Neural SDEs Made Easy: SDEs are Infinite-Dimensional GANS

Patrick Kidger[1] **James Foster**[1] **Xuechen Li**[2] **Harald Oberhauser**[1] **Terry Lyons**[1]
[1] Mathematical Institute, University of Oxford
[1] The Alan Turing Institute, The British Library
[2] Google Research
{kidger, foster, oberhauser, tlyons}@maths.ox.ac.uk
lxuechen@cs.toronto.edu

Abstract

Several authors have introduced *Neural Stochastic Differential Equations* (Neural SDEs), often involving complex theory with various limitations. Here, we aim to introduce a generic, user friendly approach to neural SDEs. Our central contribution is the observation that an SDE is a map from Wiener measure (Brownian motion) to a solution distribution, which may be sampled from, but which does not admit a straightforward notion of probability density – and that this is just the familiar formulation of a GAN. This produces a continuous-time generative model, arbitrary drift and diffusions are admissible, and in the infinite data limit any SDE may be learnt.

1 Introduction

Neural differential equations are an elegant concept, bringing together the two dominant modelling paradigms of neural networks and differential equations. Indeed, since their introduction, Neural Ordinary Differential Equations [Chen et al., 2018] have prompted the creation of a wide variety of similarly-inspired models, for example based around controlled differential equations [Kidger et al., 2020, Morrill et al., 2020], Lagrangians [Cranmer et al., 2020], higher-order ODEs [Massaroli et al., 2020, Norcliffe et al., 2020], and equilibrium points [Bai et al., 2019].

In particular, several authors have introduced *Neural Stochastic Differential Equations* (neural SDEs), such as Tzen and Raginsky [2019], Li et al. [2020], Hodgkinson et al. [2020] among others. This is what we aim to improve upon here.

1.1 Contributions

We observe that the mathematical formulation of SDEs is directly comparable to the machine learning formulation of GANs. Using this connection, we show how it becomes straightforward to train neural SDEs as generative time series models. Arbitrary drift and diffusions are admissible, and in the infinite data limit any SDE may be learnt.

Specifically, an SDE is a map from a noise distribution (Wiener measure, the distribution of Brownian motion), to the solution of the SDE, which is some other distribution on path space. The model can easily be sampled from: this is what a numerical SDE solver does. However, evaluating

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

its probability density is not possible; in fact it is not even defined in the usual sense.[1] This scenario – no available/tractable densities, but sampling is available – is the familiar setting of a GAN.

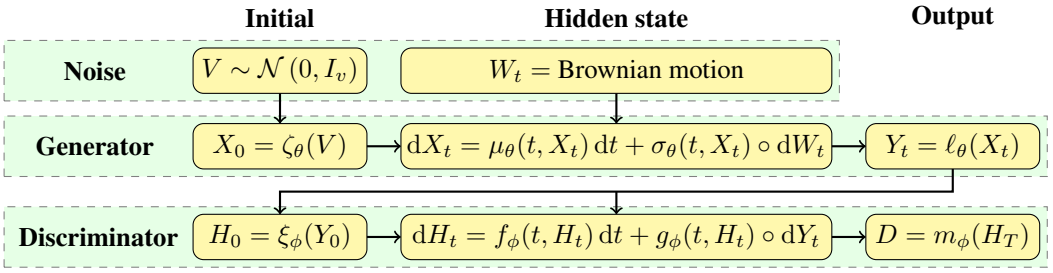

Figure 1: Summary of equations.

2 SDEs as GANs

See Figure 1 for a summary of what we are about to present.

2.1 Generator

Let Z be a random variable on y-dimensional path space. Loosely speaking, this is the space of continuous functions $f\colon [0, T] \to \mathbb{R}^y$ for some fixed time horizon $T > 0$. For example, this may correspond to the (interpolated) evolution of stock prices over time. This is what we seek to model.

Let $W\colon [0, T] \to \mathbb{R}^w$ be a w-dimensional Brownian motion, and $V \sim \mathcal{N}(0, I_v)$ be drawn from a v-dimensional standard multivariate normal. The values w, v are hyperparameters describing the size of the noise.

Let

$$\zeta_\theta\colon \mathbb{R}^v \to \mathbb{R}^x, \qquad \mu_\theta\colon [0, T] \times \mathbb{R}^x \to \mathbb{R}^x, \qquad \sigma_\theta\colon [0, T] \times \mathbb{R}^x \to \mathbb{R}^{x \times w}, \qquad \ell_\theta\colon \mathbb{R}^x \to \mathbb{R}^y,$$

where ζ_θ, μ_θ and σ_θ are (Lipschitz) neural networks, and ℓ_θ is linear. Collectively they are parameterised by θ. The dimension x is a hyperparameter describing the size of the hidden state.

We seek to learn a (Stratonovich) SDE of the form

$$X_0 = \zeta_\theta(V), \qquad \mathrm{d}X_t = \mu_\theta(t, X_t)\,\mathrm{d}t + \sigma_\theta(t, X_t) \circ \mathrm{d}W_t, \qquad Y_t = \ell_\theta(X_t), \tag{1}$$

for $t \in [0, T]$, with $X\colon [0, T] \to \mathbb{R}^x$ the (strong) solution to the SDE, such that $Y \overset{\mathrm{d}}{\approx} Z$. That is to say, the model Y should have approximately the same distribution as the target Z.

Sampling Given a trained model, we sample from it by sampling some initial noise V and some Brownian motion W, and then solving equation (1) with a numerical SDE solver. Any standard numerical SDE solver may be used. It will be a little theoretically neater to use Stratonovich SDEs, rather than Itô SDEs, and so we use the midpoint method (which converges to the Stratonovich solution), rather than the simpler Euler–Maruyama method (which convergees to the Itô solution).

Hidden state The solution X represents hidden state, and is not the output of the model. If it were the output, then future evolution would satisfy a Markov property, of being dependent on the past only through the present, which need not be true in general.

This is the reason for the additional ℓ_θ mapping to Y. Practically speaking, during an SDE solve, Y may be concatenated alongside X, and ℓ_θ concatenated with μ_θ.

[1]Technically speaking, a probability density is the Radon–Nikodym derivative of the measure with respect to the Lebesgue measure. However, the Lebesgue measure only exists for finite dimensional spaces. In infinite dimensions, it is possible to define densities with respect to for example Gaussian measures, but this is less obviously meaningful when used with maximum likelihood.

Initial condition It is important that there be an additional source of noise for the initial condition, passed through a nonlinear ζ_θ, as $Y_0 = \ell_\theta(\zeta_\theta(V))$ does not depend on the Brownian noise W.

2.2 Discriminator

Each sample from the generator is a path $Y \colon [0, T] \to \mathbb{R}^y$; the discriminator must accept such paths as inputs. There is a natural choice: parameterise the discriminator as another neural SDE.

Let

$$\xi_\phi \colon \mathbb{R}^y \to \mathbb{R}^h, \quad f_\phi \colon [0, T] \times \mathbb{R}^h \to \mathbb{R}^h, \quad g_\phi \colon [0, T] \times \mathbb{R}^h \to \mathbb{R}^{h \times y}, \quad m_\phi \colon \mathbb{R}^h \to \mathbb{R},$$

where ξ_ϕ, f_ϕ and g_ϕ are (Lipschitz) neural networks, and m_ϕ is linear. Collectively they are parameterised by ϕ. The dimension h is a hyperparameter describing the size of the hidden state.

Recalling that Y is the generated sample, then the discriminator is an SDE of the form

$$H_0 = \xi_\phi(Y_0), \qquad \mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t) \circ \mathrm{d}Y_t, \qquad D = m_\phi(H_T), \tag{2}$$

for $t \in [0, T]$, with $H \colon [0, T] \to \mathbb{R}^h$ the (strong) solution to this SDE. The value $D \in \mathbb{R}$, which is a function of the terminal hidden state H_T, is the discriminator's score for real versus fake.

Neural CDEs The discriminator follows the formulation of a neural CDE [Kidger et al., 2020] with respect to the control Y. Neural CDEs are the continuous-time analogue to RNNs, just as neural ODEs are the continuous-time analogue to residual networks [Chen et al., 2018]. This is what motivates equation (2) as a probably sensible choice of discriminator. Moreover, it means that the discriminator enjoys theoretical properties, such as universal approximation with respect to compact sets of paths.

Training data Just described is how the discriminator is applied to the generator output. For the training data, the analogous thing is done, as follows.

Suppose for simplicity that we observe samples from Z as an irregularly sampled but fully observed time series $\mathbf{z} = ((t_0, z_0), \ldots, (t_n, z_n))$, where without loss of generality $t_0 = 0$ and $t_n = T$.

Then we may (linearly) interpolate to produce $\widehat{z} \colon [0, T] \to \mathbb{R}^y$ such that $\widehat{z}(t_i) = z_i$, and compute

$$H_0 = \xi_\phi(Y_0), \qquad \mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t) \circ \mathrm{d}\widehat{z}_t, \qquad D = m_\phi(H_T)$$

as before.

If the data is actually partially observed, has asynchronous sampling, or is of variable length, then the interpolation may still be performed in much the same way. See the examples of Kidger [2020].

Initial condition and hidden state As with the generator, it is important that there be a learnt initial condition, and that the output be a function of H_T and not a univariate H_T itself. (See also Kidger et al. [2020], who emphasise the need for a learnt initial condition.)

Single SDE solve In practice, both generator and discriminator may be concatenated together into a single SDE solve. The state is the combined drift $[X, Y, H]$, the drift is the combined diffusion $[\mu_\theta, \ell_\theta, f_\phi]$, and the diffusion is the combined $[\sigma_\theta, 0, g_\phi]$. Then H_T is extracted from the final hidden state, and m_θ applied, to produce the discriminator's score for that sample.

Training loss The training losses used are the usual one for Wasserstein GANs [Goodfellow et al., 2014, Arjovsky et al., 2017]. Let $Y_\theta \colon (V, W) \mapsto Y$ represent the overall action of the generator, and $D_\phi \colon Y \mapsto D$ the overall action of the discriminator. Then the generator is optimised with respect to

$$\min_\theta \left[\mathbb{E}_{V,W} D_\phi(Y_\theta(V, W))\right],$$

and the discriminator is optimised with respect to

$$\max_\phi \left[\mathbb{E}_{V,W} D_\phi(Y_\theta(V, W)) - \mathbb{E}_{\mathbf{z}} D_\phi(\widehat{z})\right].$$

Training is performed via stochastic gradient descent techniques as usual. Backpropagation may be performed either through the internal operations of the numerical SDE solver, or via the adjoint method for SDEs [Li et al., 2020]. In the latter case, then the entire SDE is treated as a single differentiable primitive within the computation graph.

Lipschitz regularisation Wasserstein GANs need a Lipschitz discriminator, for which a variety of methods have been proposed. We use gradient penalty [Gulrajani et al., 2017], finding that neither weight clipping nor spectral normalisation worked [Arjovsky et al., 2017, Miyato et al., 2018].

We attribute this to the observation that neural SDEs (as with RNNs) have a recurrent structure. If a single step has Lipschitz constant λ, then the Lipschitz constant of the overall neural SDE will be $\mathcal{O}(\lambda^T)$ in the time horizon T. Even small positive deviations from $\lambda = 1$ produce large Lipschitz constants. Gradient penalty avoids this by regularising the Lipschitz constant of the overall network.

3 Experiments

We experiment on three datasets. The first is Alphabet/Google stock prices over the course a year, sliced into one-minute intervals. We seek to model the midpoint and log-spread. The second is a univariate dataset consisting of how the weights of a neural network change during training. The third is a dataset of air quality in Beijing, with 6 channels.

We compare against the Latent ODE model of Rubanova et al. [2019] and the continuous time flow process (CTFP) of Deng et al. [2020]. These were selected as being related to neural SDEs: latent ODEs are pure drift models, whilst CTFPs are (nearly) pure diffusion models. Neural SDEs combine both. Latent ODEs are trained as VAEs; CTFPs are trained as normalising flows; neural SDEs are trained as GANs. To our knowledge, neural SDEs are the first model in their class, of continuous-time GANs.

3.1 Results

We study three test metrics: classification, prediction, and MMD. In each case every model is run three times and mean and standard deviation of the test metrics are reported.

Classification is given by training a (neural CDE) model to distinguish real from fake data. Larger losses, meaning inability to classify, indicate better performance of the generative model.

Prediction is a train on synthetic, test on real (TSTR) metric [Hyland et al., 2017]. We train a sequence-to-sequence (neural CDE / neural ODE) model to predict the latter part of a time series given the first part. Smaller losses, meaning ability to predict, are better.

Maximum mean discrepancy is a distance between probability distributions with respect to a kernel or feature map. Smaller values, meaning closer distributions, are better.

Table 1: Results. (Bold indicates best performance.)

		Neural SDE	CTFP	Latent ODE
Stocks	Classification	$\mathbf{0.357 \pm 0.045}$	0.165 ± 0.087	0.000239 ± 0.000086
	Prediction	$\mathbf{0.144 \pm 0.045}$	0.725 ± 0.233	46.2 ± 12.3
	MMD	$\mathbf{1.92 \pm 0.09}$	2.70 ± 0.47	60.4 ± 35.8
Weights	Classification	0.507 ± 0.019	$\mathbf{0.676 \pm 0.014}$	0.0112 ± 0.0025
	Prediction	$\mathbf{0.00843 \pm 0.00759}$	0.0808 ± 0.0514	0.127 ± 0.152
	MMD	$\mathbf{5.28 \pm 1.27}$	12.0 ± 0.5	23.2 ± 11.8
Beijing Air Quality	Classification	0.589 ± 0.051	$\mathbf{0.764 \pm 0.064}$	0.392 ± 0.011
	Prediction	$\mathbf{0.395 \pm 0.056}$	0.810 ± 0.083	0.456 ± 0.095
	MMD	$\mathbf{0.000160 \pm 0.000029}$	0.00198 ± 0.00001	0.000242 ± 0.000002

We use `torchdiffeq`, `torchsde` and `torchcde` to implement these models. [Chen, 2018, Li, 2020, Kidger, 2020]

4 Conclusion

We have shown that SDEs and GANs follow similar formalisms. Using this connection, we show to straightforwardly train neural SDEs as continuous time, infinite dimensional, time series GANs.

Broader Impact

SDE models are archetypal of certain disciplines, such as parts of physics, or quantitative finance. We expect this work to be of particular benefit to these disciplines, as it offers a straightforward way to train general, flexible SDE models. No specific negative consequences are anticipated as a result of this work.

Acknowledgements

PK was supported by the EPSRC grant EP/L015811/1. JF was supported by the EPSRC grant EP/N509711/1. PK, JF, HO, TL were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1. PK thanks Penny Drinkwater for advice on Figure 1.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 2017. PMLR.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems 32*, pages 690–701. Curran Associates, Inc., 2019.

Ricky T. Q. Chen. torchdiffeq, 2018. https://github.com/rtqichen/torchdiffeq.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Ruizhi Deng, Bo Chang, Marcus A. Brubaker, Greg Mori, and Andreas Lehrmann. Modeling Continuous Stochastic Processes with Dynamic Normalizing Flows. *arXiv:2002.10516*, 2020.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. title = Generative Adversarial Nets, Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

Liam Hodgkinson, Chris van der Heide, Fred Roosta, and Michael Mahoney. Stochastic Normalizing Flows. *arXiv:2002.09547*, 2020.

Stephanie L. Hyland, Cristóbal Esteban, and Gunnar Rätsch. Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs. *arXiv:1706.02633*, 2017.

Patrick Kidger. `torchcde`, 2020. https://github.com/patrick-kidger/torchcde.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural Controlled Differential Equations for Irregular Time Series. *arXiv:2005.08926*, 2020.

Xuechen Li. `torchsde`, 2020. https://github.com/google-research/torchsde.

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David K. Duvenaud. Scalable Gradients and Variational Inference for Stochastic Differential Equations. *AISTATS*, 2020.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. *arXiv:2002.08071*, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations*, 2018.

James Morrill, Patrick Kidger, Cristopher Salvi, James Foster, and Terry Lyons. Neural CDEs for Long Time-Series via the Log-ODE Method. *arXiv:2009.08295*, 2020.

Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Lió. On Second Order Behaviourin Augmented Neural ODEs. *arXiv:2006.07220*, 2020.

Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems 32*, pages 5320–5330. Curran Associates, Inc., 2019.

Belinda Tzen and Maxim Raginsky. Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit. *arXiv:1905.09883*, 2019.