NEURAL SDES MADE EASY: SDES ARE INFINITE-DIMENSIONAL GANS

Anonymous authors
Paper under double-blind review

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. After that, we construct a new scheme for sampling *and reconstructing* Brownian motion, with constant average-case time and memory costs, adapted to the access patterns of an SDE solver. Finally, we demonstrate that the adjoint SDE (used for backpropagation) may be constructed via rough path theory, without the previous theoretical complexity of two-sided filtrations.

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., 2020b; 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 & Raginsky (2019a); Li et al. (2020); Hodgkinson et al. (2020) among others.

1.1 RELATED WORK

We begin by discussing previous formulations, and applications, of Neural SDEs.

Tzen & Raginsky (2019a;b) obtain Neural SDEs as a continuous limit of deep latent Gaussian models. They train by optimising a variational bound, using forward-mode autodifferentiation. They consider only theoretical applications, for modelling distributions as the terminal value of an SDE.

Li et al. (2020) give arguably the closest analogue to the neural ODEs of Chen et al. (2018). They introduce neural SDEs via a subtle argument involving two-sided filtrations and backward Stratonovich integrals, but in doing so are able to introduce a backward-in-time adjoint equation, using only efficient-to-compute vector-Jacobian products. In applications, they use neural SDEs in a latent variable modelling framework, using the stochasticity to model Bayesian uncertainty.

Hodgkinson et al. (2020) introduce Neural SDEs via an elegant theoretical argument, as a limit of random ODEs. The limit is made meaningful via rough path theory. In applications, they use the limiting random ODEs, and treat stochasticity as a regulariser within a normalising flow. However, they remark that in this setting the optimal diffusion is zero. This is a recurring problem: Innes et al. (2019) also train neural SDEs for which the optimal diffusion is zero.

Rackauckas et al. (2020) treat neural SDEs in classical Feynman–Kac fashion, and like Hodgkinson et al. (2020); Tzen & Raginsky (2019a;b), optimise a loss on just the terminal value of the SDE.

Briol et al. (2020); Gierjatowicz et al. (2020) instead consider the more general case of using a neural SDE to model a time-varying quantity, for which the stochasticity in the system models the variability (specifically certain statistics) of time-varying data. Letting μ, ν denote the learnt and true distributions, both train by minimising $|\mu(f) - \nu(f)|$ for functions of interest f (such as derivative payoffs). This corresponds to training with a non-characteristic MMD (Gretton et al., 2013).

Several authors, such as Oganesyan et al. (2020); Hodgkinson et al. (2020); Liu et al. (2019), seek to use stochasticity as a way to enhance or regularise a neural ODE model.

Our approach is most similar to Li et al. (2020), in that we treat neural SDEs as learnt continuous-time model components of a differentiable computation graph, and Briol et al. (2020); Gierjatowicz et al. (2020), in that we use stochasticity to model distributions on path space. The resulting neural SDE is not a improvement to a similar neural ODE, but a standalone concept in its own right.

1.2 Contributions

Our central contribution is the observation 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.

Next, we introduce a new way of sampling Brownian motion, adapted to the query patterns typical to SDE solvers. The scheme produces exact samples using $\mathcal{O}(1)$ memory and average-case $\mathcal{O}(1)$ time. In particular, it can reconstruct its past trajectory, which is necessary for the use of adjoint equations. The scheme operates by combining splittable Pseudo-Random Number Generators (PRNGs), a binary tree of dependent intervals, and a Least Recently Used (LRU) cache of recent queries.

Finally, we demonstrate that the theoretical construction of adjoint SDEs (which may be used to backpropagate through an SDE) may be simplified by using the pathwise formulation of rough path theory. In particular this avoids the previous theoretical complexity of two-sided filtrations.

To facilitate the use of these techniques, we have implemented them as part of an open-source PyTorch-compatible general-purpose SDE library, [redacted] . This may be found at `https://github.com/[redacted]` .

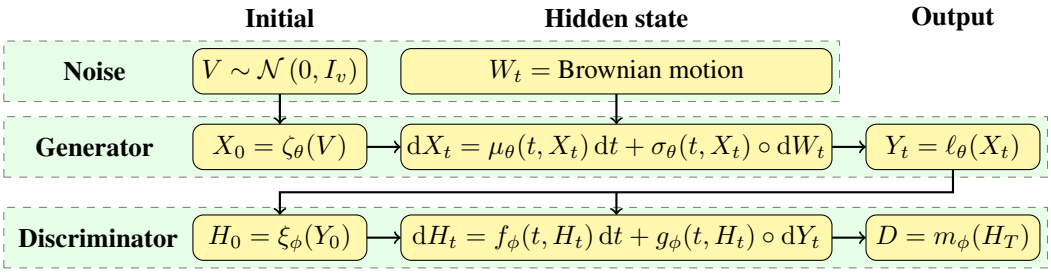

Figure 1: Summary of equations.

2 SDEs as GANs

2.1 Overview

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 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.

[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.

2.2 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.

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.

2.3 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., 2020b) 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. (2020b), 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.

The use of gradient penalty does require a double backward. This is a concern we shall return to in Section 5.

2.4 Extensions

Conditional GANs This approach may be extended to conditional GANs; simply append the extra context to both generator and discriminator as in Mirza & Osindero (2014); Ren et al. (2016)

MMD-GANs Given a kernel on path space (not just on samples as in Briol et al. (2020)), for example the signature kernel (Király & Oberhauser, 2019; Toth & Oberhauser, 2020), then the neural SDE may alternatively be trained as an MMD-GAN as well (Li et al., 2015).

Jumps It is straightforward to include jump terms (Jia & Benson, 2019) in the generator. The formulation of the discriminator is unchanged.

We consider conditional GANs in our experiments, and leave MMDs and jumps for future work.

3 EFFICIENT COMPUTATION

The SDE-as-GAN formulation is expected to be the primary interest for its machine learning applications.

Still, we now provide two further technical contributions. First, we demonstrate how the construction of the adjoint equations may be performed straightforwardly via rough path theory. Second, we construct a new scheme for simulating Brownian motion, which we dub the Brownian Interval.

As a practical matter, these may be handled by SDE libraries, without the end user having to worry about how they are performed. And indeed, we make these available in the [redacted] library.

3.1 ROUGH ADJOINT EQUATION

Neural differential equations may be backpropagated through either by backpropagating through the internal operations of the solver, or by treating the entire neural differential equation as a differentiable primitive via the *adjoint method* (Pontryagin et al., 1962; Chen et al., 2018). The adjoint method backpropagates by solving another differential equation, the *adjoint equation*, backwards in time. See Kidger et al. (2020a) for a clear exposition in the ODE setting.

However, it does not straightforwardly extend to the SDE setting, as the theory of SDEs relies on arguments that depend on the arrow of time, such as filtrations. Li et al. (2020) manage to handle this by using a nonstandard setting involving subtle arguments with two-sided filtrations.

Here we show that the issue may instead be straightforwardly resolved by the adoption of rough path theory. Hodgkinson et al. (2020) give a very readable introduction, and Friz & Hairer (2014); Friz & Victoir (2010) are standard textbooks. In this formulation: (a) the integrals $\mathbb{W}_{s,t} = \int_s^t (W_r - W_s) \circ \mathrm{d}W_r$ are defined probabilistically as typically Stratonovich integrals; (b) the joint probability distribution $(W, \mathbb{W})$ is sampled; (c) the solution to the SDE is defined *pathwise* with respect to the sample.[2] The solution is the same as the strong solution given by usual SDE theory, but now the probability is contained just within $(W, \mathbb{W})$, and complications such as filtrations do not appear.

It is instructive to note that this is the same procedure as performed when using numerical SDE solvers. Samples are drawn from $(W, \mathbb{W})$, and then the SDE is solved pathwise with respect to those samples. The higher order terms $\mathbb{W}$ appear for example in Milstein's method (Kloeden & Platen, 1992), originally constructed for solving Itô SDEs.

What this then means is that it is straightforward to make sense of notions like running SDEs backwards in time: we sample from $(W, \mathbb{W})$ as before, and then just traverse the sample backwards.

Theorem (Informal). *Consider the Stratonovich SDE of equation* (1)*, and let L be a (scalar) loss on X_T. Then the adjoint process $a_t = {}^{\mathrm{d}L(X_T)}/_{\mathrm{d}X_t}$ is a strong solution of the linear Stratonovich SDE*

$$\mathrm{d}a_t = -(a_t \cdot \nabla)\, \mu_\theta(t, X_t)\, \mathrm{d}t - (a_t \cdot \nabla)\, \sigma_\theta(t, X_t) \circ \mathrm{d}W_t$$

for $t \in [0, T]$. In particular W_t is the same Brownian noise as used in the forward pass.

This is cheap to compute as it involves only vector-Jacobian products. This is equivalent to the adjoint as given by Li et al. (2020): the difference is through our use of conventional Stratonovich integrals over their "backward Stratonovich integrals" – and we argue that interpreting these as rough integrals is better still.

The proof may be found in Appendix A.

3.2 SIMULATING BROWNIAN MOTION

Numerically solving an SDE requires sampling Brownian motion, conditional on its previous samples. Mathematically this is straightforward. Let $W_{s,t} = W_t - W_s \in \mathbb{R}^w$. Then for $s < t < u$, Lévy's Brownian bridge (Revuz & Yor, 2013) states that

$$W_{s,t} | W_{s,u} = \mathcal{N}\left(\frac{t-s}{u-s} W_{s,u}, \frac{(u-t)(t-s)}{u-s} I_w\right). \tag{3}$$

[2] For almost all samples. In fact the excluded null set is the same for all SDEs. Note the use of Stratonovich integrals – this is the reason for our previous preference for them over Itô integrals.

The difficulty here is computational. On the adjoint pass, the same Brownian sample must be reconstructed, potentially at locations other than were used on the forward pass. A memory intensive approach is to store every sample, and apply equation (3) when appropriate. Gaines & Lyons (1997); Li et al. (2020) instead approach this via the "Brownian Tree". However this produces only approximations, as the real line must be discretised to some tolerance at which the tree is terminated, and practically speaking this is slow, as small tolerances demand deep traversals of the tree.

We introduce the "Brownian Interval", which improves upon this with exact samples and fast query times. Similar to the dyadic tree of points used in the Brownian Tree, we now instead we have a binary tree of intervals. Each parent interval is the disjoint union of its child intervals. What is simulated is actually the increments $W_{s,t}$, not displacements W_t. This is because (a) this is what is actually used in an SDE solver, and (b) this is the appropriate interface when additionally desiring the higher order term $\mathbb{W}_{s,t} = \int_s^t W_{s,r} \circ \mathrm{d}W_r$, such as for Milstein's method.

The tree starts as a stump consisting of just the global interval $[0, T]$. New leaf nodes are created as queries over intervals are made. For example, making a first query at $[s, t] \subseteq [0, T]$ (an operation that returns $W_{s,t}$) produces the binary tree shown in Figure 2a; making a subsequent query at $[u, v]$ with $u < s < v < t$ produces Figure 2b. Using a splittable PRNG (Salmon et al., 2011), each child node also has a random seed deterministically produced from the seed of its parent.

The tree thus completely encodes the conditional statistics of Brownian motion: $W_{s,t}, W_{t,u}$ are completely specified by s, t, u, $W_{s,u}$, equation (3), and the random seed associated with $[s, u]$.

Algorithm 1: Sampling the Brownian Interval

Type: Let *Node* denote a 5-tuple consisting of an interval, a seed, and three optional *Node*s, corresponding to the parent node, and two child nodes, respectively. (Optional as the root has no parent and leaves have no children.)
Input: Interval $[s, t] \subseteq [0, T]$
State: Binary tree with elements of type *Node*, with root $\widehat{I} = ([0, T], \widehat{s}, *, \widehat{I}_{\texttt{left}}, \widehat{I}_{\texttt{right}})$. A *Node* $\widehat{J}$.
Result: Sample increment $W_{s,t}$

# The returned 'nodes' is a list of *Node*s whose
# intervals partition $[s, t]$. Practically speaking
# this will usually have only one or two elements.
nodes = `traverse`($\widehat{J}$, $[s, t]$, [ ])

def `sample`(I : *Node*)**:**
  if I is $\widehat{I}$ **then**
    return $\mathcal{N}(0, T)$ sampled with seed $\widehat{s}$.
  Let $I = ([a, b], s, I_{\texttt{parent}}, I_{\texttt{left}}, I_{\texttt{right}})$
  Let $I_{\texttt{parent}} = ([a_p, b_p], s_p, I_{pp}, I_{lp}, I_{rp})$
  $W_{\texttt{parent}}$ = `sample`($I_{\texttt{parent}}$)
  if I_i is I_{rp} **then**
    $W_{\texttt{left}} = \texttt{bridge}(a_p, b_p, a, W_{\texttt{parent}}, s)$
    return $W_{\texttt{parent}} - W_{\texttt{left}}$
  else
    return `bridge`(a_p, b_p, b, $W_{\texttt{parent}}$, s)

`sample` = `LRUCache`(`sample`)

$\widehat{J} \leftarrow$ nodes$[-1]$
return $\sum_{I \in \text{nodes}}$ `sample`(I)

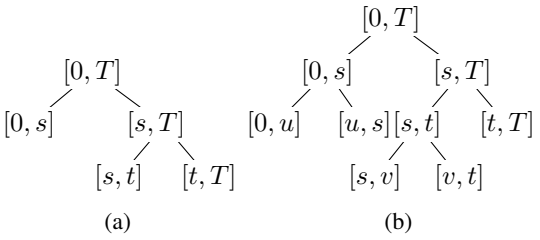

Figure 2: Binary tree of intervals.

Computing $W_{s,t}$ in this way requires $W_{s,u}$, which is not itself stored; in principle it is instead calculated by recursing up the tree. This would be very slow (recursing to the root on every query), except that an LRU cache is additionally applied to the computed increments $W_{s,t}$.

Queries are exact because the tree aligns with the query points. Queries are fast because of the LRU cache; in SDE solvers, subsequent queries are likely to be close to (and thus conditional on) previous queries. The average-case (modal) time complexity is thus $\mathcal{O}(1)$. Even in the event of cache misses all the way up the tree, the worst-case time complexity will only be $\mathcal{O}(\log(1/s))$ in the average step size s of the SDE solver. The (GPU) memory cost is essentially the size of the LRU cache, which is constant. There is the small additional cost of storing the tree structure itself, but this is held in CPU memory, which is for practical purposes essentially infinite.

See Algorithm 1, where `bridge` denotes equation (3), and `traverse` traverses the binary tree to find a list of nodes that are of interest, and is defined explicitly in Appendix B. Also see Appendix B for various extensions and technical considerations need to ensure this algorithm works.

4 EXPERIMENTS

4.1 DATASETS

Stocks We consider a dataset consisting of Alphabet/Google stock prices for 2018–2019, obtained from LOBSTER (Haase, 2013). The data consists of limit orders, in particular ask and bid prices. On average there are 605 054 values per day. Many of these do not actually change the price (specifically the midpoint or spread), so we downsample to 40 000 observations per day, specifically over the trading period 9.30am–4pm. This is then sliced into windows of length approximately one minute. We model the bivariate path consisting of the midpoint and the log-spread over this time interval.

Weights Next, we consider another problem that is classically produced via (stochastic) differential equations: the weight updates when training a neural network via stochastic gradient descent. We train a small convolutional network on MNIST (LeCun et al., 2010) for 100 epochs, and record its weights on every epoch. Repeated over 10 models, this produces a dataset of univariate time series; each time series corresponding to a particular scalar weight.

Beijing Air Quality We consider a dataset of the air quality in Beijing, from the UCI repository (Zhang et al., 2017; Dua & Graff, 2017). Each sample is a 6-dimensional time series of the SO_2, NO_2, CO, O_3, $PM_{2.5}$ and PM_{10} concentrations, as they change over the course of a day. We train this problem as a conditional GAN, by conditioning on the one-hot encoded label for which of 14 different locations the data was measured at.

4.2 MODELS

We compare against the Latent ODE model of Rubanova et al. (2019) and the continuous time flow process (CTFP) of Deng et al. (2020). We used the full version of CTFPs, including latent variables.

These were selected for being competing neural differential equation models, which additionally represent differing extremes of neural SDEs. The Latent ODE model samples its noise as an initial condition, and is thereafter a pure-drift model. Meanwhile, continuous time flow processes sweep a normalising flow over a Brownian noise, and thus represent a pure-diffusion model. Neural SDEs, however, combine both drift and diffusion terms.

Between them these models cover a variety of training regimes. Latent ODEs are trained as variational autoencoders; CTFPs are trained as normalising flows; neural SDEs are trained as GANs. To our knowledge neural SDEs are the first model in their class, namely continuous-time GANs.

4.3 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. See Appendix C for details of hyperparameters, learning rates, optimisers and so on.

Classification is given by training a model to distinguish real from fake data. We use a neural CDE (Kidger et al., 2020b) for the classifier. Larger losses, meaning inability to classify, indicate better performance of the generative model.

Table 1: Classification loss. (Bold indicates best performance.)

	Neural SDE	CTFP	Latent ODE
Stocks	$\mathbf{0.357 \pm 0.045}$	0.165 ± 0.087	0.000239 ± 0.000086
Weights	0.507 ± 0.019	$\mathbf{0.676 \pm 0.014}$	0.0112 ± 0.0025
Beijing Air Quality	0.589 ± 0.051	$\mathbf{0.764 \pm 0.064}$	0.392 ± 0.011

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

Table 2: Prediction loss. (Bold indicates best performance.)

	Neural SDE	CTFP	Latent ODE
Stocks	$\mathbf{0.144 \pm 0.0446}$	0.725 ± 0.233	46.2 ± 12.3
Weights	$\mathbf{0.00843 \pm 0.00759}$	0.0808 ± 0.0514	0.127 ± 0.152
Beijing Air Quality	$\mathbf{0.395 \pm 0.056}$	0.810 ± 0.083	0.456 ± 0.095

Maximum mean discrepancy is a distance between probability distributions with respect to a kernel or feature map. We use the depth-5 signature transform as the feature map (Király & Oberhauser, 2019; Toth & Oberhauser, 2020). Smaller values, meaning closer distributions, are better.

Table 3: MMD loss. (Bold indicates best performance.)

	Neural SDE	CTFP	Latent ODE
Stocks	$\mathbf{1.92 \pm 0.09}$	2.70 ± 0.47	60.4 ± 35.8
Weights	$\mathbf{5.28 \pm 1.27}$	12.0 ± 0.5	23.2 ± 11.8
Beijing Air Quality	$\mathbf{0.000160 \pm 0.000029}$	0.00198 ± 0.00001	0.000242 ± 0.000002

Overall, we see that Neural SDEs are strong performers.

Neural SDEs produce substantially better results with respect to both the predictive (forecasting) and MMD metrics. On the stocks data, they additionally perform substantially better on the classification metric – stocks being a regime in which SDE models have classically been applied.

We see that both neural SDEs and CTFPs consistently outperform Latent ODEs, which we attribute to the nature of these datasets: on these problems we expect to see some random fluctuations, and the underlying dynamics are not those of a pure drift.

5 Considerations

Stochastic weight averaging We found that using stochastic weight averaging (Izmailov et al., 2018) was particularly helpful for improving performance, as it averages out the oscillatory training behaviour for the min-max objective used in GAN training.

Final tanh nonlinearity Using a final tanh nonlinearity (on both drift and diffusion, for both generator and discriminator) constraints the rate of change of hidden state (Kidger et al., 2020b), which we found helped training.

Lipschitz regularisation We found that training using the adjoint equations was difficult due to the use of gradient penalty: this involves a double backward, and thus a double adjoint, which for reasonable step sizes we found produced inaccurate enough gradients to prevent models from training. As such our experiments do not use adjoints, instead backpropagating through the solver. This is an issue that we hope may be resolved in future work.

6 Conclusion

We have shown that SDEs and GANs follow similar formalisms. Using this connection, we train *neural SDEs* as continuous time, infinite dimensional, time series GANs. Next, we introduce a new way of sampling and reconstructing Brownian motion that is both fast and memory efficient. Finally, we show that the adjoint equations may straightforwardly be developed through rough path theory.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. volume 70 of *Proceedings of Machine Learning Research*, pp. 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*, pp. 690–701. Curran Associates, Inc., 2019.

Françios-Xavier Briol, Alessandro Barp, Andrew Duncan, and Mark Girolami. Statistical Inference for Generative Models with Maximum Mean Discrepancy. *arXiv:1906.05944*, 2020.

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*, pp. 6571–6583. Curran Associates, Inc., 2018.

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

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.

Alexander Davie. KMT theory applied to approximations of SDE. In *Stochastic Analysis and Applications 2014*, pp. 185–201. Springer, 2014.

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

Andrew Dickinson. Optimal Approximation of the Second Iterated Integral of Brownian Motion. *Stochastic Analysis and Applications*, 25(5):1109–1128, 2007.

Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Guy Flint and Terry Lyons. Pathwise approximation of SDEs by coupling piecewise abelian rough paths. *arXiv:1505.01298*, 2015.

James Foster, Harald Oberhauser, and Terry Lyons. An optimal polynomial approximation of Brownian motion. *SIAM Journal on Numerical Analysis*, 58(3):1393–1421, 2020.

Peter Friz and Martin Hairer. *A Course on Rough Paths*. Springer International Publishing, 2014.

Peter K. Friz and Nicolas B. Victoir. *Multidimensional stochastic processes as rough paths: theory and applications*. Cambridge University Press, 2010.

Jessica Gaines and Terry Lyons. Random Generation of Stochastic Area Integrals. *SIAM Journal on Applied Mathematics*, 54(4):1132–1146, 1994.

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

Patryk Gierjatowicz, Marc Sabate-Vidales, David Šiška, Lukasz Szpruch, and Žan Žurič. Robust Pricing and Hedging via Neural SDEs. *arXiv:2007.04154*, 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*, pp. 2672–2680. Curran Associates, Inc., 2014.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(1):723–773, 2013.

Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1(1):35–54, 1992.

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*, pp. 5767–5777. Curran Associates, Inc., 2017.

Jonas Haase. Limit order book system – the efficient reconstructor, 2013. URL https://lobsterdata.com/.

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.

Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A Differentiable Programming System to Bridge Machine Learning and Scientific Computing. *arXiv:1907.07587*, 2019.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *UAI*, 2018.

Junteng Jia and Austin R Benson. Neural Jump Stochastic Differential Equations. In *Advances in Neural Information Processing Systems 32*, pp. 9847–9858. Curran Associates, Inc., 2019.

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

Patrick Kidger and Terry Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. *arXiv:2001.00706*, 2020. URL https://github.com/patrick-kidger/signatory.

Patrick Kidger, Ricky T. Q. Chen, and Terry Lyons. "Hey, that's not an ODE": Faster ODE Adjoints with 12 Lines of Code. *arXiv:2009.09457*, 2020a.

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

D Kingma and J Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Franz J Király and Harald Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 2019.

Peter Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

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.

Yujia Li, Kevin Swersky, and Richard Zemel. Generative Moment Matching Networks. In *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

Xuanqing Liu, Tesi Xiao, 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.

Terry Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14, 1998.

Terry Lyons, Caruana Michael, and Lévy Thierry. *Differential equations driven by rough paths*. In École d'été de probabilités de Saint-Flour XXXIV-2004, edited by J. Picard in Volume 1908 of Lecture Notes in Mathematics, Berlin, Springer, 2007.

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

Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784*, 2014.

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.

Viktor Oganesyan, Alexandra Volokhova, and Dmitry Vetrov. Stochasticity in Neural ODEs: An Empirical Study. *arXiv:2002.09779*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

L. S. Pontryagin, E. F. Mishchenko, V. G. Boltyanskii, and R. V. Gamkrelidze. The mathematical theory of optimal processes. 1962.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal Differential Equations for Scientific Machine Learning. *arXiv:2001.04385*, 2020.

Yong Ren, Jun Zhu, Jialian Li, and Yucen Luo. Conditional Generative Moment-Matching Networks. In *Advances in Neural Information Processing Systems 29*, pp. 2928–2936. Curran Associates, Inc., 2016.

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

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

J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. *Proc. High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2011.

Csaba Toth and Harald Oberhauser. Variational Gaussian Processes with Signature Covariances. *ICML 2020*, 2020.

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

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

S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, and S.X. Chen. Cautionary Tales on Air-Quality Improvement in Beijing. *Proceedings of the Royal Society A*, 473(2205), 2017.