TRAIN NEURAL DIFFERENTIAL EQUATIONS 30% FASTER WITH 12 LINES OF CODE

Patrick Kidger **Terry Lyons**
Mathematical Institute, University of Oxford
The Alan Turing Institute, The British Library
{kidger, tlyons}@maths.ox.ac.uk

ABSTRACT

Neural differential equations may be trained by backpropagating gradients via the adjoint method, which is another differential equation typically solved using an adaptive-step-size numerical differential equation solver. A proposed step is accepted if its error, *relative to some norm*, is sufficiently small; else it is rejected, the step is shrunk, and the process is repeated. Here, we demonstrate that the particular structure of the adjoint equations makes the usual choice of RMS norm unnecessarily stringent. By replacing it with a more appropriate (semi)norm, fewer steps are unnecessarily rejected and the backpropagation is made faster. This requires only minor code modifications and is compatible with existing libraries. Experiments on convolutional (classification) models, Neural CDE (time series) models and Continuous Normalising Flow (generative) models demonstrate performance improvements of X%–Y%.

1 INTRODUCTION

We begin by recalling the usual set-up for neural differential equations.

1.1 NEURAL ORDINARY DIFFERENTIAL EQUATIONS

The most popular neural differential equation is arguably the Neural Ordinary Differential Equation (Neural ODE) of E (2017); Chen et al. (2018), which seeks to approximate a map $x \mapsto y$ by learning a function $f(\,\cdot\,,\,\cdot\,,\theta)$ and linear maps $\ell_1(\,\cdot\,,\phi)$, $\ell_2(\,\cdot\,,\psi)$ such that over some interval $[\tau, T]$,

$$z(\tau) = \ell_1(x, \phi), \qquad z(t) = z(\tau) + \int_\tau^t f(s, z(s), \theta)\mathrm{d}s \quad \text{and} \quad y \approx \ell_2(z(T), \psi), \tag{1}$$

with θ, ϕ, ψ learnt parameters.

1.2 APPLICATIONS

Neural differential equations have to the best our knowledge three main applications:

1. Continuous Normalising Flows Grathwohl et al. (2019), in which the overall model acts as map between probability distributions,
2. Irregular time series, either by interleaving with RNNs as in Rubanova et al. (2019) or by taking $f(t, z, \theta) = g(z, \theta)\frac{\mathrm{d}X}{\mathrm{d}t}(t)$ to be dependent on some time-varying input X as in Kidger et al. (2020).
3. Modelling physics, for which a differential equation based model is explicitly desired. **?**

1.3 ADJOINT EQUATIONS

The integral in equation (1) may be backpropagated through either by backpropagating through the internal operations of a numerical solver, or by solving the backwards-in-time *adjoint equations* with respect to some (scalar) loss L.

$$\begin{aligned}
a_z(T) &= \frac{\mathrm{d}L}{\mathrm{d}z(T)}, & a_z(t) &= a_z(T) - \int_T^t a_z(s) \cdot \frac{\partial f}{\partial z}(s, z(s), \theta)\,\mathrm{d}s \quad \text{and} \quad & \frac{\mathrm{d}L}{\mathrm{d}z(\tau)} &= a_z(\tau),\\
a_\theta(T) &= 0, & a_\theta(t) &= a_\theta(T) - \int_T^t a_z(s) \cdot \frac{\partial f}{\partial \theta}(s, z(s), \theta)\,\mathrm{d}s \quad \text{and} \quad & \frac{\mathrm{d}L}{\mathrm{d}\theta} &= a_\theta(\tau),\\
a_t(T) &= \frac{\mathrm{d}L}{\mathrm{d}T}, & a_t(t) &= a_t(T) - \int_T^t a_z(s) \cdot \frac{\partial f}{\partial s}(s, z(s), \theta)\,\mathrm{d}s \quad \text{and} \quad & \frac{\mathrm{d}L}{\mathrm{d}\tau} &= a_t(\tau),
\end{aligned} \tag{2}$$

These equations are typically solved together as a joint system $a(t) = [a_z(t), a_\theta(t), a_t(t)]$. (They are already coupled; the latter two equations depend on a_z.) As additionally their integrands require $z(s)$, and as the results of the forward computation of equation (1) are usually not stored, then the adjoint equations are typically additionally augmented by recovering z according to the backwards-in-time equation

$$z(t) = z(T) + \int_T^t f(s, z(s), \theta)\mathrm{d}s. \tag{3}$$

1.4 Contributions

We demonstrate that the particular structure of the adjoint equations implies that numerical equation solvers will (with typical default arguments) take too many steps, that are too small, wasting time during backpropagation. Specifically, the accept/reject step of adaptive-step-size solvers is too stringent.

By applying a simple correction to account for this, we demonstrate that the number of steps needed to solve the adjoint equations can be reduced by as much as half. Factoring in the forward pass (which is unchanged), the overall training time may be improved by as much as 30%, although naturally this varies by problem.

The correction itself involves very few lines of code, and at least with the `torchdiffeq` package (our chosen differential equation package), may be implemented with only 12 lines of code, making this an easy adjustment to make to any existing codebase.

2 Method

2.1 Numerical solvers

Both the forward pass given by equation (1), and the backward pass given by equations (2) and (3), are solved by invoking a numerical differential equation solver. Our interest here is in adaptive-step-size solvers, and indeed a default choice for solving many equations is the adaptive-step-size Runge–Kutta 5(4) scheme of Dormand–Prince (**?**), for example as implemented by `dopri5` in the `torchdiffeq` package or `ode45` in MATLAB.

A full explanation of the internal operations of these solvers is beyond our scope here; the part of interest to us is the accept/reject scheme. Consider the case of solving the general ODE

$$y(t) = y(\tau) + \int_\tau^t f(s, y(s))\,\mathrm{d}s,$$

with $y(t) \in \mathbb{R}^d$.

Suppose for some fixed t the solver has computed some estimate $\widehat{y}(t) \approx y(t)$, and it now seeks to take a step $\Delta > 0$ to compute $\widehat{y}(t + \Delta) \approx y(t + \Delta)$. A step is made, and some candidate $\widetilde{y}(t + \Delta)$ is generated. The solver additionally produces $y_{\text{err}} \in \mathbb{R}^d$ representing an estimate of the numerical error made in each channel during that step.

Given some prespecified absolute tolerance $ATOL$ (for example 10^{-9}), relative tolerance $RTOL$ (for example 10^{-6}), and (semi)norm $\|\cdot\| : \mathbb{R}^d \to [0, \infty)$ (for example the RMS norm $\|y\| =$

$\sqrt{\frac{1}{d}\sum_{i=1}^{d} y_i^2}$), then an estimate of the size of the equation is given by

$$SCALE = ATOL + RTOL \cdot \max(\widehat{y}(t), \widetilde{y}(t+\Delta)) \in \mathbb{R}^d, \tag{4}$$

where the maximum is taken channel-wise, and the error ratio

$$r = \left\| \frac{y_{\text{err}}}{SCALE} \right\| \tag{5}$$

is then computed. If $r \leq 1$ then the error is deemed acceptable, the step is accepted and we take $\widehat{y}(t+\Delta) = \widetilde{y}(t+\Delta)$. If $r > 1$ then the error is deemed too large, the candidate $\widetilde{y}(t+\Delta)$ is rejected, and the procedure is repeated with a smaller Δ.

Note the dependence on the choice of norm $\|\cdot\|$: in particular this determines the relative importance of each channel towards the accept/reject criterion.

2.2 Adjoint norms

In equation (2), we observe that $a_\theta(T) = 0$. When solving equation (2) numerically, this means for t close to T that the second term in equation (4) is small. As $ATOL$ is also typically very small, then $SCALE$ is small and the norm computed in equation (5) becomes very large.

This implies that it becomes easy for the error ratio r to violate $r \leq 1$, and it is easy for the step to be rejected.

The sign that something may be amiss with this strategy is that a_θ does not appear anywhere in the vector fields on equation (2). Improving the accuracy of a_θ will not help improve the accuracy with which the ODE is solved (except incidentally, by increasing the resolution of the other channels, by having forced smaller step sizes).

It *will* improve the accuracy with which we compute ${}^{\mathrm{d}L}/_{\mathrm{d}\theta} = a_\theta(\tau)$, but the improvement is marginal: when a_θ grows larger for t away from T, then $SCALE$ grows larger, steps are accepted more easily, the level of noise in the estimates grows, and the additional accuracy we gained at the start is lost. There was little reason to be so strict in the first place.

Thus, we come to our proposed strategy: when solving the adjoint equations equation (2), choose a $\|\cdot\|$ that scales down the effect in those channels corresponding to a_θ.

Additionally, software may automatically compute a_t even if the desired gradient is never used, for example because τ and T are fixed: in such a scenario then much the same argument applies, and this channel may be scaled as well.

In practice, in our experiments, we scale $\|\cdot\|$ all the way down by applying zero weight to the offending channels, so that $\|\cdot\|$ is in fact a seminorm.

2.3 Code

Depending on the software package, the code for making this change can be trivial. For example, using PyTorch and `torchdiffeq` (Chen et al., 2018), the standard set-up

```python
import torchdiffeq

func = ...
y0 = ...
t = ...
torchdiffeq.odeint_adjoint(func=func, y0=y0, t=t)
```

is just changed to

```python
import torchdiffeq

def _rms_norm(tensor):
    return tensor.pow(2).mean().sqrt()

def make_norm(state):
    state_size = state.numel()
    def _norm(aug_state):
        y = aug_state[1:1 + state_size]
        adj_y = aug_state[1 + state_size:1 + 2 * state_size]
        return max(_rms_norm(y), _rms_norm(adj_y))
    return _norm

func = ...
y0 = ...
t = ...
torchdiffeq.odeint_adjoint(func=func, y0=y0, t=t,
                           adjoint_options=dict(norm=make_norm(y0)))
```

and that's it.

To keep the remainder of this discussion software-agnostic, we defer further explanation of this specific code to Appendix A.

3 Experiments

We demonstrate our proposed technique by comparing it against a conventionally-trained neural differential equation, across multiple regimes – classification, time series, and generative.

TODO: discuss architectures, normalisation etc.

3.1 Convolutional classification model

TODO: experiments ongoing (effect does seem to be present)

3.2 Neural Controlled Differential Equations

This technique applies to more than just standard Neural ODE models. Consider the Neural Controlled Differential Equation (Neural CDE) model of Kidger et al. (2020).

To recap, given some (potentially irregularly sampled) time series $\mathbf{x} = ((t_0, x_0), \ldots, (t_n, x_n))$, with each $t_i \in \mathbb{R}$ the timestamp of the observation $x_i \in \mathbb{R}^v$, let $X\colon [t_0, t_n] \to \mathbb{R}^{1+v}$ be an interpolation such that $X(t_i) = (t_i, x_i)$. For example X could be a natural cubic spline.

Then take $f(t, z, \theta) = g(z, \theta)\frac{\mathrm{d}X}{\mathrm{d}t}(t)$ in a Neural ODE model, so that changes in $\mathbf{x}$ provoke changes in the vector field, and the model incorporates the incoming information $\mathbf{x}$. The output is $z(T)$ as in the Neural ODE, and the model may be thought of as a continuous-time RNN.

As this may be interpreted as an ODE (and indeed solved with the same software), then the same technique applies.

We apply a Neural CDE to the Speech Commands dataset (**?**), using the same hyperparameters as in Kidger et al. (2020) (see Appendix B for details). This is a dataset of one-second audio recordings of spoken words such as 'left', 'right' and so on. We use 34975 time series corresponding to 10 spoken words so as to produce a balanced classification problem. We preprocess the dataset by computing mel-frequency cepstrum coefficients so that each time series is then regularly spaced with length 161 and 20 channels.

We demonstrate how the effect changes for varying tolerances by considering each pairs

$$ATOL = 10^{-6}, RTOL = 10^{-3},$$
$$ATOL = 10^{-7}, RTOL = 10^{-4},$$
$$ATOL = 10^{-8}, RTOL = 10^{-5},$$

and for each such pair run five repeated experiments.

See Table 1. We see that the accuracy of the model is unaffected by our proposed change. The backward pass requires roughly half the number of steps, and the overall training time (factoring in the forward pass that is left unchanged) is reduced by approximately 30%.

Table 1: Across the test set: accuracy; number of function evaluations (NFE) to calculate an adjoint (backward) pass; total time to perform forward *and* backward passes.[1]

	$ATOL = 10^{-6}, RTOL = 10^{-3}$		
	Accuracy (%)	Backward NFE (10^3)	Forward + Backward Time (seconds)
Neural CDE (Default norm)	**92.6** ± **0.4**	21.5 ± 1.5767	275.4 ± 15.5
Neural CDE (Our norm)	**92.5** ± **0.5**	**12.5** ± **0.9625**	**212.2** ± **11.4**
	$ATOL = 10^{-7}, RTOL = 10^{-4}$		
	Accuracy (%)	Backward NFE (10^3)	Forward + Backward Time (seconds)
Neural CDE (Default norm)	TODO	TODO	TODO
Neural CDE (Our norm)	TODO	TODO	TODO
	$ATOL = 10^{-8}, RTOL = 10^{-5}$		
	Accuracy (%)	Backward NFE (10^3)	Forward + Backward Time (seconds)
Neural CDE (Default norm)	TODO	TODO	TODO
Neural CDE (Our norm)	TODO	TODO	TODO

3.3 Continuous Normalising Flows

4 Related work

5 Conclusion

References

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.

Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Commun. Math. Stat.*, 5(1): 1–11, 2017.

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.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural Controlled Differential Equations for Irregular Time Series. *arXiv:2005.08926*, 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*, pp. 5320–5330. Curran Associates, Inc., 2019.

[1] For the avoidance of doubt: these backward passes were performed on the test set purely to evaluate their speed, and the calculated gradients were not used to train the model.

A Code for torchdiffeq

TODO

B Experiments