2 for a comparison of our method against previous approaches in terms of asymptotic time and memory complexity.

We incorporate SDEs into a stochastic variational inference framework, whereby we efficiently compute likelihood ratios and backpropagate through the evidence lower bound using our adjoint approach. This effectively generalizes existing model families such as latent ODEs (Rubanova et al., 2019) and deep Kalman filters (Krishnan et al., 2017).

2. Background

We review works on learning ODEs and SDEs. We refer the reader to Appendix B on background for stochastic flows (Kunita, 2019) and (backward) Stratonovich integrals.

2.1. Adjoint Sensitivity Method

The adjoint sensitivity method is an efficient approach to solve optimization problems by considering the dual form (Pontryagin, 2018). Chen et al. (2018) recently applied this idea to obtain gradients with respect to parameters of a neural network defining an ODE. The method is scalable due to its memory-efficiency, as intermediate computations need not be cached as in regular backpropagation (Rumelhart et al., 1988).

2.2. Neural Stochastic Differential Equation

Recent works have considered SDEs whose drift and diffusion functions are defined by neural networks (Tzen and Raginsky, 2019a,b; Liu et al., 2019; Jia and Benson, 2019). Consider a filtered probability space $(\Omega, \mathrm{F}, \{\mathrm{F}_t\}_{t\in\mathbb{T}}, P)$ on which an m-dimensional adapted Wiener process $\{W_t\}_{t\in\mathbb{T}}$ is defined. An Itô SDE defines a stochastic process $\{Z_t\}_{t\in\mathbb{T}}$ by

$$Z_T = z_0 + \int_0^T b(Z_t, t)\ \mathrm{d}t + \sum_{i=1}^m \int_0^T \sigma_i(Z_t, t)\ \mathrm{d}W_t^{(i)}, \tag{1}$$

where $z_0 \in \mathbb{R}^d$ is a deterministic starting value, and $b : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ and $\sigma_i : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ are the drift and diffusion functions, respectively. Here, the second integral on the right hand side of (1) is the Itô stochastic integral (Øksendal, 2003). When the coefficients are globally Lipschitz in both the state and time components, there exists a unique strong solution to the SDE (Øksendal, 2003). Therefore, one can consider coefficients defined by neural networks that have smooth activation functions (e.g. tanh) of the form $b(z, t, \theta)$ and $\sigma(z, t, \theta)$. This results in a model known as the *neural SDE*.

3. Sensitivities via Stochastic Adjoint

We derive a backward Stratonovich SDE for what we call the *stochastic adjoint process*. A direct implication of this is a gradient computation algorithm that works by solving a set of dynamics in reverse time and relies on vector-Jacobian products without storing intermediate computation. Recall from Appendix B.3, $\Phi_{s,t}(z) := Z_t^{s,z}$ is the solution at time t when the process is started at z at time s, and its inverse is defined as $\check{\Psi}_{s,t}(z) := \Phi_{s,t}^{-1}(z)$.

Consider $A_{s,t}(z) = \nabla(\mathcal{L}(\Phi_{s,t}(z)))$, where $\mathcal{L}$ is a scalar loss function. The chain rule gives $A_{s,t}(z) = \nabla\mathcal{L}(\Phi_{s,t}(z))\nabla\Phi_{s,t}(z)$. Let $\tilde{A}_{s,t}(z) := A_{s,t}(\check{\Psi}_{s,t}(z)) = \nabla\mathcal{L}(z)\nabla\Phi_{s,t}(\check{\Psi}_{s,t}(z)) =$

$\nabla\mathcal{L}(z)K_{s,t}(z)$. Note that $A_{s,t}(z) = \tilde{A}_{s,t}(\Phi_{s,t}(z))$. Since $\nabla\mathcal{L}(z)$ is constant, we see that $(\tilde{A}_{s,t}(z), \check{\Psi}_{s,t}(z))$ satisfies the following backward SDE system by Lemma C.1 (cf. Appendix C)

$$\begin{aligned}\tilde{A}_{s,t}(z) =& \nabla\mathcal{L}(z) + \int_s^t \nabla b(\check{\Psi}_{r,t}(z), r)^\top \tilde{A}_{r,t}(z)\ \mathrm{d}r + \sum_{i=1}^m \int_s^t \nabla\sigma_i(\check{\Psi}_{r,T}(r), u)^\top \tilde{A}_{r,t}(z) \circ \check{\mathrm{d}}W_r^i,\\ \check{\Psi}_{s,t}(z) =& z - \int_s^t b(\check{\Psi}_{r,t}(z), r)\ \mathrm{d}r - \sum_{i=1}^m \int_s^t \sigma_i(\check{\Psi}_{r,t}(z), r) \circ \check{\mathrm{d}}W_r^{(i)}.\end{aligned} \tag{2}$$

Since (11) can be viewed as a single SDE (with smooth coefficients) for an augmented state, $\tilde{A}_{s,T}(z)$ also has a unique strong solution. Therefore, for $t = 0$, we may write

$$\tilde{A}_0(z) = \mathsf{F}(z, W_\cdot), \tag{3}$$

where $W_\cdot = \{W_t\}_{0\le t\le T}$ denotes the path of the Brownian motion and $\mathsf{F} : \mathbb{R}^d \times C([0,1], \mathbb{R}^m) \to \mathbb{R}^d$ is a deterministic measurable function (the Itô map) (Rogers and Williams, 2000, Definition 10.9). The next theorem follows immediately from (3) and the definition of F.

Theorem 3.1:
For P-almost all $\omega \in \Omega$, $A_{0,T}(z) = \tilde{A}_0(z) = \mathsf{F}(\mathsf{G}(z, W_\cdot), W_\cdot)$, where $\mathsf{G}(z, W_\cdot) = Z_T^{0,z}$.

The theorem is a consequence of $A_{0,T}(z) = \tilde{A}_{0,T}(\Phi_{0,T}(z)) = \tilde{A}_{0,T}(Z_T^{0,z})$ and (3). This implies we may solve the dynamics (11) starting from the end state of the forward solve $Z_T^{0,z}$ to obtain the gradient of the loss with respect to the starting value z. To obtain the gradient with respect to the parameters, we augment the original state with parameters. Algorithm 1 summarizes this assuming access to a black-box solver `SDESolve`. See details in Appendix C.

Algorithm 1 Stratonovich SDE Adjoint Computation

Input: parameters θ, start time t_0, stop time t_1, final state z_{t_1}, loss gradient $\partial\mathcal{L}/z_{t_1}$.
Input: drift $b(z, t, \theta)$, diffusion $\sigma(z, t, \theta)$, Wiener process sample $w(t)$.
  $s_{t_1} = [z_{t_1}, \partial\mathcal{L}/\partial z_{t_1}, \mathbf{0}_p]$
  def aug$b([z_t, a_t, \cdot],\ t,\ \theta)$:
    return $[-b(z_t, -t, \theta),\ a_t^\top \partial b/\partial z,\ a_t^\top \partial b/\partial\theta]$
  def aug$\sigma_i([z_t, a_t, \cdot],\ t,\ \theta)$:
    return $[-\sigma_i(z_t, -t, \theta), a_t^\top \partial\sigma_i/\partial z, a_t^\top \partial\sigma_i/\partial\theta]$
  def aug$w(t)$:
    return $[-w(-t), -w(-t), -w(-t)]$
  $[z_{t_0}, \partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta] =$ `SDESolve`$(s_{t_1}$,aug$b$,augσ_1,...,augσ_m,aug$w$,$-t_1$,$-t_0)$
return $[\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta]$

4. Seeded Brownian Tree

We present a data structure that allows arbitrarily-precise query of the sample path of the Wiener process given a global random seed based on the *Brownian tree* construction. The data structure facilitates the adjoint method such that we can ensure the noise

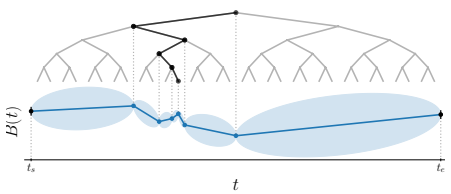


Figure 1: Brownian tree.

sample in the backward solve is the same as in the forward solve using a split pseudorandom random number generator (PRNG). We present the procedure in Algorithm D.2 and details in Appendix D.

5. Latent SDE

Consider the SDEs

$$\mathrm{d}\tilde{Z}_t = h_0(\tilde{Z}_t, t) + \sigma(\tilde{Z}_t, t)\,\mathrm{d}\tilde{W}_t, \quad \tilde{Z}_0 = z \in \mathbb{R}^d, \tag{4}$$

$$\mathrm{d}Z_t = h_1(Z_t, t) + \sigma(Z_t, t)\;\mathrm{d}W_t, \quad Z_0 = z \in \mathbb{R}^d, \tag{5}$$

where $h_0, h_1 \in \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^{d\times m}$ are Lipschitz in both arguments. Suppose (4) and (5) define the prior and posterior processes, respectively. Additionally, assume there is a function $u : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^m$ such that $\sigma(x,t)u(x,t) = h_1(x,t) - h_0(x,t)$ for all $x \in \mathbb{R}^d$ and $t \in \mathbb{R}$. Then, the variational free energy (Opper, 2019) can be written as

$$\mathcal{L}_{\mathrm{VI}} = \mathbb{E}\Big[\tfrac{1}{2}\textstyle\int_0^T |u(Z_t,t)|^2\;\mathrm{d}t + \int_0^T u(Z_t,t)^\top\;\mathrm{d}W_t - \sum_{i=1}^N \log p(y_{t_i}|z_{t_i})\Big], \tag{6}$$

where the expectation is taken over the distribution of the posterior process defined by (5), and $y_1, \ldots, y_N$ are observations at times $t_1, \ldots, t_N$, respectively. To compute the gradient with respect to parameters, we need only augment the forward equation with an extra variable whose drift function returns $\frac{1}{2}|u(Z_t,t)|^2$ and diffusion function is 0. In this case, the backward adjoint dynamics can be derived analogously using (11). Appendix E includes details.

6. Experiments

We verify our theory by comparing the gradients obtained by our stochastic adjoint framework against analytically derived gradients for chosen test problems with closed-form solutions. We then fit latent SDE models with our framework on two synthetic datasets and a real dataset, verifying that the variational inference framework promotes learning a generative model of time series. Due to space constraint, we refer the reader to Appendix F for results on numerical studies and Appendix N for results on synthetic data. We present only the results on the motion capture dataset here.

6.1. Motion Capture Dataset

We experiment on a dataset extracted from the CMU motion capture library. We use the dataset adopted by Gan et al. (2015) which consists of 23 walking sequences of subject number 35 that is partitioned into 16 training, 3 validation, and 4 test sequences. We include the settings in Appendix O and report the test MSE here following Yıldız et al. (2019).

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Table 1: Test MSE; 95% confidence interval averaged over 50 samples. †from Yıldız et al. (2019).

Method	Test MSE
npODE Heinonen et al. (2018)	22.96†
NeuralODE Chen et al. (2018)	22.49 ± 0.88†
ODE^{2}VAE Yıldız et al. (2019)	10.06 ± 1.4†
ODE^{2}VAE-KL Yıldız et al. (2019)	8.09 ± 1.95†
Latent ODE Chen et al. (2018); Rubanova et al. (2019)	5.98 ± 0.28
Latent SDE (this work)	$\mathbf{4.03 \pm 0.20}$

Ludwig Arnold. Stochastic differential equations. *New York*, 1974.

VI Arnold. *Ordinary Differential Equations*. The MIT Press, 1978.

Jonathan Baxter and Peter L Bartlett. Infinite-horizon gradient-based policy search. 2001.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Koen Claessen and Michał H Pałka. Splittable pseudorandom number generators using cryptographic hashing. In *ACM SIGPLAN Notices*, volume 48, pages 47–58. ACM, 2013.

Warren J Ewens. *Mathematical population genetics 1: theoretical introduction*, volume 27. Springer Science & Business Media, 2012.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing, 2018.

Jessica G Gaines and Terry J Lyons. Variable step size control in the numerical solution of stochastic differential equations. *SIAM Journal on Applied Mathematics*, 57(5):1455–1484, 1997.

Zhe Gan, Chunyuan Li, Ricardo Henao, David E Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling. In *Advances in Neural Information Processing Systems*, pages 2467–2475, 2015.

Mike Giles and Paul Glasserman. Smoking adjoints: Fast Monte Carlo greeks. *Risk*, 19(1): 88–92, 2006.

Paul Glasserman and David D Yao. Some guidelines and guarantees for common random numbers. *Management Science*, 38(6):884–908, 1992.

Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

Emmanuel Gobet and Rémi Munos. Sensitivity analysis using Itô–Malliavin calculus and martingales, and application to stochastic optimal control. *SIAM Journal on control and optimization*, 43(5):1676–1713, 2005.

Pashupati Hegde, Markus Heinonen, Harri Lähdesmäki, and Samuel Kaski. Deep learning with differential gaussian process flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1812–1821, 2019.

Markus Heinonen, Cagatay Yildiz, Henrik Mannerström, Jukka Intosalmi, and Harri Lähdesmäki. Learning unknown ode models with gaussian processes. *arXiv preprint arXiv:1803.04303*, 2018.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

Silvana Ilie, Kenneth R Jackson, and Wayne H Enright. Adaptive time-stepping for the strong numerical solution of stochastic differential equations. *Numerical Algorithms*, 68(4): 791–812, 2015.

Junteng Jia and Austin R. Benson. Neural Jump Stochastic Differential Equations. *arXiv e-prints*, art. arXiv:1905.10403, May 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jack PC Kleijnen and Reuven Y Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

Peter E Kloeden and Andreas Neuenkirch. The pathwise convergence of approximation schemes for stochastic differential equations. *LMS journal of Computation and Mathematics*, 10:235–253, 2007.

Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Hiroshi Kunita. *Stochastic Flows and Jump-Diffusions*. Springer, 2019.

Harold Kushner and Paul G Dupuis. *Numerical methods for stochastic control problems in continuous time*, volume 24. Springer Science & Business Media, 2013.

Pierre L'Ecuyer and Gaétan Perron. On the convergence rates of ipa and fdc derivative estimators. *Operations Research*, 42(4):643–656, 1994.

Xuanqing Liu, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

Yi-An Ma, Tianqi Chen, and Emily Fox. A complete recipe for stochastic gradient mcmc. In *Advances in Neural Information Processing Systems*, pages 2917–2925, 2015.

Dougal Maclaurin, David Duvenaud, M Johnson, and RP Adams. Autograd: Reverse-mode differentiation of native python. In *ICML workshop on Automatic Machine Learning*, 2015.

Grigorii Noikhovich Milstein. *Numerical integration of stochastic differential equations*, volume 313. Springer Science & Business Media, 1994.

Bernt Øksendal. *Stochastic Differential Equations*. Springer, 2003.

Manfred Opper. Variational inference for stochastic differential equations. *Annalen der Physik*, 531(3):1800233, 2019.

Etienne Pardoux and Shige Peng. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic Partial Differential Equations and Their Applications*, pages 200–217. Springer, 1992.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Stefano Peluchetti and Stefano Favaro. Neural stochastic differential equations. *arXiv preprint arXiv:1904.01681*, 2019.

Shige Peng. A general stochastic maximum principle for optimal control problems. *SIAM Journal on Control and Optimization*, 28(4):966–979, 1990.

Shige Peng and Zhen Wu. Fully coupled forward-backward stochastic differential equations and applications to optimal control. *SIAM Journal on Control and Optimization*, 37(3): 825–843, 1999.

Eckhard Platen. An introduction to numerical methods for stochastic differential equations. *Acta numerica*, 8:197–246, 1999.

Lev Semenovich Pontryagin. *Mathematical Theory of Optimal Processes*. Routledge, 2018.

Christopher Rackauckas and Qing Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete and Continuous Dynamical Systems. Series B*, 22(7):2731, 2017.

Daniel Revuz and Marc Yor. *Continuous martingales and Brownian motion*, volume 293. Springer Science & Business Media, 2013.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

L Chris G Rogers and David Williams. *Diffusions, Markov Processes and Martingales: Volume 2, Itô Calculus*, volume 2. Cambridge University Press, 2000.

Andreas Rößler. Runge–Kutta methods for stratonovich stochastic differential equation systems with commutative noise. *Journal of Computational and Applied mathematics*, 164: 613–627, 2004.

Andreas Rößler. Runge–Kutta methods for the strong approximation of solutions of stochastic differential equations. *SIAM Journal on Numerical Analysis*, 48(3):922–952, 2010.

Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019a.

Belinda Tzen and Maxim Raginsky. Theoretical guarantees for sampling and inference in generative models with latent diffusions. *arXiv preprint arXiv:1903.01608*, 2019b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (2):283–298, 2007.

Magnus Wiktorsson et al. Joint characteristic function and simultaneous simulation of iterated itô integrals for multiple independent brownian motions. *The Annals of Applied Probability*, 11(2):470–487, 2001.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Jichuan Yang and Harold J Kushner. A monte carlo method for sensitivity analysis and parametric optimization of nonlinear stochastic systems. *SIAM Journal on Control and Optimization*, 29(5):1216–1249, 1991.

Çağatay Yıldız, Markus Heinonen, and Harri Lähdesmäki. Ode2vae: Deep generative second order odes with bayesian neural networks. *arXiv preprint arXiv:1905.10994*, 2019.

Appendix A. Asymptotic Complexity Comparison

Table 2: Asymptotic time complexity of differentiation. L is the number of steps used in the solve, and D is the size of the state. Both memory and time are expressed in units of the cost of evaluating the drift and diffusion functions once each.

Method	Memory	Time
Forward pathwise (Yang and Kushner, 1991)	$\mathcal{O}(1)$	$\mathcal{O}(LD)$
Backprop through solver's ops. (Giles and Glasserman, 2006)	$\mathcal{O}(L)$	$\mathcal{O}(L)$
Stochastic adjoint (ours)	$\mathcal{O}(1)$	$\mathcal{O}(L)$

Appendix B. Additional Background

Notation. For a fixed terminal time $T > 0$, we denote by $\mathbb{T} = [0, T] \subseteq \mathbb{R}$ the time horizon. Let C^∞ be the class of infinitely differentiable functions from $\mathbb{R}^d$ to itself. Let $C^{p,q}$ be the class of functions from $\mathbb{R}^d \times \mathbb{T}$ to $\mathbb{R}^d$ that are p and q times continuously differentiable in the first and second component, respectively. Let $C_b^{p,q} \subseteq C^{p,q}$ be the subclass with bounded derivatives of all possible orders. For a positive integer m, we adopt the short hand $[m] = \{1, 2, \ldots, m\}$. We denote the Euclidean norm of a vector v by $|v|$. For an m-dimensional Wiener process W_t, we denote the ith component of W_t by $W_t^{(i)}$. For the diffusion functions $\sigma_1, \ldots, \sigma_m$, we will also write $\sigma : \mathbb{R}^d \to \mathbb{R}^{d \times m}$ as the matrix-valued function obtained by stacking the component functions σ_i in a columnwise fashion, and index its jth row and ith column by $\sigma_{j,i}$.

B.1. Recent works on Neural SDEs

Among recent work on neural SDEs, none has enabled an efficient training framework. In particular, Tzen and Raginsky (2019a); Liu et al. (2019) considered computing the gradient by simulating the forward dynamics of an explicit Jacobian matrix the size either the squared number of parameters or the number of parameters times the number of states, building on the pathwise approach (Gobet and Munos, 2005; Yang and Kushner, 1991). By contrast, the approach we present only requires evaluating vector-Jacobian products a constant number of times with respect to the number of parameters and states, which has the same asymptotic time cost as evaluating the drift and diffusion functions, and can be done automatically by modern machine learning libraries (Maclaurin et al., 2015; Paszke et al., 2017; Abadi et al., 2016; Frostig et al., 2018).

B.2. Backward Stratonovich Integral

Our stochastic adjoint sensitivity method involves stochastic processes running forward and backward in time. The Stratonovich stochastic integral, due to its symmetry, gives nice expressions for the backward dynamics and so is more convenient for our purpose. Our results can be applied straightforwardly to Itô SDEs as well using a conversion result (see e.g. (Platen, 1999, Sec. 2)).

Following the treatment of Kunita (Kunita, 2019), we introduce the forward and backward Stratonovich integrals. Let $\{\mathrm{F}_{s,t}\}_{s\leq t;s,t\in\mathbb{T}}$ be a two-sided filtration, where $\mathrm{F}_{s,t}$ is the σ-algebra generated by $\{W_v - W_u : s \leq u \leq v \leq t\}$ for all $s,t \in \mathbb{T}$ such that $s \leq t$. For a continuous semimartingale $\{Y_t\}_{t\in\mathbb{T}}$ adapted to the forward filtration $\{\mathrm{F}_{0,t}\}_{t\in\mathbb{T}}$, the Stratonovich stochastic integral is defined as

$$\int_0^T Y_t \circ \mathrm{d}W_t = \lim_{|\Pi|\to 0} \sum_{k=1}^{N} \frac{1}{2}\left(Y_{t_k} + Y_{t_{k-1}}\right)\left(W_{t_k} - W_{t_{k-1}}\right),$$

where $\Pi = \{0 = t_0 < \cdots < t_N = T\}$ is a partition of the interval $\mathbb{T} = [0,T]$, $|\Pi| = \max_k t_k - t_{k-1}$ denotes the size of largest segment of the partition, and the limit is to be interpreted in the L^2 sense. The Itô integral uses instead the left endpoint Y_{t_k} rather than the average. In general, the Itô and Stratonovich integrals differ by a term of finite variation.

To define the backward Stratonovich integral, we consider the backward Wiener process $\{\check{W}_t\}_{t\in\mathbb{T}}$ defined as $\check{W}_t = W_t - W_T$ for all $t \in \mathbb{T}$ that is adapted to the backward filtration $\{\mathrm{F}_{t,T}\}_{t\in\mathbb{T}}$. For a continuous semimartingale $\check{Y}_t$ adapted to the backward filtration, we define

$$\int_s^T \check{Y}_t \circ \check{\mathrm{d}}W_t = \lim_{|\Pi|\to 0} \sum_{k=1}^{N} \frac{1}{2}\left(\check{Y}_{t_k} + \check{Y}_{t_{k-1}}\right)\left(\check{W}_{t_{k-1}} - \check{W}_{t_k}\right),$$

where $\Pi = \{0 = t_N < \cdots < t_0 = T\}$ is a partition, and the limit is again in the L^2 sense.

B.3. Stochastic Flow of Diffeomorphisms

It is well known that an ODE defines a flow of diffeomorphisms (Arnold, 1978). Here we consider the stochastic analogue for the Stratonovich SDE

$$Z_T = z_0 + \int_0^T b(Z_t,t)\ \mathrm{d}t + \sum_{i=1}^{m} \int_0^T \sigma_i(Z_t,t) \circ \mathrm{d}W_t^{(i)}. \tag{7}$$

Throughout the paper, we assume b, σ are of class $C_b^{\infty,1}$, so that the SDE has a unique strong solution. Let $\Phi_{s,t}(z) := Z_t^{s,z}$ be the solution at time t when the process is started at z at time s. Given a realization of the Wiener process, this defines a collection $\mathcal{S} = \{\Phi_{s,t}\}_{s\leq t;s,t\in\mathbb{T}}$ of continuous maps from $\mathbb{R}^d$ to itself.

The following theorem shows that these maps are diffeomorphisms and that they satisfy backward SDEs.

Theorem B.1 (Thm. 3.7.1 (Kunita, 2019)):

(i) With probability 1, the collection $\mathcal{S} = \{\Phi_{s,t}\}_{s\leq t;s,t\in\mathbb{T}}$ satisfies the flow property

$$\Phi_{s,t}(z) = \Phi_{s,u}(\Phi_{u,t}(z)), \quad s \leq u \leq t,\ z \in \mathbb{R}^d.$$

Moreover, each $\Phi_{s,t}$ is a smooth diffeomorphism from $\mathbb{R}^d$ to itself. We thus call $\mathcal{S}$ the stochastic flow of diffeomorphisms generated by the SDE (7).

(ii) The backward flow $\check{\Psi}_{s,t} := \Phi_{s,t}^{-1}$ satisfies the backward SDE:

$$\check{\Psi}_{s,t}(z) = z - \int_s^t b(\check{\Psi}_{u,t}(z), u)\ \mathrm{d}u - \sum_{i=1}^m \int_s^t \sigma_i(\check{\Psi}_{u,t}(z), u) \circ \check{\mathrm{d}}W_u^{(i)}, \tag{8}$$

for all $z \in \mathbb{R}^d$ and $s, t \in \mathbb{T}$ such that $s \leq t$ a.s.

Note that the coefficients in (7) and (8) differ by only a negative sign. This symmetry is due to our use of the Stratonovich integral (see Figure 2).

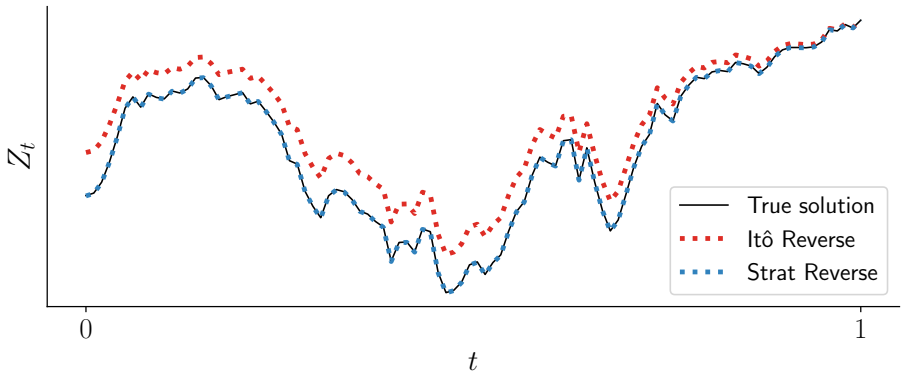


Figure 2: Negating the drift and diffusion functions for an Itô SDE and simulating backwards from the end state gives the wrong solution. Negating the drift and diffusion functions for the converted Stratonovich SDE, however, gives the correct path when simulated in reverse time.

Appendix C. Details on Stochastic Adjoint Derivation and Implementation

We present our main contribution, i.e. the stochastic analog of the adjoint sensitivity method for SDEs. We use (8) to derive another backward Stratonovich SDE for what we call the *stochastic adjoint process*. The direct implication of this is a gradient computation algorithm that works by solving a set of dynamics in reverse time and relies on vector-Jacobian products without storing intermediate computation produced in the forward pass.

C.1. Stochastic Adjoint Process

The goal is to derive the stochastic adjoint process $\{\partial \mathcal{L}/\partial Z_t\}_{t\in\mathbb{T}}$ that can be simulated by evaluating only vector-Jacobian products, where $\mathcal{L} = \mathcal{L}(Z_T)$ is a scalar loss of the terminal state Z_T. The main theoretical result is Theorem 3.1. We first derive a backward SDE for the process $\{\partial Z_T/\partial Z_t\}_{t\in\mathbb{T}}$, assuming that $Z_t = \check{\Psi}_{t,T}(Z_T)$ for a deterministic $Z_T \in \mathbb{R}^d$ that does not depend on the realized Wiener process. We then extend to the case where $Z_T = \Phi_{0,T}(z_0)$ for a deterministic $z_0 \in \mathbb{R}^d$. In the latter case, the resulting value cannot be interpreted as the solution to a backward SDE anymore due to loss of adaptiveness; instead we will formulate the result using the *Itô map* (Rogers and Williams, 2000). Finally, we extend the state of Z to include parameters and obtain the gradient with respect to them.

We first derive the SDE for the Jacobian matrix of the backward flow.

Lemma C.1 (Dynamics of $\partial Z_T/\partial Z_t$):
Consider the stochastic flow generated by the backward SDE (8) as in Theorem B.1(ii). Let $J_s(z) := \nabla \check{\Psi}_{s,T}(z)$, then it satisfies the backward SDE

$$J_{s,t}(z) = I_d - \int_s^t \nabla b(\check{\Psi}_{r,t}(z), r) J_{r,t}(z)\ \mathrm{d}r - \sum_{i=1}^m \int_s^t \nabla \sigma_i(\check{\Psi}_{r,t}(z), r) J_{r,t}(z) \circ \check{\mathrm{d}} W_r^{(i)}, \tag{9}$$

for all $s \le t$ and $x \in \mathbb{R}^d$ a.s. Furthermore, let $K_{s,t}(z) = J_{s,t}(z)^{-1}$, we have

$$K_{s,t}(z) = I_d + \int_s^t K_{r,t}(z) \nabla b(\check{\Psi}_{r,t}(z), r)\ \mathrm{d}r + \sum_{i=1}^m \int_s^t K_{r,t}(z) \nabla \sigma_i(\check{\Psi}_{r,t}(z), r) \circ \check{\mathrm{d}} W_r^{(i)}, \tag{10}$$

for all $s \le t$ and $x \in \mathbb{R}^d$ a.s.

The proof included in Appendix I relies on Itô's lemma in the Stratonovich form (Kunita, 2019, Theorem 2.4.1). This lemma considers only the case where the endpoint z is fixed and deterministic.

Now we compose the state process (represented by the flow) and the loss function $\mathcal{L}$. Consider $A_{s,t}(z) = \nabla(\mathcal{L}(\Phi_{s,t}(z)))$. The chain rule gives $A_{s,t}(z) = \nabla \mathcal{L}(\Phi_{s,t}(z)) \nabla \Phi_{s,t}(z)$. Let

$$\tilde{A}_{s,t}(z) := A_{s,t}(\check{\Psi}_{s,t}(z)) = \nabla \mathcal{L}(z) \nabla \Phi_{s,t}(\check{\Psi}_{s,t}(z)) = \nabla \mathcal{L}(z) K_{s,t}(z).$$

Note that $A_{s,t}(z) = \tilde{A}_{s,t}(\Phi_{s,t}(z))$. Since $\nabla \mathcal{L}(z)$ is constant, we see that $(\tilde{A}_{s,t}(z), \check{\Psi}_{s,t}(z))$ satisfies the backward SDE system

$$\begin{aligned}
\tilde{A}_{s,t}(z) &= \nabla \mathcal{L}(z) + \int_s^t \nabla b(\check{\Psi}_{r,t}(z), r)^\top \tilde{A}_{r,t}(z)\ \mathrm{d}r + \sum_{i=1}^m \int_s^t \nabla \sigma_i(\check{\Psi}_{r,T}(r), u)^\top \tilde{A}_{r,t}(z) \circ \check{\mathrm{d}} W_r^i, \\
\check{\Psi}_{s,t}(z) &= z - \int_s^t b(\check{\Psi}_{r,t}(z), r)\ \mathrm{d}r - \sum_{i=1}^m \int_s^t \sigma_i(\check{\Psi}_{r,t}(z), r) \circ \check{\mathrm{d}} W_r^{(i)}.
\end{aligned} \tag{11}$$

Writing $\check{X}_s = (\tilde{A}_{s,T}(z)^\top, \check{\Psi}_{s,T}(z)^\top)^\top$ as the augmented process, the system (11) is a backward Stratonovich SDE of the form

$$\check{X}_s(z) = A(z) + \int_s^T B(\check{X}_r(z), r)\ \mathrm{d}r + \sum_{i=1}^m \int_s^T S_i(\check{X}_r(z), r) \circ \check{\mathrm{d}} W_r^{(i)},$$

where $B, S_i \in C_b^{\infty,1}$. As a result (11) has a unique strong solution.

Without loss of generality, assume $t = 0$. Since (11) admits a strong solution, we may write

$$\tilde{A}_0(z) = \mathsf{F}(z, W_\cdot),$$

where $W_\cdot = \{W_t\}_{0 \le t \le T}$ denotes the path of the Brownian motion and

$$\mathsf{F} : \mathbb{R}^d \times C([0,1], \mathbb{R}^m) \to \mathbb{R}^d$$

is a deterministic measurable function (the Itô map) (Rogers and Williams, 2000, Definition 10.9). Intuitively, F can be thought as an algorithm that computes the solution to the backward SDE (11) given the position z at time T and the realized Brownian path. Similarly, we let G be the solution map for the forward flow (7). Immediately, we arrive at Theorem 3.1.

C.2. Numerical Approximation

In practice, we compute solutions to SDEs with numerical solvers F_h and G_h, where $h = T/N$ denotes the mesh size of a fixed grid[1]. The approximate algorithm thus outputs $\mathsf{F}_h(\mathsf{G}_h(z, W_\cdot), W_\cdot)$. The following theorem provides sufficient conditions for convergence.

Theorem C.2:
Suppose the schemes F_h and G_h satisfy the following conditions: (i) $\mathsf{F}_h(z, W_\cdot) \to \mathsf{F}(z, W_\cdot)$ and $\mathsf{G}_h(z, W_\cdot) \to \mathsf{G}(z, W_\cdot)$ converge to 0 in probability as $h \to 0$, and (ii) for any $M > 0$, we have $\sup_{|z| \leq M} |\mathsf{F}_h(z, W_\cdot) - \mathsf{F}(z, W_\cdot)| \to 0$ in probability as $h \to 0$. Then, for any starting point z of the forward flow, we have

$$\mathsf{F}_h(\mathsf{G}_h(z, W_\cdot), W_\cdot) \to \mathsf{F}(\mathsf{G}(z, W_\cdot), W_\cdot) = A_{0,T}(z)$$

in probability as $h \to 0$.

For details and the proof see Appendix J. Usual schemes such as the Euler-Maruyama and Milstein method satisfy condition (i). Indeed, they converge pathwise (i.e. almost surely) with explicit rates for any fixed starting point (Kloeden and Neuenkirch, 2007). While condition (ii) is rather strong, we note that the SDEs considered here have smooth coefficients and thus the solutions enjoy nice regularity properties in the starting position. Therefore, it is reasonable to expect that the corresponding numerical schemes to also behave nicely as a function of *both* the mesh size and the starting position. To the best of our knowledge this property is not considered at all in the literature on numerical methods for SDEs (where the initial position is fixed), but is crucial in the proof of Theorem C.2. Detailed analysis for specific schemes is beyond the scope of this paper and is left for future research.

C.3. The Algorithm

So far we have derived the gradient of the loss with respect to the initial state. We can extend these results to give gradients with respect to parameters of the drift and diffusion functions by treating them as an additional part of the state whose dynamics has zero drift and diffusion. We summarize this in Algorithm 1[2], assuming access to a numerical solver `SDESolve`. Note for the Euler-Maruyama scheme, the most costly terms to compute $a_t^\top \partial b / \partial \theta$ and $a_t^\top \partial \sigma_i / \partial \theta$ can be evaluated by calling $\texttt{vjp}(a_t, b, \theta)$ and $\texttt{vjp}(a_t, \sigma_i, \theta)$, respectively.

In principle, we can simulate the forward and backward adjoint dynamics with any high-order solver of choice. However, in practice, to obtain a strong numerical solution[3] with order beyond $1/2$, we need to simulate multiple integrals of the Wiener process such as $\int_0^t \int_0^s \mathrm{d}W_u^{(i)} \, \mathrm{d}W_s^{(j)}$ for $i, j \in [m], i \neq j$. These random variables are difficult to simulate exactly and costly to approximate using truncated infinite series (Wiktorsson et al., 2001).

Note that even though the backward SDE for the stochastic adjoint does not have diagonal noise, it satisfies a commutativity property (Rößler, 2004) when the SDE of the original

1. We may also use adaptive solvers (Ilie et al., 2015).
2. We use row vector notation here.
3. A numerical scheme is of strong order p if $\mathbb{E}\left[|X_T - X_{N\eta}|\right] \leq C\eta^p$ for all $T > 0$, where X_t and $X_{N\eta}$ are respectively the coupled true solution and numerical solution, N and η are respectively the iteration index and step size such that $N\eta = T$, and C is independent of η.

dynamics has diagonal noise. In this case, we can safely adopt certain numerical schemes of strong order 1.0 (e.g. Milstein (Milstein, 1994) and stochastic Runge-Kutta (Rößler, 2010)) without approximating multiple integrals or the Lévy area during simulation. We verify this formally in Appendix K.

C.4. Software and Implementation

We have implemented several SDE solvers in PyTorch (Paszke et al., 2017) which include Euler-Maruyama, Milstein, and stochastic Runge-Kutta schemes with adaptive time-stepping using a PI controller (Ilie et al., 2015). In addition, following `torchdiffeq` (Chen et al., 2018), we have created a user-friendly subclass of `torch.autograd.Function` that facilitates gradient computation using our stochastic adjoint framework when the neural SDE is implemented as a subclass of `torch.nn.Module`. We include a short code snippet covering the main idea of the stochastic adjoint in Appendix L and plan to release all code after the double-blind reviewing process.

Appendix D. Details on Seeded Brownian Tree

The formulation of the adjoint ensures it can be numerically integrated by merely evaluating dynamics cheaply defined by vector-Jacobian products, as opposed to whole Jacobians. However, the backward-in-time nature also introduces the additional difficulty that the same Wiener process sample path in the forward pass has to be queried again during the backward pass. Naïvely storing Brownian motion increments and related quantities (e.g. Lévy area approximations) not only implies a large memory consumption but also disables using adaptive time-stepping numerical integrators, where the evaluation timestamps in the backward pass may be different from those in the forward pass.

To overcome this issue, we combine Brownian trees with splittable Pseudorandom number generators (PRNGs) and obtain a data structure that allows querying values of the Wiener process path at arbitrary times with logarithmic time cost with respect to some error tolerance.

D.1. Brownian Bridge and Brownian Tree

Lévy's *Brownian bridge* (Revuz and Yor, 2013) states that given a start time t_s and end time t_e along with their respective Wiener process values w_s and w_e, the marginal of the process at time $t \in (t_s, t_e)$ is a normal distribution:

$$\mathcal{N}\left(\frac{(t_e - t)w_s + (t - t_s)w_e}{t_e - t_s}, \frac{(t_e - t)(t - t_s)}{t_e - t_s} I_d\right). \tag{12}$$

We can recursively apply this formula to evaluate the process at the midpoint of any two distinct timestamps where the values are already known. Constructing the whole sample path of a Wiener process in this manner results in what is known as the *Brownian tree* (Gaines and Lyons, 1997).

D.2. Combining Brownian Tree with Splittable PRNG

We assume access to a splittable PRNG (Claessen and Pałka, 2013), which has an operation `split` that deterministically generates two (or more) keys[4] using an existing key. In addition, we assume access to an operation `BrownianBridge` which samples from (12) given a key. To obtain the Wiener process value at a specific time, the *seeded Brownian tree* works by recursively sampling according to the Brownian tree with keys split from those of parent nodes, assuming the values at some initial and terminal times are known. The algorithm terminates when the current time under consideration is within a certain error tolerance of the desired time. We outline the full procedure in Algorithm D.2.

Algorithm 2 Seeded Brownian Tree

Input: seed s, query t, tolerance ϵ, start time t_s, start state w_s, end time t_e, end state w_e.
$t_{\text{mid}} = (t_s + t_e)/2$; $w_{\text{mid}} = \texttt{BrownianBridge}(t_s, w_s, t_e, w_e, t_{\text{mid}}, s)$
while $|t - t_{mid}| > \epsilon$ **do**
  $s_l, s_r = \texttt{split}(s)$ ▷ Generates two keys corresponding to left and right subtrees.
  if $t < t_{mid}$ **then**
    $t_e, x_e, s = t_{\text{mid}}, w_{\text{mid}}, s_l$
  else
    $t_s, x_s, s = t_{\text{mid}}, w_{\text{mid}}, s_r$
  end
  $t_{\text{mid}} = (t_s + t_e)/2$; $w_{\text{mid}} = \texttt{BrownianBridge}(t_s, w_s, t_e, w_e, t_{\text{mid}}, s)$
end
return t_{mid}

This algorithm has constant memory cost. For fixed-step-size solvers, the tolerance that the tree will be queried at will scale as $1/L$, where L is the number of steps in the solver. Thus the complexity per-step will scale as $\log L$.

Appendix E. Stochastic Adjoint for Latent SDE

Note that the variational free energy (6) can be derived from Girsanov's change of measure theorem (Opper, 2019). To efficiently Monte Carlo estimate this quantity and its gradient, we simplify the equation by noting that for a one-dimensional process $\{V_t\}_{t\in\mathbb{T}}$ adapted to the filtration generated by a one-dimensional Wiener process $\{W_t\}_{t\in\mathbb{T}}$, if Novikov's condition (Øksendal, 2003) is satisfied, then the process defined by the Itô integral $\int_0^t V_s \, \mathrm{d}W_s$ is a Martingale (Øksendal, 2003). Hence, $\mathbb{E}\big[\int_0^T u(Z_t, t)^\top \, \mathrm{d}W_t\big] = 0$, and

$$\mathcal{L}_{\text{VI}} = \mathbb{E}\Big[\tfrac{1}{2}\textstyle\int_0^T |u(Z_t, t)|^2 \, \mathrm{d}t - \sum_{i=1}^{N} \log p(y_{t_i}|z_{t_i})\Big].$$

To Monte Carlo simulate the quantity in the forward pass along with the original dynamics, we need only extend the original augmented state with an extra variable L_t such that the

4. We consider PRNGs that are based on hashing keys to values which are the desired random numbers.

new drift and diffusion functions for the new augmented state $Y_t = (Z_t^\top, \theta^\top, L_t)^\top$ are

$$f(x,t) = \begin{pmatrix} b(z,t) \\ \mathbf{0}_p \\ \frac{1}{2}|u(z,t)|_2^2 \end{pmatrix} \in \mathbb{R}^{d+p+1}, \quad g_i(x,t) = \begin{pmatrix} \sigma_i(z,t) \\ \mathbf{0}_p \\ 0 \end{pmatrix} \in \mathbb{R}^{d+p+1}, \quad i \in [m].$$

By (11), the backward SDEs of the adjoint processes become

$$\begin{aligned} A_t^z =& A_T^z + \int_t^T \left(\frac{\partial b(z,s)}{\partial z}\bigg|_{z=Z_s}^\top A_s^z + \frac{1}{2}\frac{\partial \|u(z,s)\|_2^2}{\partial z}\bigg|_{z=Z_s} A_s^l \right) \mathrm{d}t + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z,s)}{\partial z}\bigg|_{z=Z_s}^\top A_s^z \circ \check{\mathrm{d}}W_s^{(i)}, \\ A_t^\theta =& A_T^\theta + \int_t^T \left(\frac{\partial b(z,s)}{\partial \theta}\bigg|_{z=Z_s}^\top A_s^z + \frac{1}{2}\frac{\partial \|u(z,s)\|_2^2}{\partial \theta}\bigg|_{z=Z_s} A_s^l \right) \mathrm{d}t + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z,s)}{\partial \theta}\bigg|_{z=Z_s}^\top A_s^z \circ \check{\mathrm{d}}W_s^{(i)}, \\ A_t^l =& A_T^l. \end{aligned} \tag{13}$$

In this case, neither do we need to actually simulate the backward SDE of the extra variable nor do we need to simulate its adjoint. Moreover, when considered as a single system for the augmented adjoint state, the diffusion function of the backward SDE (13) satisfies the commutativity property (17).

Appendix F. Numerical Studies

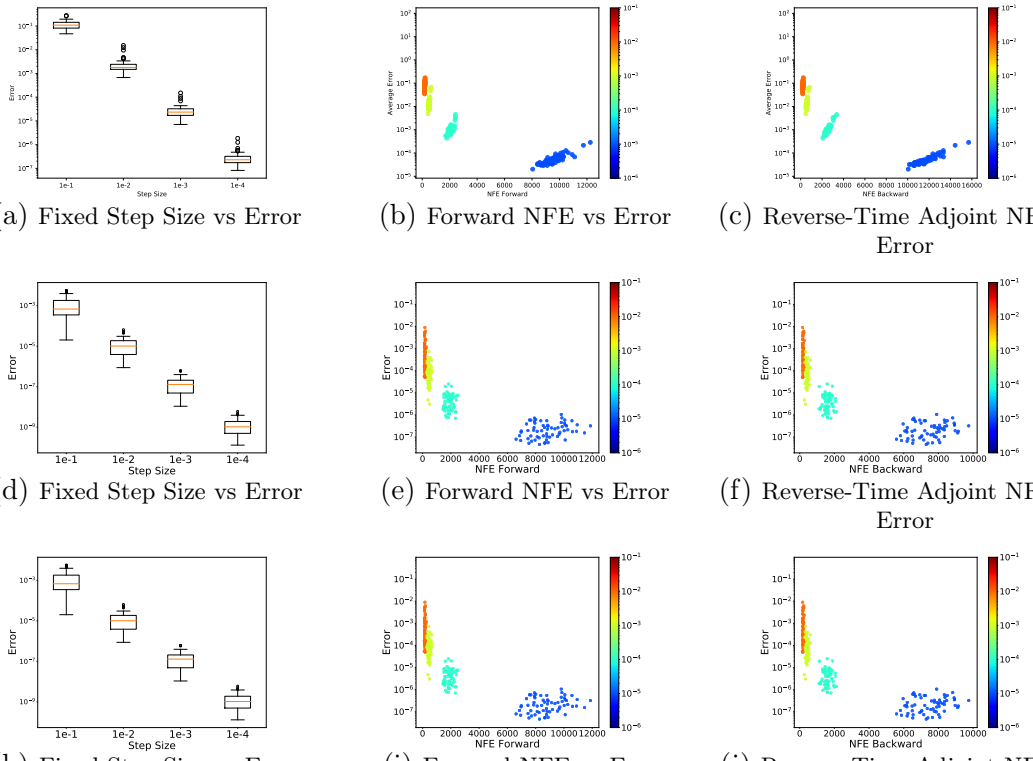
We consider three carefully designed test problems (examples 1-3 (Rackauckas and Nie, 2017); details in Appendix M) all of which have closed-form solutions. We compare the gradient computed from simulating our stochastic adjoint process using the Milstein scheme against the gradient evaluated by analytically solving the equations. Figure F (a) shows that for test example 1, the error between the adjoint gradient and analytical gradient decreases as the fixed step size decreases.

One phenomenon not covered by our theory is that the error can be indeed be controlled by the adaptive solver. This is shown by the fact that for all three test problems, the mean-square error across dimensions tends to be smaller as the absolute tolerance is reduced (see Figure F (c, f, j)). However, we note that the Number of Function Evaluations (NFEs) tends to be much larger than that in the ODE case (Chen et al., 2018), which is expected given the inherent roughness of Brownian motion paths.

Appendix G. Related Work

Sensitivity Analysis for SDEs. Gradient computation is closely related to sensitivity analysis. Computing gradients with respect to parameters of vector fields of an SDE has been extensively studied in the stochastic control literature (Kushner and Dupuis, 2013). In particular, for low dimensional problems, this is done effectively using dynamic programming (Baxter and Bartlett, 2001) and finite differences (Glasserman and Yao, 1992; L'Ecuyer and Perron, 1994). However, both approaches scale poorly with the dimensionality of the parameter vector.

Analogous to REINFORCE (or score-function estimator) (Williams, 1992; Kleijnen and Rubinstein, 1996; Glynn, 1990), Yang and Kushner (1991) considered deriving the gradient



(a) Fixed Step Size vs Error (b) Forward NFE vs Error (c) Reverse-Time Adjoint NFE vs Error

(d) Fixed Step Size vs Error (e) Forward NFE vs Error (f) Reverse-Time Adjoint NFE vs Error

(h) Fixed Step Size vs Error (i) Forward NFE vs Error (j) Reverse-Time Adjoint NFE vs Error

Figure 3: Left of each row: Same fixed step size used in both forward and reverse simulation. Boxplot generated by repeating the experiment with different Brownian motion sample paths for 64 times. Mid and right of each row: Colors of dots represent tolerance levels and correspond to the colorbar on the right. Only `atol` was varied and `rtol` was set to 0. (a-c) Example 1. (d-f) Example 2. (h-j) Example 3.

as $\nabla \mathbb{E}\left[\mathcal{L}(Z_T)\right] = \mathbb{E}\left[\mathcal{L}(Z_T)H\right]$ for some random variable H[5]. However, H usually depends on the density of Z_T with respect to the Lebesgue measure which can be difficulty to compute. Gobet and Munos (2005) extended this approach by weakening a non-degeneracy condition using Mallianvin calculus.

Closely related to the current submission is the pathwise method (Yang and Kushner, 1991), which is the continuous-time analog of the reparameterization trick (Kingma and Welling, 2013; Rezende et al., 2014). Existing methods in this regime (Tzen and Raginsky, 2019a; Gobet and Munos, 2005; Liu et al., 2019) all require simulating a forward SDE where each step requires computing entire Jacobian matrices. This computational cost is prohibitive for high-dimensional systems with a large number of parameters.

Based on the Euler discretization, Giles and Glasserman (2006) considered storing the intermediate values and performing reverse-mode automatic differentiation. They named this method the *adjoint approach*, which, by modern standards, is a form of "backpropagation

5. The random variable H is not unique.

through the operations of a numerical solver". We comment that this approach, despite widely adopted in the field of finance for calibrating market models (Giles and Glasserman, 2006), has high memory cost, and relies on a fixed step size Euler-Maruyama discretization. This approach was used by (Hegde et al., 2019) to parameterize the drift and diffusion of an SDE using Gaussian processes.

Backward SDEs. Our backward SDE for the stochastic adjoint process relies on the notion of backward SDEs by Kunita (2019) which is based on two-sided filtrations. This is different from the more traditional notion of backward SDEs where only a single filtration is defined (Peng, 1990; Pardoux and Peng, 1992). Based on the latter notion, forward-backward SDEs (FBSDEs) have been proposed to solve the stochastic optimal control problem (Peng and Wu, 1999). However, simulating FBSDEs is costly due to the need to estimate conditional expectations in the backward pass. Estimating conditional expectations, however, is a direct consequence of the appearance of an auxiliary process from the Martingale representation theorem (Pardoux and Peng, 1992).

Appendix H. Discussion

We presented a stochastic analog of the adjoint sensitivity method to compute gradients through solutions of SDEs. In contrast to existing approaches, this method has nearly the same time and memory complexity as simply solving the SDE. We showed how our stochastic adjoint framework can incorporate a gradient-based stochastic variational inference scheme for training latent SDEs.

Our method opens up a broad set of opportunities for fitting any differentiable SDE model, such as the mutation and selection parameters in Wright-Fisher models (Ewens, 2012), derivative models in finance, or infinitely-deep Bayesian neural networks (Peluchetti and Favaro, 2019). In addition, the latent SDE model enabled by our framework can be extended to include structure, domain knowledge, and stationarity constraints (Ma et al., 2015) in the prior process.

Appendix I. Proof of Theorem C.1

Proof [Proof of Theorem C.1] We have $J_{s,t}(z) = \nabla \check{\Psi}_{s,t}(z)$, where $\check{\Psi}_{s,t}(z)$ is defined in (8). Now we take the gradient with respect to z on both sides. The solution is differentiable with respect to z and we may differentiate under the stochastic integral (Kunita, 2019, Proposition 2.4.3). Theorem 3.4.3 Kunita (2019) is sufficient for the regularity conditions required. Since $K_{s,t}(z) = J_{s,t}(z)^{-1}$, applying the Stratonovich version of Itô's formula to (9), we have (10). ■

Appendix J. Proof of Theorem C.2

Proof [Proof of Theorem C.2] We want to bound the difference

$$
\begin{aligned}
&|\mathsf{F}(\mathsf{G}(z, W_\cdot), W_\cdot) - \mathsf{F}_h(\mathsf{G}_h(z, W_\cdot), W_\cdot)| \\
&\leq |\mathsf{F}(\mathsf{G}(z, W_\cdot), W_\cdot) - \mathsf{F}(\mathsf{G}_h(z, W_\cdot), W_\cdot)| + |\mathsf{F}(\mathsf{G}_h(z, W_\cdot), W_\cdot) - \mathsf{F}_h(\mathsf{G}_h(z, W_\cdot), W_\cdot)| \\
&=: I_1 + I_2.
\end{aligned}
$$

For notational convenience we suppress z and W..

Bounding I_1. Let $\epsilon > 0$ be given. Since $G_h \to G$ in probability, there exist $M_1 > 0$ and $h_0 > 0$ such that

$$\mathbb{P}(|G| > M_1) < \epsilon, \quad \mathbb{P}(|G_h| > 2M_1) < \epsilon, \quad \text{for all } h \leq h_0.$$

Since the SDE defines a stochastic flow of diffeomorphisms, there exists a finite random variable C_2 such that $\sup_{|z|\leq 2M_1} |\nabla_z \mathsf{F}| \leq C_2$, and there exists $M_2 > 0$ such that $\mathbb{P}(|C_2| > M_2) < \epsilon$. Given M_2, there exists $h_1 > 0$ such that

$$\mathbb{P}(|G - G_h| > \frac{\epsilon}{M_2}) < \epsilon, \quad \text{for all } h \leq h_1.$$

Now suppose $h \leq \min\{h_0, h_1\}$. Then, by the union bound, with probability at least $1 - 4\epsilon$, we have

$$|G| \leq M_1, \quad |G_h| \leq 2M_1, \quad |C_2| \leq M_2, \quad |G - G_h| \leq \frac{\epsilon}{M_2}.$$

On this event, we have

$$I_1 = |\mathsf{F}(\mathsf{G}) - \mathsf{F}(\mathsf{G}_h)| \leq C_2|G - G_h| \leq M_2 \frac{\epsilon}{M_2} = \epsilon.$$

Thus, we have shown that I_1 converges to 0 in probability as $h \to 0$.

Bounding I_2. The idea is similar. By condition (ii), we have

$$\lim_{h\to 0} \sup_{|z_T|\leq M} |\mathsf{F}_h(z_T) - \mathsf{F}(z_T)| = 0$$

in probability. Using this and condition (i), for given $\epsilon > 0$, there exist $M > 0$ and $h_0 > 0$ such that for $h \geq h_0$, we have

$$|G_h| \leq M \text{ and } \sup_{|z_T|\leq M} |\mathsf{F}_h(z_T) - \mathsf{F}(z_T)| < \epsilon$$

with probability at least $1 - \epsilon$. On this event, we have

$$|\mathsf{F}(\mathsf{G}_h) - \mathsf{F}_h(\mathsf{G}_h)| \leq \sup_{|z_T|\leq M} |\mathsf{F}_h(z_T) - \mathsf{F}(z_T)| < \epsilon.$$

Thus I_2 also converges to 0 in probability. ■

Appendix K. Stochastic Adjoint has Commutative Noise when Original SDE has Diagonal Noise

Recall the Stratonovich SDE (7) with drift and diffusion functions $b, \sigma_1, \ldots, \sigma_m \in \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ governed by a set of parameters $\theta \in \mathbb{R}^p$. Consider the augmented state composed of the original state and parameters $Y_t = (Z_t^\top, \theta^\top)^\top$. The augmented state satisfies a Stratonovich SDE with the drift function $f(y, t) = (b(z, t)^\top, \mathbf{0}_p^\top)^\top$ and diffusion functions

$g_i(y,t) = (\sigma_i(z,t)^\top, \mathbf{0}_p^\top)^\top$ for $i \in [m]$. By (10) and (3), the dynamics for the adjoint process of the augmented state is characterized by the backward SDE:

$$A_t^y = A_T^y + \int_t^T \nabla f(Y_s, s)^\top A_s^y \, \mathrm{d}t + \sum_{i=1}^m \int_t^T \nabla g_i(Y_s, s)^\top A_s^y \circ \check{\mathrm{d}}W_s^{(i)}.$$

By definitions of f and g_i, the Jacobian matrices $\nabla f(x,s)$ and $\nabla g_i(x,s)$ can be written as:

$$\nabla f(y,s) = \begin{pmatrix} \frac{\partial b(z,s)}{\partial z} & \mathbf{0}_{d\times p} \\ \mathbf{0}_{p\times d} & \mathbf{0}_{p\times p} \end{pmatrix} \in \mathbb{R}^{(d+p)\times(d+p)}, \quad \nabla g_i(y,s) = \begin{pmatrix} \frac{\partial \sigma_i(z,s)}{\partial z} & \mathbf{0}_{d\times p} \\ \mathbf{0}_{p\times d} & \mathbf{0}_{p\times p} \end{pmatrix} \in \mathbb{R}^{(d+p)\times(d+p)}.$$

Thus, we can write out the backward SDEs for the adjoint processes of the state and parameters separately:

$$\begin{aligned} A_t^z =& A_T^z + \int_t^T \left.\frac{\partial b(z,s)}{\partial z}\right|_{z=Z_s}^\top A_s^z \, \mathrm{d}t + \sum_{i=1}^m \int_t^T \left.\frac{\partial \sigma_i(z,s)}{\partial z}\right|_{z=Z_s}^\top A_s^z \circ \check{\mathrm{d}}W_s^{(i)}, \\ A_t^\theta =& A_T^\theta + \int_t^T \left.\frac{\partial b(z,s)}{\partial \theta}\right|_{z=Z_s}^\top A_s^z \, \mathrm{d}t + \sum_{i=1}^m \int_t^T \left.\frac{\partial \sigma_i(z,s)}{\partial \theta}\right|_{z=Z_s}^\top A_s^z \circ \check{\mathrm{d}}W_s^{(i)}. \end{aligned} \tag{14}$$

Now assume the original SDE has diagonal noise. Then, $m = d$ and Jacobian matrix $\nabla\sigma_i(z)$ can be written as:

$$\frac{\partial \sigma_i(z)}{\partial z} = \begin{pmatrix} 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \frac{\partial \sigma_{i,i}(z)}{\partial z_i} & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{pmatrix}. \tag{15}$$

Consider the adjoint process for the augmented state along with the backward flow of the backward SDE (8). We write the overall state as $X_t = (Z_t^\top, (A_t^z)^\top, (A_t^\theta)^\top)^\top$, where we abuse notation slightly to let $\{Z_t\}_{t\in\mathbb{T}}$ denote the backward flow process. Then, by (14) and (15), $\{X_t\}_{t\in\mathbb{T}}$ satisfies a backward SDE with a diffusion function that can be written as:

$$G(x) = \left(\begin{array}{ccccc} -\sigma_{1,1}(z_1) & 0 & \dots & 0 & 0 \\ & & \dots & & \\ 0 & 0 & \dots & 0 & -\sigma_{d,d}(z_d) \\ \hline \frac{\partial \sigma_{1,1}(z_1)}{\partial z_1} a_1^z & 0 & \dots & 0 & 0 \\ & & \dots & & \\ 0 & 0 & \dots & 0 & \frac{\partial \sigma_{d,d}(z_d)}{\partial z_d} a_d^z \\ \hline \frac{\partial \sigma_{1,1}(z_1)}{\partial \theta_1} a_1^z & \dots & \dots & \dots & \frac{\partial \sigma_{d,d}(z_d)}{\partial \theta_1} a_d^z \\ & & \dots & & \\ \frac{\partial \sigma_{1,1}(z_1)}{\partial \theta_p} a_1^z & \dots & \dots & \dots & \frac{\partial \sigma_{d,d}(z_d)}{\partial \theta_p} a_d^z \end{array}\right) \in \mathbb{R}^{(2d+p)\times d}. \tag{16}$$

Recall, for an SDE with diffusion function $\Sigma(x) \in \mathbb{R}^{d\times m}$, it is said to satisfy the commutativity property (Rößler, 2004) if

$$\sum_{i=1}^d \Sigma_{i,j_2}(x) \frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i} = \sum_{i=1}^d \Sigma_{i,j_1}(x) \frac{\partial \Sigma_{k,j_2}(x)}{\partial x_i}, \tag{17}$$

for all $j_1, j_2 \in [m]$ and $k \in [d]$. When an SDE has commutative noise, the computationally intensive double Itô integrals (and the Lévy areas) need not be simulated by having the numerical scheme take advantage of the following property of iterated integrals (Ilie et al., 2015):

$$\int_s^t \int_s^u \mathrm{d}W_r^{(i)} \, \mathrm{d}W_u^{(j)} + \int_s^t \int_s^u \mathrm{d}W_r^{(j)} \, \mathrm{d}W_u^{(i)} = \Delta W^{(i)} \Delta W^{(j)},$$

where the Brownian motion increment $\Delta W^{(i)} = W_t^{(i)} - W_s^{(i)}$ for $i \in [m]$ can be easily sampled.

To see that the diffusion function (16) indeed satisfies the commutativity condition (17), we consider several cases:

- $k = 1, \ldots, d$: Both LHS and RHS are zero unless $j_1 = j_2 = k$, since for $\Sigma_{i,j_2}(x)\frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i}$ to be non-zero, $i = j_1 = j_2 = k$.
- $k = d + 1 \ldots, 2d$: Similar to the case above.
- $k = 2d + 1 \ldots, 2d + p$: Write $k = 2d + l$, where $l \in [p]$. Both LHS and RHS are zero unless $j_1 = j_2 = l$, since for $\Sigma_{i,j_2}(x)\frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i}$ to be non-zero $i = l$ or $i = d + l$ and $j_1 = j_2 = l$.

Since in all scenarios, LHS = RHS, we conclude that the commutativity condition holds.

Finally, we comment that the Milstein scheme for the stochastic adjoint of diagonal noise SDEs can be implemented such that during each iteration of the backward solve, `vjp` is only called a number of times constant with respect to the dimensionality of the original SDE.

Appendix L. Stochastic Adjoint Implementation

We include the core code of our stochastic adjoint framework, assuming access to a callable Brownian motion `bm`, an Euler-Maruyama integrator `ito_int_diag` for diagonal noise SDEs, and several other helper functions whose purposes can be inferred from their names.

```
class _SdeintAdjointMethod(torch.autograd.Function):

  @staticmethod
  def forward(ctx, *args):
    (y0, f, g, ts, flat_params_f, flat_params_g, dt, bm) = (
      args[:-8], args[-7], args[-6], args[-5], args[-4], args[-3], args[-2], args[-1])
    ctx.f, ctx.g, ctx.dt, ctx.bm = f, g, dt, bm

    def g_prod(t, y, noise):
      g_eval = g(t=t, y=y)
      g_prod_eval = tuple(
        g_eval_i * noise_i for g_eval_i, noise_i in _zip(g_eval, noise))
      return g_prod_eval

    with torch.no_grad():
      ans = ito_int_diag(f, g_prod, y0, ts, dt, bm)
    ctx.save_for_backward(ts, flat_params_f, flat_params_g, *ans)
```

```
    return ans

  @staticmethod
  def backward(ctx, *grad_outputs):
    ts, flat_params_f, flat_params_g, *ans = ctx.saved_tensors
    f, g, dt, bm = ctx.f, ctx.g, ctx.dt, ctx.bm
    f_params, g_params = tuple(f.parameters()), tuple(g.parameters())
    n_tensors = len(ans)

    def aug_f(t, y_aug):
      y, adj_y = y_aug[:n_tensors], y_aug[n_tensors:2 * n_tensors]

      with torch.enable_grad():
        y = tuple(y_.detach().requires_grad_(True) for y_ in y)
        adj_y = tuple(adj_y_.detach() for adj_y_ in adj_y)

        g_eval = g(t=-t, y=y)
        gdg = torch.autograd.grad(
          outputs=g_eval,
          inputs=y,
          grad_outputs=g_eval,
          retain_graph=True, create_graph=True)
        f_eval = f(t=-t, y=y)
        f_eval = _sequence_subtract(gdg, f_eval)  # -f + gdg.

        vjp_y_and_params = torch.autograd.grad(
          outputs=f_eval,
          inputs=y + f_params + g_params,
          grad_outputs=tuple(-adj_y_ for adj_y_ in adj_y),
          retain_graph=True, allow_unused=True)
        vjp_y = vjp_y_and_params[:n_tensors]
        vjp_f = vjp_y_and_params[-len(f_params + g_params):-len(g_params)]
        vjp_g = vjp_y_and_params[-len(g_params):]

        vjp_y = tuple(
          torch.zeros_like(y_) if vjp_y_ is None
          else vjp_y_ for vjp_y_, y_ in zip(vjp_y, y))

        adj_times_dgdx = torch.autograd.grad(
          outputs=g_eval,
          inputs=y,
          grad_outputs=adj_y,
          create_graph=True)
        extra_vjp_y_and_params = torch.autograd.grad(
          outputs=g_eval,
          inputs=y + f_params + g_params,
          grad_outputs=adj_times_dgdx,
          allow_unused=True)
        extra_vjp_y = extra_vjp_y_and_params[:n_tensors]
        extra_vjp_f = extra_vjp_y_and_params[-len(f_params + g_params):-len(g_params)]
        extra_vjp_g = extra_vjp_y_and_params[-len(g_params):]

        extra_vjp_y = tuple(
          torch.zeros_like(y_) if extra_vjp_y_ is None
          else extra_vjp_y_ for extra_vjp_y_, y_ in zip(extra_vjp_y, y))
```

```
      vjp_y = _sequence_add(vjp_y, extra_vjp_y)
      vjp_f = vjp_f + extra_vjp_f
      vjp_g = vjp_g + extra_vjp_g

    return (*f_eval, *vjp_y, vjp_f, vjp_g)

  def aug_g_prod(t, y_aug, noise):
    y, adj_y = y_aug[:n_tensors], y_aug[n_tensors:2 * n_tensors]

    with torch.enable_grad():
      y = tuple(y_.detach().requires_grad_(True) for y_ in y)
      adj_y = tuple(adj_y_.detach() for adj_y_ in adj_y)

      g_eval = tuple(-g_ for g_ in g(t=-t, y=y))
      vjp_y_and_params = torch.autograd.grad(
        outputs=g_eval,
        inputs=y + f_params + g_params,
        grad_outputs=tuple(-noise_ * adj_y_ for noise_, adj_y_ in zip(noise, adj_y)),
        allow_unused=True)
      vjp_y = vjp_y_and_params[:n_tensors]
      vjp_f = vjp_y_and_params[-len(f_params + g_params):-len(g_params)]
      vjp_g = vjp_y_and_params[-len(g_params):]

      vjp_y = tuple(
        torch.zeros_like(y_) if vjp_y_ is None
        else vjp_y_ for vjp_y_, y_ in zip(vjp_y, y)
      )
      g_prod_eval = _sequence_multiply(g_eval, noise)

    return (*g_prod_eval, *vjp_y, vjp_f, vjp_g)

  def aug_bm(t):
    return tuple(-bmi for bmi in bm(-t))

  T = ans[0].size(0)
  with torch.no_grad():
    adj_y = tuple(grad_outputs_[-1] for grad_outputs_ in grad_outputs)
    adj_params_f = torch.zeros_like(flat_params_f)
    adj_params_g = torch.zeros_like(flat_params_g)

    for i in range(T - 1, 0, -1):
      ans_i = tuple(ans_[i] for ans_ in ans)
      aug_y0 = (*ans_i, *adj_y, adj_params_f, adj_params_g)
      aug_ans = ito_int_diag(
        f=aug_f, g_prod=aug_g_prod, y0=aug_y0,
        ts=torch.tensor([-ts[i], -ts[i - 1]]).to(ts),
        dt=dt, bm=aug_bm)
      adj_y = aug_ans[n_tensors:2 * n_tensors]
      adj_params_f, adj_params_g = aug_ans[-2], aug_ans[-1]

      # Take the result at the end time.
      adj_y = tuple(adj_y_[1] for adj_y_ in adj_y)
      adj_params_f, adj_params_g = adj_params_f[1], adj_params_g[1]
```

```
        # Accumulate gradients at intermediate points.
        adj_y = _sequence_add(
          adj_y, tuple(grad_outputs_[i - 1] for grad_outputs_ in grad_outputs)
        )

    return (*adj_y, None, None, None, adj_params_f, adj_params_g, None, None)
```

Appendix M. Test Problems

In the following, α, β, and p are parameters of SDEs, and x_0 is a fixed initial value.

Example 1.

$$\mathrm{d}X_t = \alpha X_t \, \mathrm{d}t + \beta X_t \, \mathrm{d}W_t, \quad X_0 = x_0.$$

Analytical solution:

$$X_t = X_0 e^{\left(\beta - \frac{\alpha^2}{2}\right)t + \alpha W_t}.$$

Example 2.

$$\mathrm{d}X_t = -\left(p^2\right)^2 \sin\left(X_t\right) \cos^3\left(X_t\right) \, \mathrm{d}t + p \cos^2\left(X_t\right) \, \mathrm{d}W_t, \quad X_0 = x_0$$

Analytical solution:

$$X_t = \arctan\left(pW_t + \tan\left(X_0\right)\right).$$

Example 3.

$$\mathrm{d}X_t = \left(\frac{\beta}{\sqrt{1+t}} - \frac{1}{2(1+t)} X_t\right) \mathrm{d}t + \frac{\alpha\beta}{\sqrt{1+t}} \, \mathrm{d}W_t, \quad X_0 = x_0$$

Analytical solution:

$$X_t = \frac{1}{\sqrt{1+t}} X_0 + \frac{\beta}{\sqrt{1+t}} \left(t + \alpha W_t\right).$$

In each numerical experiment, we duplicate the equation 10 times to obtain a system of SDEs where each dimension had their own parameter values sampled from the standard Gaussian distribution and then passed through a sigmoid to ensure positivity. Moreover, we also sample the initial value for each dimension from a Gaussian distribution.

Appendix N. Synthetic Datasets

We consider training latent SDE models with our adjoint framework to recover (1) a 1D Geometric Brownian motion, and (2) a 3D stochastic Lorenz attractor process. The main objective is to verify that the learned posterior is able to reconstruct the training data, and the learned prior exhibit stochastic behavior. We jointly optimize the variational free energy (6) with respect to parameters of the prior and posterior distributions at the initial

latent state z_0, the prior and posterior drift, the diffusion function, the encoder, and the decoder. We include the details of dataset and architecture in Appendix N.1.

For the stochastic Lorenz attractor, not only is the model able to reconstruct the data well, but also the learned prior process can produce bimodal samples in both data and latent space. This is showcased in the last row of Figure 4, where once the initial position sampled from the learned prior distribution is fixed, the latent and data space samples cluster around two modes. Note that this cannot be achieved by a latent ODE, where trajectories are determined once their initial latent state is determined.

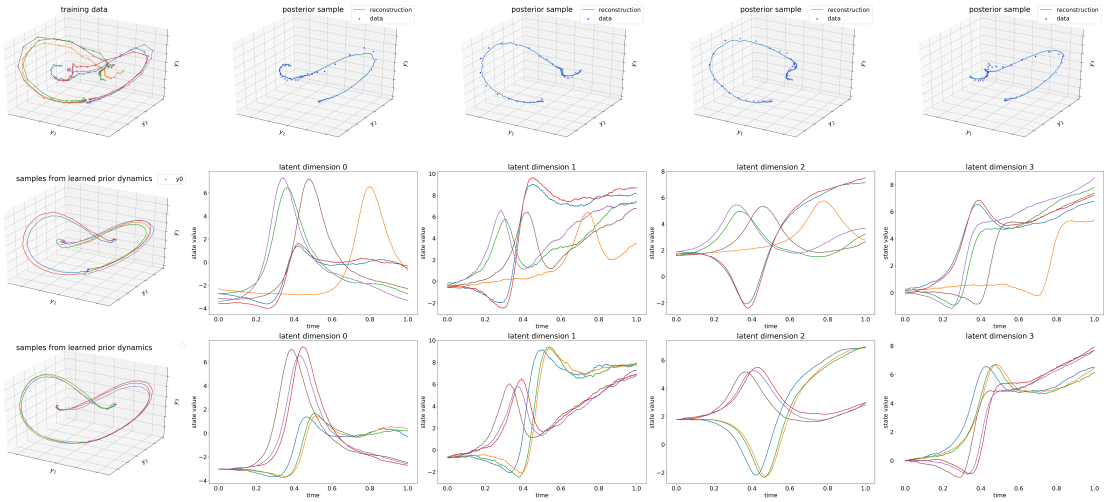


Figure 4: Additional visualizations of learned posterior and prior dynamics on the synthetic stochastic Lorenz attractor dataset. First row displays the true data and posterior reconstructions. Second row displays samples with initial latent state for each trajectory is sampled independently. Third row displays samples with initial latent state sampled and fixed to be the same for different trajectories.

See Figure 4 for additional visualization on the synthetic Lorenz attractor dataset. See Figure 5 for visualization on the synthetic geometric Brownian motion dataset. We comment that for the second example, the posterior reconstructs the data well, and the prior process exhibit behavior of the data. However, from the third row, we can observe that the prior process is learned such that most of the uncertainty is account for in the initial latent state. We leave the investigation of more interpretable prior process for future work.

N.1. Synthetic Datasets Configuration

N.1.1. Geometric Brownian Motion

Consider a geometric Brownian motion SDE:

$$\mathrm{d}X_t = \mu X_t \,\mathrm{d}t + \sigma X_t \,\mathrm{d}W_t, \quad X_0 = x_0.$$

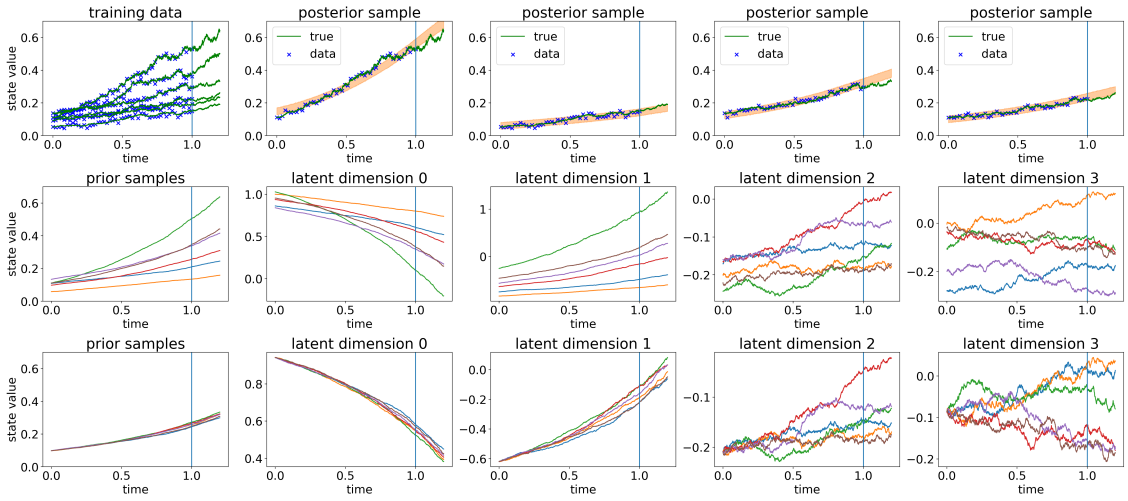


Figure 5: Visualizations of learned posterior and prior dynamics on the synthetic geometric Brownian motion dataset. First row displays the true data and posterior reconstructions. Orange contour covers 95% of 512 samples. Second row displays samples with initial latent state for each trajectory is sampled independently. Third row displays samples with initial latent state sampled and fixed to be the same for different trajectories.

We use $\mu = 1$, $\sigma = 0.5$, and $x_0 = 0.1 + \epsilon$ as the ground-truth model, where $\epsilon \sim \mathrm{N}(0, 0.03^2)$. We sample 1024 time series, each of which is observed at intervals of 0.02 from time 0 to time 1. We corrupt this data using Gaussian noise with mean zero and standard deviation 0.01.

To recover the dynamics, we use a GRU-based Cho et al. (2014) latent SDE model where the GRU has 1 layer and 100 hidden units, the prior and posterior drift functions are MLPs with 1 hidden layer of 100 units, and the diffusion function is an MLP with 1 hidden layer of 100 hidden units and the sigmoid activation applied at the end. The drift function in the posterior is time-inhomogenous in the sense that it takes in a context vector of size 1 at each observation that is output by the GRU from running backwards after processing all future observations. The decoder is a linear mapping from a 4 dimensional latent space to observation space. For all nonlinearities, we use the softplus function. We fix the observation model to be Gaussian with noise standard deviation 0.01.

We optimize the model jointly with respect to the parameters of a Gaussian distribution for initial latent state distribution, the prior and posterior drift functions, the diffusion function, the GRU encoder, and the decoder. We use a fixed discretization with step size of 0.01 in both the forward and backward pass. We use the Adam optimizer Kingma and Ba (2014) with an initial learning rate of 0.01 that is decay by a factor of 0.999 after each iteration. We use a linear KL annealing schedule over the first 50 iterations.

N.1.2. Stochastic Lorenz Attractor

Consider a stochastic Lorenz attractor SDE with diagonal noise:

$$\begin{aligned}
\mathrm{d}X_t &= \sigma\left(Y_t - X_t\right)\,\mathrm{d}t + \alpha_x\,\mathrm{d}W_t, \quad X_0 = x_0,\\
\mathrm{d}Y_t &= \left(X_t\left(\rho - Z_t\right) - Y_t\right)\,\mathrm{d}t + \alpha_y\,\mathrm{d}W_t, \quad Y_0 = y_0,\\
\mathrm{d}Z_t &= \left(X_t Y_t - \beta Z_t\right)\,\mathrm{d}t + \alpha_z\,\mathrm{d}W_t, \quad Z_0 = z_0.
\end{aligned}$$

We use $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, $(\alpha_x, \alpha_y, \alpha_z) = (.1, .28., .3)$, and (x_0, y_0, z_0) sampled from the standard Gaussian distribution as the ground-truth model. We sample 1024 time series, each of which is observed at intervals of 0.025 from time 0 to time 1. We normalize these samples by their mean and standard deviation across each dimension and corrupt this data by Gaussian noise with mean zero and standard deviation 0.01.

We use the same architecture and training procedure for the latent SDE model as in the geometric Brownian motion section, except that the diffusion function consists of four small neural networks, each for a single dimension of the latent SDE.

Appendix O. Settings for Learning from Motion Capture Dataset

We follow the preprocessing used by Wang et al. (2007). Following Yıldız et al. (2019), we use a fully connected network to encode the first three observations of each sequence and thereafter predicted the remaining sequence. Note the current choice of encoder is for comparing fairly to models in the existing literature, and it may be extended to be a recurrent or attention model Vaswani et al. (2017) to enhance performance. The overall architecture is described in Appendix O and is similar to that of ODE^{2}VAE Yıldız et al. (2019) with a similar number of parameters. We also use a fixed step size that is 1/5 of smallest interval between any two observations Yıldız et al. (2019).

We train latent ODE and latent SDE models with the Adam optimizer Kingma and Ba (2014) and its default hyperparameter settings, with an initial learning rate of 0.01 that is exponentially decayed with rate 0.999 during each iteration. We perform validation over the number of training iterations, KL penalty Higgins et al. (2017), and KL annealing schedule. All models were trained for at most 400 iterations, where we start to observe severe overfitting for most model instances.

We use a latent SDE model with an MLP encoder which takes in the first three frames and outputs the mean and log-variance of the variational distribution of the initial latent state and a context vector. The decoder has a similar architecture as that for the ODE^{2}VAE model Yıldız et al. (2019) and projects the 6-dimensional latent state into the 50-dimensional observation space. The posterior drift function takes in a 3-dimensional context vector output by the encoder and the current state and time, whereas the prior drift only takes in the current state and time. The diffusion function is composed of multiple small neural nets, each producing a scalar for the corresponding dimension such that the posterior SDE has diagonal noise. We comment that the overall parameter count of our model (11605) is smaller than that of ODE^{2}VAE for the same task (12157).

The latent ODE baseline was implemented with a similar architecture, except is does not have the diffusion and prior drift components, and its vector field defining the ODE does not

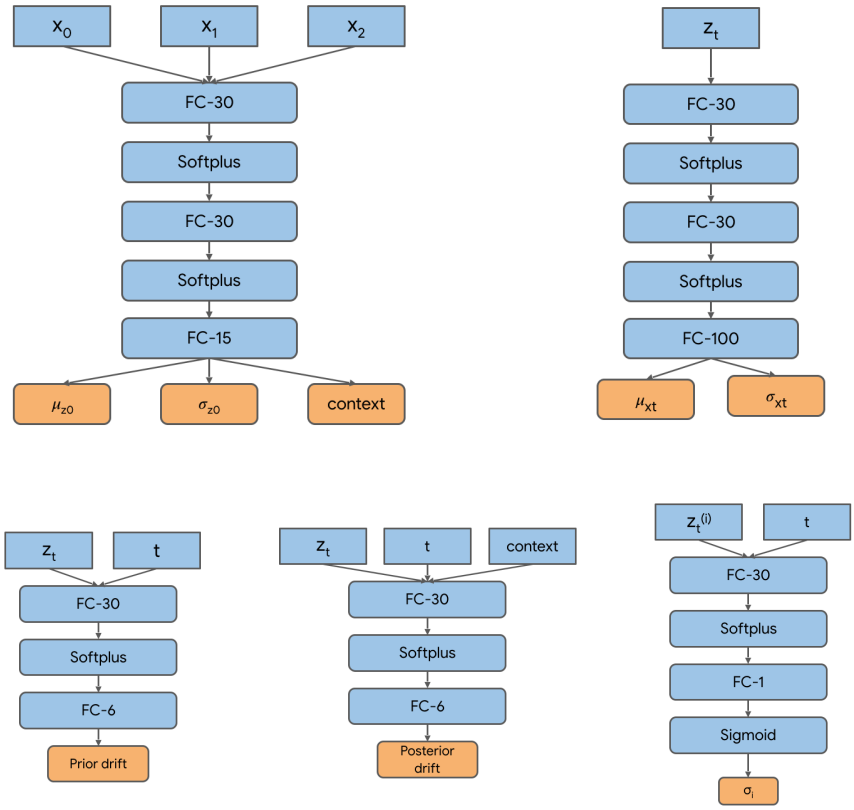


Figure 6: Architecture specifics for the latent SDE model used to train on the mocap dataset. First row from left to right are the encoder and decoder. Second row from left to right are the prior drift, posterior drift, and diffusion functions.

take in a context vector. Therefore, the model has slightly fewer parameters (10573) than the latent SDE model. See Figure 6 for overall details of the architecture.

The main hyperparameter we tuned was the coefficient for reweighting the KL. For both the latent ODE and SDE, we considered training the model with a reweighting coefficient in $\{1, 0.1, 0.01, 0.001\}$, either with or without a linear KL annealing schedule that increased from 0 to the prescribed value over the first 200 iterations of training.