Neural stochastic differential equations for time series modelling

James Foster

University of Bath

(joint with Patrick Kidger, Xuechen Li,
Terry Lyons and Harald Oberhauser)

Outline

Introduction

Two dominant modelling paradigms:

Differential equations and Neural networks

Neural differential equations: awkward hybrid or perfect match?

Goal for this talk: convince you of the latter!

Introduction

Two dominant modelling paradigms:

Differential equations and Neural networks

Neural differential equations: awkward hybrid or perfect match?

Goal for this talk: convince you of the latter!

- *Neural Ordinary Differential Equations*, NeurIPS 2018.
- *Neural Controlled Differential Equations for Irregular Time Series*, NeurIPS 2020.
- *Universal Differential Equations for Scientific Machine Learning*, 2020.
- *Scalable Gradients for Stochastic Differential Equations*, AISTATS 2020.
- *Neural SDEs as Infinite-Dimensional GANs*, ICML 2021.
- *Efficient and Accurate Gradients for Neural SDEs*, NeurIPS 2021.

Outline

What is a neural differential equation?

There are differential equations where the vector field is parametrised as a neural network.

Standard example – Neural ODEs (Chen et al. 2018).

$$\frac{dy}{dt} = f_\theta(t, y(t)),$$
$$y(0) = y_0,$$

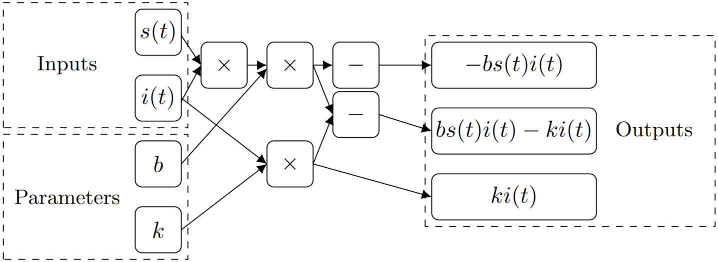
where f_θ can be any neural network (feedforward, convolutional, etc).

Examples of neural ordinary differential equations

A simple example: The SIR model for modelling infectious diseases

$$\frac{d}{dt}\begin{pmatrix} s(t) \\ i(t) \\ r(t) \end{pmatrix} = \begin{pmatrix} -bs(t)i(t) \\ bs(t)i(t) - ki(t) \\ ki(t) \end{pmatrix},$$

where b and k are parameters that are learnt from data.



At the other extreme, Neural ODEs can outperform standard machine learning models (e.g. ResNets) on tasks such as image classification [2].

Reconstruction and extrapolation of spirals with irregular time points (taken from [1])

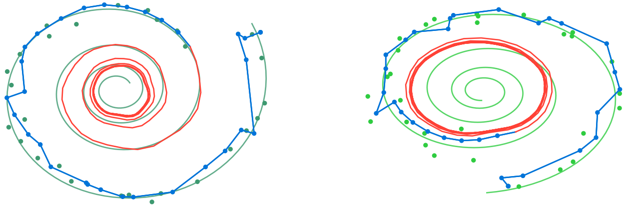
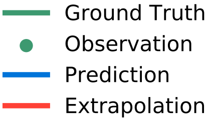


Figure: Recurrent Neural Network



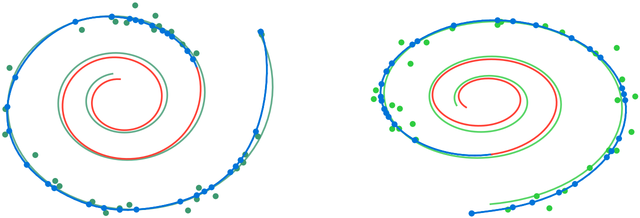


Figure: Neural ODE

Universal differential equations for scientific computing

Universal differential equations [3] are the general idea of modelling systems with

$$\frac{dy}{dt} = f_{\text{known}}(t, y(t)) + f_{\text{unknown}}(t, y(t)). \tag{1}$$

- f_{known} describes the system well and utilizes domain knowledge.
- f_{unknown} is a (small) neural network so that (1) can better fit data.

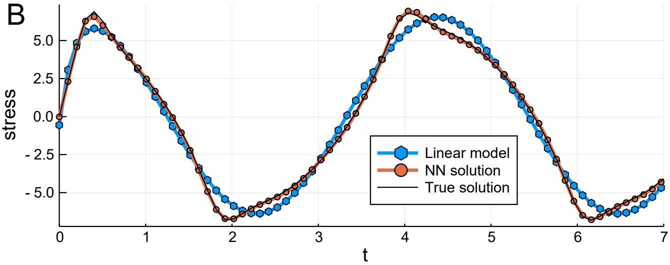


Figure: Approximating a FENE-P model for non-Newtonian fluids (from [3]).

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series
- Flexible, includes "mechanistic" and "deep" models (+ hybrids [3])

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series
- Flexible, includes "mechanistic" and "deep" models (+ hybrids [3])
- State-of-the-art ODE solvers, e.g. adaptive steps or reversibility [2]

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series
- Flexible, includes “mechanistic” and “deep” models (+ hybrids [3])
- State-of-the-art ODE solvers, e.g. adaptive steps or reversibility [2]
- Choice of ODE solver allows trade-offs between accuracy and cost

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series
- Flexible, includes "mechanistic" and "deep" models (+ hybrids [3])
- State-of-the-art ODE solvers, e.g. adaptive steps or reversibility [2]
- Choice of ODE solver allows trade-offs between accuracy and cost
- Allows for "continuous time" backpropagation with $\mathcal{O}(1)$ memory!

Why Neural ODEs?

- Continuous time, so well suited for handling (irregular) time series
- Flexible, includes "mechanistic" and "deep" models (+ hybrids [3])
- State-of-the-art ODE solvers, e.g. adaptive steps or reversibility [2]
- Choice of ODE solver allows trade-offs between accuracy and cost
- Allows for "continuous time" backpropagation with $\mathcal{O}(1)$ memory!

Potential limitation

ODEs are deterministic, so are not suitable for modelling "noisy" data.

Outline

Neural Stochastic Differential Equations

The Neural SDE takes the form

$$\begin{aligned} y_t &= \ell_\theta(x_t), \\ dx_t &= \mu_\theta(t, x_t)\,dt + \sigma_\theta(t, x_t)\,dW_t, \\ x_0 &\sim \nu_\theta(\xi), \end{aligned}$$

where

- $\mu_\theta, \sigma_\theta$ and ν_θ are neural networks.
- ℓ_θ is a linear map.
- W is a multidimensional Brownian motion.
- $\xi \sim \mathcal{N}(0, \mathrm{I}_d)$ is some initial noise.

Neural Stochastic Differential Equations

The Neural SDE takes the form

$$
\begin{aligned}
y_t &= \ell_\theta(x_t),\\
dx_t &= \mu_\theta(t, x_t)\,dt + \sigma_\theta(t, x_t)\,dW_t,\\
x_0 &\sim \nu_\theta(\xi),
\end{aligned}
$$

where

- $\mu_\theta, \sigma_\theta$ and ν_θ are neural networks.
- ℓ_θ is a linear map.
- W is a multidimensional Brownian motion.
- $\xi \sim \mathcal{N}(0, \mathrm{I}_d)$ is some initial noise.

Questions

- What does it mean for a Neural SDE to correctly model the data?
- Should we minimize mean squared error? (like for Neural ODEs)

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Some approaches:

- Match mean behaviour, i.e. minimize $|\mathbb{E}_{y\sim\text{SDE}}[F(y)] - \mathbb{E}_{y\sim\text{Data}}[F(y)]|$

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Some approaches:

- Match mean behaviour, i.e. minimize $\left|\mathbb{E}_{y\sim\text{SDE}}[F(y)] - \mathbb{E}_{y\sim\text{Data}}[F(y)]\right|$
  - F is a neural network with $\|F\| \leq 1$ trained to maximize the difference

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Some approaches:

- Match mean behaviour, i.e. minimize $|\mathbb{E}_{y\sim\text{SDE}}[F(y)] - \mathbb{E}_{y\sim\text{Data}}[F(y)]|$
  - F is a neural network with $\|F\| \leq 1$ trained to maximize the difference
  - F is defined by a reproducing kernel $k(\cdot,\cdot)$. If $F = \sum_i \alpha_i k(x_i,\cdot)$, then

$$\max_{\|F\|\leq 1}\big|\mathbb{E}_{\text{SDE}}[F(y)] - \mathbb{E}_{\text{Data}}[F(y)]\big| = \mathbb{E}_{x,x'}\big[k(x,x')\big] - 2\,\mathbb{E}_{x,y}\big[k(x,y)\big] + \mathbb{E}_{y,y'}\big[k(y,y')\big],$$

where x, x', y, y' are independent with $x, x' \sim \text{SDE}$ and $y, y' \sim \text{Data}$.

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Some approaches:

- Match mean behaviour, i.e. minimize $\left|\mathbb{E}_{y\sim\text{SDE}}[F(y)] - \mathbb{E}_{y\sim\text{Data}}[F(y)]\right|$
    - F is a neural network with $\|F\| \leq 1$ trained to maximize the difference
    - F is defined by a reproducing kernel $k(\cdot,\cdot)$. If $F = \sum_i \alpha_i k(x_i,\cdot)$, then

$$\max_{\|F\|\leq 1} \left|\mathbb{E}_{\text{SDE}}[F(y)] - \mathbb{E}_{\text{Data}}[F(y)]\right| = \mathbb{E}_{x,x'}\big[k(x,x')\big] - 2\,\mathbb{E}_{x,y}\big[k(x,y)\big] + \mathbb{E}_{y,y'}\big[k(y,y')\big],$$

where x, x', y, y' are independent with $x, x' \sim \text{SDE}$ and $y, y' \sim \text{Data}$.

    - Domain knowledge: E.g. in finance, F can be the pay-off of an option

Loss functions for Neural SDEs

We want

$$\text{Distribution}\big(\text{SDE solution}\big) \approx \text{Distribution}\big(\text{Data}\big)$$

Some approaches:

- Match mean behaviour, i.e. minimize $|\mathbb{E}_{y\sim\text{SDE}}[F(y)] - \mathbb{E}_{y\sim\text{Data}}[F(y)]|$
  - F is a neural network with $\|F\| \leq 1$ trained to maximize the difference
  - F is defined by a reproducing kernel $k(\cdot,\cdot)$. If $F = \sum_i \alpha_i k(x_i,\cdot)$, then

$$\max_{\|F\|\leq 1}\Big|\mathbb{E}_{\text{SDE}}[F(y)] - \mathbb{E}_{\text{Data}}[F(y)]\Big| = \mathbb{E}_{x,x'}\big[k(x,x')\big] - 2\,\mathbb{E}_{x,y}\big[k(x,y)\big] + \mathbb{E}_{y,y'}\big[k(y,y')\big],$$

where x, x', y, y' are independent with $x, x' \sim \text{SDE}$ and $y, y' \sim \text{Data}$.

  - Domain knowledge: E.g. in finance, F can be the pay-off of an option

- Variational inference gives lower quality SDEs, but is easier to train!

Training Neural SDEs in the Wasserstein metric

We would like to train the SDE to minimize the 1-Wasserstein distance:

$$W_1\big(\mathsf{SDE}, \mathsf{Data}\big) := \sup_{\|F\|_{\mathsf{Lipschitz}}\leq 1} \Big|\mathbb{E}_{y\sim\mathsf{SDE}}\big[F(y)\big] - \mathbb{E}_{y\sim\mathsf{Data}}\big[F(y)\big]\Big|.$$

That is, we want to find F that distinguishes between real and fake data.

Training Neural SDEs in the Wasserstein metric

We would like to train the SDE to minimize the 1-Wasserstein distance:

$$W_1\big(\mathsf{SDE}, \mathsf{Data}\big) := \sup_{\|F\|_{\mathsf{Lipschitz}} \leq 1} \Big|\mathbb{E}_{y\sim\mathsf{SDE}}[F(y)] - \mathbb{E}_{y\sim\mathsf{Data}}[F(y)]\Big|.$$

That is, we want to find F that distinguishes between real and fake data.

Some natural choices:

- Feedforward neural network
- Recurrent neural network
- Another neural differential equation!

We use the latter to define F_ϕ, which is then trained alongside the SDE.

Neural SDEs as Infinite-Dimensional GANs

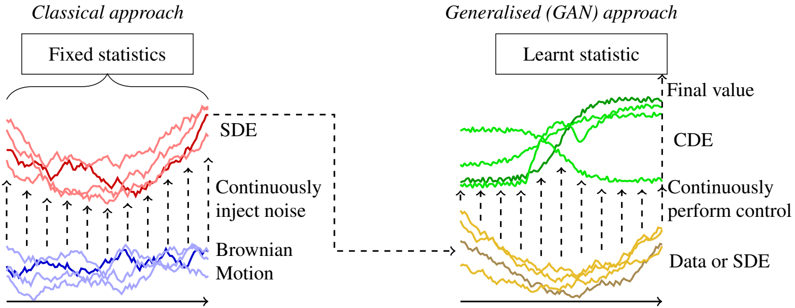
In data science, a generator (NSDE) trained with learnt discriminator(s) (F_ϕ) is known as a Generative Adversarial Network (or GAN).

They usually generate images – not time series!

Figure: Images generated by the *StyleGAN* [5]

Neural SDEs as Infinite-Dimensional GANs

	Initial	Hidden state	Output
Noise	$V \sim \mathcal{N}(0, I_v)$	W_t = Brownian motion	
Generator	$X_0 = \zeta_\theta(V)$	$\mathrm{d}X_t = \mu_\theta(t, X_t)\,\mathrm{d}t + \sigma_\theta(t, X_t) \circ \mathrm{d}W_t$	$Y_t = \alpha_\theta X_t + \beta_\theta$
Discriminator	$H_0 = \xi_\phi(Y_0)$	$\mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t) \circ \mathrm{d}Y_t$	$D = m_\phi \cdot H_T$



Outline

Numerical experiments

As a synthetic example, we generate 8192 samples $\{z_t\}_{t\in\{0,1,\cdots,63\}}$ of the time-dependent Ornstein–Uhlenbeck process:

$$dz_t = \big(\mu t - \theta z_t\big)\, dt + \sigma\, dW_t\,,$$

where $\mu = 0.02$, $\theta = 0.1$, $\sigma = 0.4$ and $z_0 \sim \mathrm{U}[-1,1]$.

We then trained a SDE-GAN with

- evolving hidden states of size 32,
- a 3-dimensional Brownian motion,
- neural networks (MLPs) with width 16 and a single hidden layer.

Numerical experiments

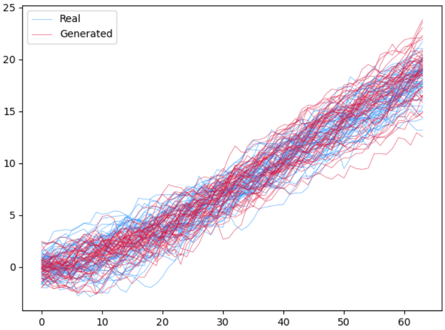


Figure: Sample paths generated by an SDE-GAN trained on an OU dataset [6]

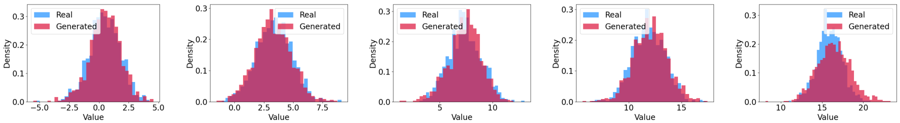


Figure: Marginal distributions at t = 6, 19, 32, 44, 57.

Numerical experiments

Next, we consider a dataset of Google/Alphabet stock prices, obtained by LOBSTER (Limit Order Book System: The Efficient Reconstructor [7])

We trained a SDE-GAN with

- evolving hidden states of size 96,
- a 3-dimensional Brownian motion,
- neural networks (MLPs) with width 64 and two hidden layers.

Numerical experiments

Next, we consider a dataset of Google/Alphabet stock prices, obtained by LOBSTER (Limit Order Book System: The Efficient Reconstructor [7])

We trained a SDE-GAN with

- evolving hidden states of size 96,
- a 3-dimensional Brownian motion,
- neural networks (MLPs) with width 64 and two hidden layers.

Metric	Neural SDE [6]	Continuous Time Flow Process [8]	Neural ODE [9]
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
Kernel distance	$\mathbf{1.92 \pm 0.09}$	2.70 ± 0.47	60.4 ± 35.8

Table: Stocks dataset: mean $\pm$ standard deviation over 3 runs. We model the 2D path consisting of the midpoint and log-spread (samples have length 100).

Conclusion

- NSDEs are continuous-time generative models for time series
- Flexible; ideas applicable to both mechanistic and deep models
- General approaches: Wasserstein GAN or Variational Inference
- NSDEs can be difficult to train! (and training can take a long time!)
- Software for neural differential equations in Python (PyTorch, Jax)
  - `https://github.com/rtqichen/torchdiffeq`
  - `https://github.com/google-research/torchsde`
  - `https://github.com/patrick-kidger/diffrax`

Thank you
for your attention!

Outline

References I

- R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud. *Neural Ordinary Differential Equations*, Neural Information Processing Systems, 2018.
- J. Zhuang, N. C. Dvornek, S. Tatikonda and J. S. Duncan. *MALI: A memory efficient and reverse accurate integrator for Neural ODEs*, International Conference on Learning Representations (ICRL), 2021.
- C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman. *Universal Differential Equations for Scientific Machine Learning*, arXiv:2001.04385, 2020.
- P. Gierjatowicz, M. Sabate-Vidales, D. Šiška, L Szpruch and Ž. Žurič. *Robust pricing and hedging via neural SDEs*, arXiv:2007.04154, 2020.

References II

- T. Karras, S. Laine and T. Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*, IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019.
- P. Kidger, J. Foster, X. Li, H. Oberhauser and T. Lyons. *Neural SDEs as Infinite-Dimensional GANs*, International Conference on Machine Learning, 2021.
- J. Haase. *Limit order book system – the efficient reconstructor*, https://lobsterdata.com, 2013.
- R. Deng, B. Chang, M. A. Brubaker, G. Mori and A. Lehrmann. *Modeling Continuous Stochastic Processes with Dynamic Normalizing Flows*, Neural Information Processing Systems, 2020.

References III

Y. Rubanova, R. T. Q. Chen and D. Duvenaud. *Latent Ordinary Differential Equations for Irregularly-Sampled Time Series*, Neural Information Processing Systems, 2019.

T. DeLise. *Neural Options Pricing*, arXiv:2105.13320, 2021.

X. Li, T.-K. L. Wong, R. T. Q. Chen and D. Duvenaud. *Scalable Gradients for Stochastic Differential Equations*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2020.

P. Kidger, J. Foster, X. Li and T. Lyons. *Efficient and Accurate Gradients for Neural SDEs*, arXiv:2105.13493, 2021.

W. Xu, R. T.Q. Chen, X. Li and D. Duvenaud. *Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations*, arXiv:2102.06559, 2021.

References IV

- P. Kidger, J. Morrill, J. Foster and T. Lyons. *Neural Controlled Differential Equations for Irregular Time Series*. Neural Information Processing Systems, 2020.
- J. Morrill, C. Salvi, P. Kidger, J. Foster and T. Lyons. *Neural Rough Differential Equations for Long Time Series*. International Conference on Machine Learning (ICML), 2021.
- J. Morrill, P. Kidger, L. Yang and T. Lyons. *Neural Controlled Differential Equations for Online Prediction Tasks*. arXiv:2106.11028, 2021.
- S. N. Cohen, C. Reisinger and S. Wang. *Arbitrage-free neural-SDE market models*, arXiv:2105.11053, 2021.