Neural Differential Equations in Machine Learning

Patrick Kidger

Mathematical Institute
University of Oxford

Random Systems CDT, October 2021

Programme

- Introduction + applications
- Neural ordinary differential equations
- Neural controlled differential equations
- Neural stochastic differential equations
- Numerical solutions to neural differential equations
- Software

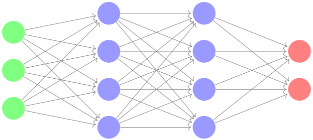
What is a neural differential equation anyway?

A differential equation with a neural network vector field.

Canonical example - neural ordinary differential equation:

$$y(0) = y_0 \qquad \frac{\mathrm{d}y}{\mathrm{d}t}(t) = f_\theta(t, y(t))$$

$f_\theta \colon \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is any standard network - for our purposes today, often a feedforward network:



(Rico-Martínez et al., Chem. Eng. Comm. 1992; Chen et al., NeurIPS 2018)

Why might we want this hybrid?

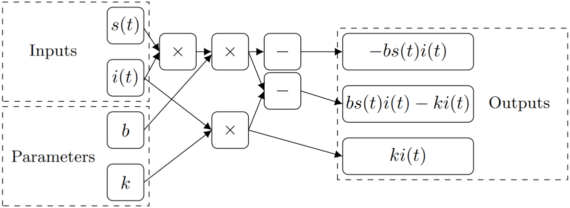
- Relative to traditional differential equations: high-capacity function approximation.
- Relative to deep learning: good priors on model space; consistent + battle-tested theory of 'what makes a good model'.

Real answer: the entire rest of this talk.

Classical example of a 'neural' differential equation: the SIR model.

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} s(t) \\ i(t) \\ r(t) \end{pmatrix} = \begin{pmatrix} -b\, s(t)\, i(t) \\ b\, s(t)\, i(t) - k\, i(t) \\ k\, i(t) \end{pmatrix}$$

b and k are parameters learnt from data.



Consider the residual network

$$y_{n+1} = y_n + f_\theta(n, y_n),$$

where $f_\theta(n, \cdot\,)$ is the nth residual block.

If we solve

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = f_\theta(t, y(t))$$

with the explicit Euler method:

$$y_{t_{n+1}} = y_{t_n} + \Delta t\, f_\theta(t_n, y_{t_n}).$$

Similar story throughout much of the rest of deep learning:

1. Other numerical solvers $\rightarrow$ other deep architectures.
2. GRUs/LSTMs $\leftrightarrow$ neural controlled differential equations;
3. StyleGAN2 $\leftrightarrow$ neural stochastic differential equations;
4. Coupling layers $\leftrightarrow$ reversible differential equation solvers;
5. . . . .

Applications

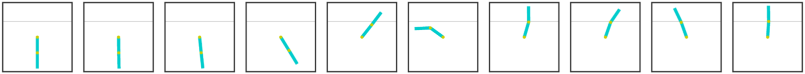
- Physical (financial, . . .) modelling, for which a differential equation is explicitly desired.
- Time series applications: data may arrive irregularly sampled, partially observed and so on.
- Generative modelling: continuous normalising flows; neural SDEs.

Hamiltonian neural networks (Greydanus et al., NeurIPS 2019)

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\partial H_\theta}{\partial p}$$
$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial H_\theta}{\partial q} + g_\theta(q)u$$

Parameterise H_θ as a neural network.

Can also include control terms (Zhong et al., ICLR 2020).



Physical (financial, biological, . . .) modelling

"Universal" differential equations: combine existing knowledge with neural network correction. (Rackauckas et al., arXiv 2020)

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = \text{known part} + \text{neural network}$$

- c.f. classical data-driven system identification.
- Nice application: closure modelling, e.g. RANS; climate models.

Physical (financial, biological, ...) modelling

Many details still not discussed:

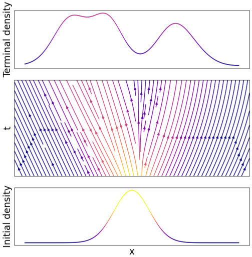
- How to train combined mechanistic/neural vector fields;
- Augmentation (i.e. Markov property);
- Choice of vector field and non-autonomy;
- Universal approximation (i.e. density in some function space).

Continuous normalising flows

Problem: want to model some unknown probability density π over some state space E.

Let ν be some 'nice' distibution e.g. a multivariate Gaussian. Then let:

$$y(0) \sim \nu \qquad \frac{\mathrm{d}y}{\mathrm{d}t}(t) = f_\theta(t, y(t))$$

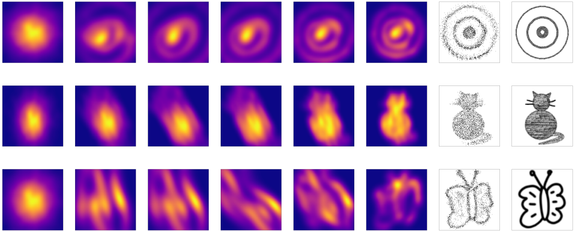


Then $y(1)$ will follow some distribution.
Let $T\colon y(0) \to y(1)$ then $y(1) \sim T^{\#}\nu$.

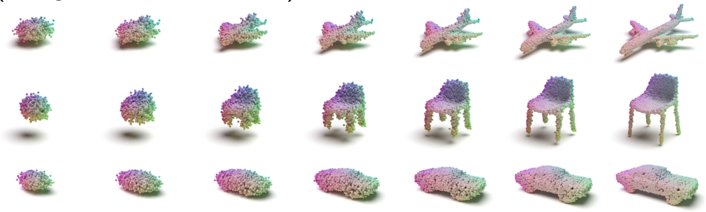
Pick θ such that $T^{\#}\nu \approx \pi$.
Solving from $t = 0$ to $t = 1$ allows for sampling.

These are *continuous normalising flows*. (Grathwohl et al., ICLR 2019)

Continuous normalising flows



(Yang et al., arXiv 2019)



Many details still not discussed:

- Training (Fokker–Planck + maximum likelihood);
- Hutchinson's trace estimator;
- Choice of vector field;
- Connections to optimal transport, SDEs, ...

Consider the (neural) ODE

$$y(t) = y(0) + \int_0^t f_\theta(y(s))\,\mathrm{d}s \quad \text{for } t \in [0, T].$$

- This takes a single input: the initial condition.
- What if we observe some ordered data $(x_0, \ldots, x_n)$?

Have the local dynamics of the system depend upon some time-varying $X\colon [0, T] \to \mathbb{R}^v$:

$$y(t) = y(0) + \int_0^t f(y(s))\,\mathrm{d}X(s) \underbrace{\int_0^t f(y(s))\,\mathrm{d}X(s)}_{=\int_0^t f(y(s))\,\frac{\mathrm{d}X}{\mathrm{d}s}(s)\,\mathrm{d}s} \qquad \text{for } t \in [0, T].$$

which is a Riemann–Stieltjes integral, and "$f(y(s))\,\mathrm{d}X(s)$" is a matrix-vector product.

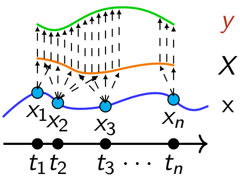
Fundamentally, a CDE is an operator $X \mapsto y$.

Let $X\colon [0, T] \to \mathbb{R}^v$ be some observed data. (e.g. an interpolation of a time series – we'll come back to this.)

Learn neural networks ζ_θ, f_θ, ℓ_θ such that

$$y(0) = \zeta_\theta(X(0)), \quad y(t) = y(0) + \int_0^t f_\theta(y(s))\,\mathrm{d}X(s),$$

and the output is either $\ell_\theta(y(T))$ or $\ell_\theta(y(t))$ (K. et al., NeurIPS 2020).



Analogous to an RNN "$y_{n+1} = f_\theta(y_n, x_n)$".

Examples: the time series $\mathrm{x} = ((t_1, x_1), \ldots, (t_n, x_n))$ could be:

- The position of a pen moved over paper, to draw a character;
- Or patient hospital records: e.g. heart rate, lab results, . . . ;
- Spoken audio;
- Weather data, e.g. temperature and pressure as they change over time;
- Physics: e.g. the movement of a double pendulum.

Comparison to control theory problems (speaking *very* broadly):

- Control theory: System f is fixed; try to find optimal X producing a desired response y.
- (Neural) CDEs: Input X is fixed; try to find optimal f_θ producing a desired response y.

Many details still not discussed:

- Good choices of f_θ;
- Why $f_\theta(y(s))\,\mathrm{d}X(s)$ and not $f_\theta(y(s), X(s))\,\mathrm{d}s$;
- Connections to rough path theory:
  - Log-ODE method: can be applied to very long (17k) time series (Morrill et al., ICML 2021a);
  - Strong theoretical guarantees e.g. neural CDEs are universal approximators;
- Choice of interpolation scheme, in particular for 'online' problems (Morrill et al., arXiv 2021b);
- Further connections to + advantages over RNNs.

(Non neural) stochastic differential equations

Consider

$$\mathrm{d}X_t = \mu(t, X_t)\,\mathrm{d}t + \sigma(t, X_t)\,\mathrm{d}W_t,$$

with X, W, μ vector-valued, and σ matrix-valued.

- The strong solution to an SDE is a map $(X_0, W) \mapsto X$.
- An SDE solver can (approximately) sample from an SDE.
- But it's tricky to write down a notion of probability density.
  (Over path space – we're looking at X, not X_T.)
- Trained by matching statistics:

  $$\mathbb{E}_X F_i(X) \approx \mathbb{E}_{\text{data}} F_i(\text{data})$$

  for all $i \in \{1, \ldots, n\}$ (e.g. with F_i called payoff functions).

Recap on GANs:

Given noise $A(\omega) \in \mathbb{R}^{d_1}$, target $B(\omega) \in \mathbb{R}^{d_2}$, a generative model is a neural network $g_\theta \colon \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ s.t. $g_\theta(A) \overset{\mathrm{d}}{\approx} B$.
(e.g. multivariate normal sample $\mapsto$ picture of a cat)

Can obtain samples as $g_\theta(A(\omega))$. But in general no tractable density $\implies$ can't train via maximum likelihood.

Train θ to minimise

$$W(g_\theta(A), B) \approx \sup_\phi \left| \frac{1}{N} \sum_{i=1}^{N} F_\phi(g_\theta(A(\omega_i)) - \frac{1}{M} \sum_{j=1}^{M} F_\phi(B(\widetilde{\omega}_j)) \right|.$$

i.e. match statistics: $\mathbb{E}_A F(g_\theta(A)) \approx \mathbb{E}_B F(B)$ for all 'discriminators' F.

Target: Want to model B, a random variable on path space.
Noise: Brownian motion W, initial noise V (e.g. with law $\mathcal{N}(0, I)$).
Let ζ_θ, μ_θ, $\sigma_\theta, \ell_\theta$ be neural networks.

Then we seek to learn an SDE of the form

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

such that $Y \overset{\mathrm{d}}{\approx} B$.

This equation has a certain minimal amount of structure: Y is the output; X is *hidden state*; V is initial noise.

Outputs of the model are continuous-time paths Y. We need a discriminator F_ϕ that accepts such objects as inputs. There is a convenient choice...

$$H_0 = \xi_\phi(Y_0), \quad \mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t)\,\mathrm{d}Y_t.$$

Then we define $F_\phi(Y) = m_\phi \cdot H_T$.

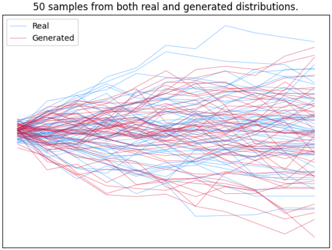
Neural stochastic differential equations

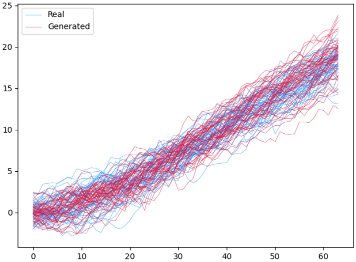
Summary of equations:

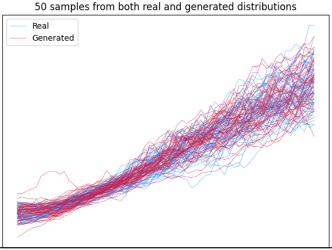
	Initial	**Hidden state**	**Output**
Noise	$V \sim \mathcal{N}(0, I)$	$W_t =$ Brownian motion	
Generator	$X_0 = \zeta_\theta(V)$	$\mathrm{d}X_t = \mu_\theta(t, X_t)\,\mathrm{d}t + \sigma_\theta(t, X_t)\,\mathrm{d}W_t$	$Y_t = \ell_\theta(X_t)$
Discriminator	$H_0 = \xi_\phi(Y_0)$	$\mathrm{d}H_t = f_\phi(t,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

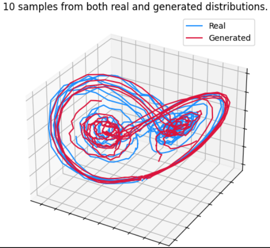
- Neural SDE / CDE form a generator/discriminator pair.
- Arbitrary drift and diffusions are admissible.
    - In the infinite data limit *any* SDE may be learnt.
- Same fundamental techniques as a 'non-neural' SDE.

Neural stochastic differential equations









Neural stochastic differential equations

Google/Alphabet Stocks:

Metric	Neural SDE	CTFP	Latent ODE
Classification	**0.357 ± 0.045**	0.165 ± 0.087	0.000239 ± 0.000086
Prediction	**0.144 ± 0.045**	0.725 ± 0.233	46.2 ± 12.3
MMD	**1.92 ± 0.09**	2.70 ± 0.47	60.4 ± 35.8

Beijing Air Quality:

Metric	Neural SDE	CTFP	Latent ODE
Classification	0.589 ± 0.051	**0.764 ± 0.064**	0.392 ± 0.011
Prediction	**0.395 ± 0.056**	0.810 ± 0.083	0.456 ± 0.095
MMD	**0.000160 ± 0.000029**	0.00198 ± 0.00001	0.000242 ± 0.000002

SGD dynamics:

Metric	Neural SDE	CTFP	Latent ODE
Classification	0.507 ± 0.019	**0.676 ± 0.014**	0.0112 ± 0.0025
Prediction	**0.00843 ± 0.00759**	0.0808 ± 0.0514	0.127 ± 0.152
MMD	**5.28 ± 1.27**	12.0 ± 0.5	23.2 ± 11.8

Many details still not discussed –

- Vector field (neural network) structure.
- Min-max training to find a Nash equilibria.
  - Optimiser: Adadelta vs SGD vs Adam vs ...
  - Stochastic weight averaging
  - ...
- Lipschitz regularisation of the discriminator.
  - Careful clipping + LipSwish activations.
  - Whole-discriminator gradient penalty.
- Alternate training strategies: KL divergence; MMD.
- Applications (in particular there's been quite a lot of finance papers).
- Connections to continuous normalising flows, ...

(K. et al., ICML 2021a; K. et al., NeurIPS 2021b)

Numerical solutions to neural differential equation

- Little structure $\implies$ typically use very general solvers.
  - Euler (probably not)
  - RK4 (much better)
  - Dormand–Prince (better still)
  - Implicit solvers?
- Somewhat unusually: *we get to control the differential equation we're solving!*
  - Can use neural network architectures that encourage dynamics that are easier to solve.
  - Can add anti-stiffness regularisers, e.g. penalise ∇f_θ to be small. (Finlay et al., ICML 2020; Kelly et al., NeurIPS 2020)

Numerical solutions to neural differential equation

- Can learn the solver too! Learn numerical solvers that do a particularly good job solving a neural differential equation. These are called hypersolvers. (Poli et al., NeurIPS 2020)
- Backpropagation through a (neural) differential equation:
  - Discretise-then-optimise;
  - Optimise-then-discretise;
  - Checkpointing (Gholami et al., arXiv 2019);
  - Interpolating adjoints;
  - Quadrature;
  - Not-an-ODE; adjoint seminorms (K. et al., ICML 2021c);
  - (Algebraically) reversible solvers (Zhuang et al., ICLR 2020; K. et al., NeurIPS 2021b);
  - Implicit function theorem (through an equilibrium).

We've got pretty good software for solving neural differential equations. (Or backpropagating through any diff. eq. really.)

- PyTorch: torchdiffeq, torchcde, torchsde;
- Julia: DifferentialEquations.jl;
- JAX: watch this space ...

Part of the PyTorch/Julia/JAX ecosystems: composable, autodifferentiable, GPU-capable.

References

R. Rico-Martínez, K. Krischer, I. G. Kevrekidis, M. C. Kube, and J. L. Hudson, "Discrete-vs. continuous-time nonlinear signal processing of Cu electrodissolution data", *Chemical Engineering Communications*, 1992

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations", *Advances in Neural Information Processing Systems*, 2018

S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian Neural Networks", *Advances in Neural Information Processing Systems*, 2019

C. Rackaucaks, 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

References

Y. D. Zhong, B. Dey, and A, Chakraborty, "Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control", *International Conference on Learning Representations*, 2020

W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form continuous dynamics for scalabe reversible generative models", *International Conference on Learning Representations*, 2019

G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan,, "PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows", *arXiv:1906.12320*, 2019

P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural Controlled Differential Equations for Irregular Time Series", *Advances in 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*, 2021a

References

J. Morrill, P. Kidger, L. Yang, and T. Lyons, "Neural Controlled Differential Equations for Online Prediction Tasks", *arXiv:2106.11028*, 2021b

P. Kidger, J. Foster, X. Li, H. Oberhauser, and T. Lyons, "Neural SDEs as Infinite-Dimensional GANs", *International Conference on Machine Learning*, 2021a

P. Kidger, J. Foster, X. Li, and T. Lyons, "Efficient and Accurate Gradients for Neural SDEs", *Neural Information Processing Systems*, 2021b

C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman, "How to train your neural ODE: the world of Jacobian and kinetic regularization", *International Conference on Machine Learning*, 2020

J. Kelly, J. Bettencourt, M. J. Johnson, and D. Duvenaud, "Learning Differential Equations that are Easy to Solve", *Advances in Neural Information Processing Systems*, 2020

M Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park, "Hypersolvers: Toward Fast Continuous-Depth Models", *Advances in Neural Information Processing Systems*, 2020

References

A. Gholami, K. Keutzer, and G. Biros, "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs", *arXiv:1902.10298*, 2019

P. Kidger, R. T. Q. Chen, T. Lyons, ""Hey, that's not an ODE": Faster ODE Adjoints via Seminorms", *International Conference on Machine Learning*, 2021c

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*, 2021

Final notes

- NDEs have applications to traditional mathematical modelling (SIR; Hamiltonian Neural Networks; Universal Differential Equations; Neural SDEs),...
- ...and to modern deep learning (Continuous Normalising Flows; Neural CDEs; Neural SDEs).

- A version of these slides are available on my website. (`https://kidger.site`)
- Feel free to send me an email / poke me on Twitter (@PatrickKidger) if you have any questions later.
- (Message me for a copy of my thesis – *On Neural Differential Equations*.)
- Any questions now?