Neural CDEs for Long Time Series via the Log-ODE Method

James Morrill **Patrick Kidger** **Cristopher Salvi** **James Foster** **Terry Lyons**
Mathematical Institute, University of Oxford;
The Alan Turing Institute, British Library
{morrill, kidger, salvi, foster, tlyons}@maths.ox.ac.uk

Abstract

Neural Controlled Differential Equations (Neural CDEs) are the continuous-time analogue of an RNN, just as Neural ODEs are analogous to ResNets. However just like RNNs, training Neural CDEs can be difficult for long time series. Here, we propose to apply a technique drawn from stochastic analysis, namely the log-ODE method. Instead of using the original input sequence, our procedure summarises the information over local time intervals via the log-signature map, and uses the resulting shorter stream of log-signatures as the new input. This represents a length/channel trade-off. In doing so we demonstrate efficacy on problems of length up to 17k observations and observe significant training speed-ups, improvements in model performance, and reduced memory requirements compared to the existing algorithm.

1 Introduction

Neural controlled differential equations (Neural CDEs) [1] are the continuous-time analogue to a recurrent neural network (RNN), and provide a natural method for modelling temporal dynamics with neural networks. In contrast to Neural ODEs [2] the vector field of a Neural CDE depends upon the time-varying data, so that the trajectory of the system is driven by a sequence of observations.

We recall the definition of a Neural CDE as introduced in [1]. Consider a time series $\mathbf{x}$ as a collection of points $x_i \in \mathbb{R}^{v-1}$ with corresponding time-stamps $t_i \in \mathbb{R}$ such that $\mathbf{x} = ((t_0, x_0), (t_1, x_1), ..., (t_n, x_n))$, and $t_0 < ... < t_n$. Let $X \colon [t_0, t_n] \to \mathbb{R}^v$ be some interpolation of the data such that $X_{t_i} = (t_i, x_i)$. Here we use a linear interpolation scheme.

Let $\xi_\theta \colon \mathbb{R}^v \to \mathbb{R}^w$ and $f_\theta \colon \mathbb{R}^w \to \mathbb{R}^{w \times v}$ be neural networks and let $\ell_\theta \colon \mathbb{R}^w \to \mathbb{R}^q$ be linear.

We define Z as the hidden state and Y as the output of a *neural controlled differential equation* driven by X if

$$Z_{t_0} = \xi_\theta(t_0, x_0), \quad \text{with} \quad Z_t = Z_{t_0} + \int_{t_0}^t f_\theta(Z_s)\mathrm{d}X_s \quad \text{and} \quad Y_t = \ell_\theta(Z_t) \quad \text{for } t \in (t_0, t_n]. \tag{1}$$

The integral of equation (1) is a Riemann–Stieltjes integral. In the original paper [1], the authors noted that, assuming differentiability of the path, $\mathrm{d}X_s$ can be replaced with $\frac{\mathrm{d}X}{\mathrm{d}s}(s)\mathrm{d}s$, reducing the CDE onto an ODE. Existing tools for Neural ODEs may be used to evaluate this, and to backpropagate.

By moving from the discrete-time formulation of an RNN to the continuous-time formulation of a Neural CDE, then every kind of time series data is put on the same footing, whether it is regularly or irregularly sampled, whether or not it has missing values, and whether or not the input sequences are of consistent length.

1.1 Contributions

Neural CDEs, as with RNNs, begin to break down for long time series. Here, we apply the *log-ODE method*, which is a numerical method from stochastic analysis and rough analysis. It is a method for converting a CDE to an ODE, acting as a drop-in replacement for the original procedure that uses the derivative of the control path.

This may be interpreted as taking integration steps larger than the discretisation of the data, incorporating substep information through higher-order correction terms. See Figure 1.

We find that this method is particularly beneficial for long time series (and incidentally does not require differentiability of the control path). With this method both training time and model performance of Neural CDEs are improved, and memory requirements are reduced.

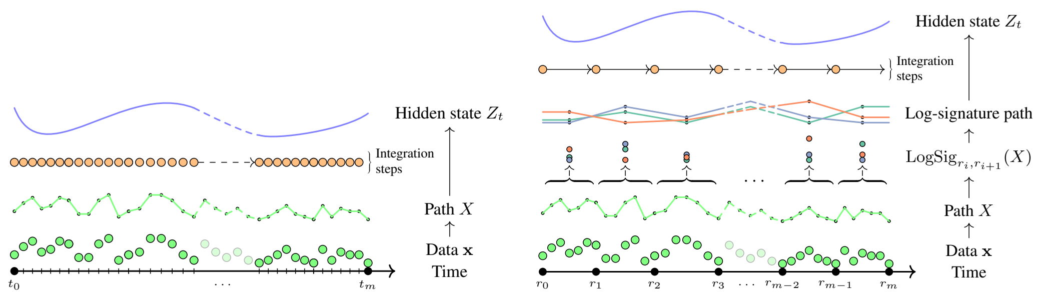

Figure 1: **Left:** The original Neural CDE formulation. The path X is quickly varying, meaning a lot of integration steps are needed to resolve it. **Right:** The log-ODE method. The log-signature path is more slowly varying (in a higher dimensional space), and needs fewer integration steps to resolve.

2 Method

Recall that we observe some time series $\mathbf{x} = ((t_0, x_0), (t_1, x_1), ..., (t_n, x_n))$, and have constructed a piecewise linear interpolation $X\colon [t_0, t_n] \to \mathbb{R}^v$ such that $X_{t_i} = (t_i, x_i)$.

We now pick points r_i such that $t_0 = r_0 < r_1 < \cdots < r_m = t_n$. In principle these can be variably spaced but in practice we will typically space them equally far apart. The total number of points m should be much smaller than n.

2.1 Theory

The Log-Signature For $X\colon [t_0, t_n] \to \mathbb{R}^v$ and $t_0 \leq r_i < r_{i+1} \leq t_n$ then the depth-N log-signature of X over the interval $[r_i, r_{i+1}]$, denoted $\mathrm{LogSig}^N_{r_i,r_{i+1}}(X) \in \mathbb{R}^{\beta(v,N)}$, is a collection of summary statistics about the path X; see [3] for a formula for β. We omit the full mathematical description here but it suffices to understand that the log-signature is a vector of real values that give a description of a path over an interval. Intuitively speaking, these values describe the information that is most important for solving the CDE equation over that interval.

The Log-ODE Method Let $\widehat{f} : \mathbb{R}^w \to \mathbb{R}^{w \times v}$. The log-ODE method states

$$Z_b \approx \widehat{Z}_b \quad \text{where} \quad \widehat{Z}_u = \widehat{Z}_a + \int_a^u \widehat{f}(\widehat{Z}_s) \frac{\mathrm{LogSig}^N_{a,b}(X)}{b-a} \mathrm{d}s, \quad \text{and} \quad \widehat{Z}_a = Z_a. \qquad (2)$$

where Z is as defined in equation (1), and the relationship between $\widehat{f}$ to f is given in [4].

The approximation becomes arbitrarily good as $N \to \infty$. That is, the solution of the CDE may be approximated by the solution to an ODE. We will additionally consider a relaxation, by modelling $\widehat{f}$ directly as a neural network.

2.2 Updating the Neural CDE Hidden State Equation

Recall in the original formulation that $\mathrm{d}X_s$ was replaced with $\frac{\mathrm{d}X}{\mathrm{d}s}(s)\mathrm{d}s$. For the log-ODE approach, over each interval $[r_i, r_{i+1}]$, we instead replace $f_\theta(Z_s)dX_s$ with $\widehat{f}_\theta(Z_s)\frac{\mathrm{LogSig}^N_{r_i,r_{i+1}}(X)}{r_{i+1}-r_i}\mathrm{d}s$.

That is, defining the piecewise $\widehat{g}(X,s) = \widehat{f}_\theta(Z_s)\frac{\mathrm{LogSig}^N_{r_i,r_{i+1}}(X)}{r_{i+1}-r_i}$ for $s \in [r_i, r_{i+1})$ we may now instead solve the Neural CDE via the log-ODE equation

$$Z_{t_0} = \xi_\theta(t_0, x_0), \quad \text{with} \quad Z_t = Z_{t_0} + \int_{t_0}^{t} \widehat{g}(X,s)\mathrm{d}s \quad \text{and} \quad Y_t = \ell_\theta(Z_t) \quad \text{for } t \in (t_0, t_n]. \tag{3}$$

2.3 Discussion

Ease of Implementation This method is straightforward to implement using pre-existing tools and is available in open source torchcde project.

Length/Channel Trade-Off The sequence of log-signatures is now of length m, which was chosen to be much smaller than n. As such, it is much more slowly varying over the interval $[t_0, t_n]$ than the original data, which was of length n. The differential equation it drives is better behaved, and so larger integration steps may be used in the numerical solver. This is the source of the speed-ups of this method; we observe typical speed-ups by a factor of about 100.

Generality of the Log-ODE Method If depth $N = 1$ and steps $r_i = t_i$ are used, then the above formulation exactly reduces onto the original Neural CDE formulation using linear interpolation. Thus the log-ODE method in fact generalises the original approach.

Applications In principle the log-ODE method may be applied to solve any Neural CDE. In practice, the reduction in length (from n to m), coupled with the loss of information (from using the log-signature as a summary statistic) makes this particularly useful for long time series.

Memory Efficiency Long sequences need large amounts of memory to perform backpropagation-through-time (BPTT). As with the original Neural CDEs, the log-ODE approach supports memory-efficient backpropagation via the adjoint equations, alleviating this issue. See [1].

3 Experiments

We investigate solving a Neural CDE with and without the log-ODE method on four real-world problems. Every problem was chosen for its long length. The lengths are in fact sufficiently long that adjoint-based backpropagation [2] was needed to avoid running out of memory at any reasonable batch size. Every problem is regularly sampled, so we take $t_i = i$.

We will denote a Neural CDE model with log-ODE method, using depth N and step s, as NCDE^s_N. Taking $N = 1$ (and any s) corresponds to not using the log-ODE method, with the data subsampled at rate $1/s$, as per section 2.3. We use NCDE^1_1 as our benchmark: no subsampling, no log-ODE method.

3.1 Classifying EigenWorms

Our first example uses the EigenWorms dataset from the UEA archive [5]. This consists of time series of length 17 984 and 6 channels, corresponding to the movement of a roundworm. The goal is to classify each worm as either wild-type or one of four mutant-type classes.

See Table 1. We see that the straightforward NCDE^1_1 model takes roughly a day to train. Using the log-ODE method (NCDE_2, NCDE_3) speeds this up to take roughly minutes. Doing so additionally improves model performance dramatically, and reduces memory usage. Naive subsampling approaches (NCDE^8_1, NCDE^{32}_1, NCDE^{128}_1) only achieve speed-ups without performance improvements.

Model	Step	Test Accuracy (%)	Time (Hrs)	Memory (Mb)
NCDE_1	1	62.4 ± 12.1	22.0	176.5
	8	64.1 ± 13.3	3.1	24.3
	32	64.1 ± 14.3	0.5	8.0
	128	48.7 ± 2.6	0.1	3.9
NCDE_2	8	**77.8 ± 5.9**	2.1	94.2
	32	67.5 ± 12.1	0.7	28.1
	128	**76.1 ± 5.9**	0.2	7.8
NCDE_3	8	70.1 ± 6.5	1.3	460.7
	32	**75.2 ± 3.0**	0.6	134.7
	128	68.4 ± 8.2	0.1	53.3

Table 1: Mean and standard deviation of test set accuracy (in %) over three repeats, as well as memory usage and training time, on the EigenWorms dataset for depths 1–3 and a small selection of step sizes. The bold values denote that the model was the top performer for that step size.

Depth	Step	$\mathbf{L^2}$			Time (H)			Memory (Mb)
		RR	HR	SpO_2	RR	HR	SpO_2	
NCDE_1	1	2.79 ± 0.04	9.82 ± 0.34	2.83 ± 0.27	23.8	22.1	28.1	56.5
	8	2.80 ± 0.06	10.72 ± 0.24	3.43 ± 0.17	3.0	2.6	4.8	14.3
	32	2.53 ± 0.23	12.23 ± 0.43	2.68 ± 0.12	1.9	0.9	2.2	9.8
	128	2.64 ± 0.18	11.98 ± 0.37	2.86 ± 0.04	0.2	0.2	0.3	8.7
NCDE_2	8	2.63 ± 0.12	8.63 ± 0.24	2.88 ± 0.15	2.1	3.4	3.3	21.8
	32	1.90 ± 0.02	7.90 ± 1.00	1.69 ± 0.20	1.2	1.1	2.0	13.1
	128	1.86 ± 0.03	6.77 ± 0.42	1.95 ± 0.18	0.3	0.4	0.7	10.9
NCDE_3	8	**2.42 ± 0.19**	**7.67 ± 0.40**	**2.55 ± 0.13**	2.9	3.2	3.1	43.3
	32	**1.67 ± 0.01**	**4.50 ± 0.70**	**1.61 ± 0.05**	1.3	1.8	7.3	20.5
	128	**1.51 ± 0.08**	**2.97 ± 0.45**	**1.37 ± 0.22**	0.5	1.7	1.7	17.3

Table 2: Mean and standard deviation of the L^2 losses on the test set for each of the vitals signs prediction tasks (RR, HR, SpO_2) on the BIDMC dataset, across three repeats. Only mean times are shown for space. The memory usage is given as the mean over all three of the tasks as it was approximately the same for any task for a given depth and step. The bold values denote the algorithm with the lowest test set loss for a fixed step size for each task.

3.2 Estimating Vitals Signs from PPG and ECG data

Next we consider the problem of estimating vital signs from PPG and ECG data. This comes from the TSR archive [6] using data from the Beth Israel Deaconess Medical Centre (BIDMC). We consider three separate tasks, in which we aim to predict a person's respiratory rate (RR), their heart rate (HR), and their oxygen saturation (SpO2). The data is sampled at 125Hz, with each series having an overall length of 4 000. There are 7 949 training samples, and 3 channels (including time). The metric used to evaluate performance is the L^2 loss. The results over a range of step sizes are presented in table 2.

We find that the depth 3 model is the top performer for every task at any step size. What's more, it does so with a significantly reduced training time. We attribute the improved performance to the log-ODE model being better able to learn long-term dependencies due to the reduced sequence length. Note that the performance of the NCDE_2^s, NCDE_3^s models actually improves as the step size is increased. This is in contrast to NCDE_1^s, which sees a degradation in performance.

4 Conclusion

We demonstrate how to effectively apply Neural CDEs to long (17k) time series, using the log-ODE method. The model may still be solved via ODE methods and thus retains adjoint backpropagation and continuous dynamics. In doing so we see significant training speed-ups, improvements to model performance, and reduced memory requirements.

Broader Impact

As data becomes increasingly abundant – in particular high frequency sensor data – we hope that the present paper will help to address some of the challenges that this data poses. This will likely benefit the time series community. We do not identify anyone who may be specifically disadvantaged by this research, or any specific negative impacts due to the introduction of this method.

Acknowledgements

JM was supported by the EPSRC grant EP/L015803/1 in collaboration with Iterex Therapuetics. PK was supported by the EPSRC grant EP/L015811/1. CS was supported by the EPSRC grant EP/R513295/1. JF was supported by the EPSRC grant EP/N509711/1. JM, CS, PK, JF, TL were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

[1] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural controlled differential equations for irregular time series," *arXiv preprint arXiv:2005.08926*, 2020.

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

[3] J. Reizenstein, "Calculation of Iterated-Integral Signatures and Log Signatures," *arXiv preprint arXiv:1712.02757*, 2017.

[4] T. Lyons, "Rough paths, signatures and the modelling of functions on streams," *Proceedings of the International Congress of Mathematicians*, vol. 4, 2014.

[5] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 2017.

[6] C. W. Tan, A. Bagnall, C. Bergmeir, E. Keogh, F. Petitjean, and G. I. Webb, "Monash university, uea, ucr time series regression archive," 2020. `http://timeseriesregression.org/`.