Neural Rough Differential Equations for Long Time Series

James Morrill [1 2] **Cristopher Salvi** [1 2] **Patrick Kidger** [1 2] **James Foster** [1 2]

Abstract

Neural controlled differential equations (CDEs) are the continuous-time analogue of recurrent neural networks, as Neural ODEs are to residual networks, and offer a memory-efficient continuous-time way to model functions of potentially irregular time series. Existing methods for computing the forward pass of a Neural CDE involve embedding the incoming time series into path space, often via interpolation, and using evaluations of this path to drive the hidden state. Here, we use rough path theory to extend this formulation. Instead of directly embedding into path space, we instead represent the input signal over small time intervals through its *log-signature*, which are statistics describing how the signal drives a CDE. This is the approach for solving *rough differential equations* (RDEs), and correspondingly we describe our main contribution as the introduction of Neural RDEs. This extension has a purpose: by generalising the Neural CDE approach to a broader class of driving signals, we demonstrate particular advantages for tackling long time series. In this regime, 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 existing approaches.

1. Introduction

Neural controlled differential equations (CDEs) (Kidger et al., 2020) are the continuous-time analogue to recurrent neural networks (RNNs) and provide a natural method for modelling temporal dynamics with neural networks.

Neural CDEs are similar to neural ordinary differential equations (ODEs), as popularised by Chen et al. (2018). A Neural ODE is determined by its initial condition, without a direct way to modify the trajectory given subsequent observations. In contrast, 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.

1.1. Controlled Differential Equations

Let $a, b \in \mathbb{R}$ with $a < b$, and let $v, w \in \mathbb{N}$. Let $\xi \in \mathbb{R}^w$. Let $X \colon [a, b] \to \mathbb{R}^v$ be a continuous function of bounded variation (which is for example implied by it being Lipschitz), and let $f \colon \mathbb{R}^w \to \mathbb{R}^{w \times v}$ be continuous.

Then we may define $Z \colon [a, b] \to \mathbb{R}^w$ as the unique solution to the *controlled differential equation*

$$Z_a = \xi, \quad Z_t = Z_a + \int_a^t f(Z_s)\,\mathrm{d}X_s \quad \text{for } t \in (a, b]. \tag{1}$$

The notation "$f(Z_s)\mathrm{d}X_s$" denotes a matrix-vector product. "$\mathrm{d}X_s$" itself denotes a Riemann–Stieltjes integral: if X is differentiable then

$$\int_a^t f(Z_s)\mathrm{d}X_s = \int_a^t f(Z_s)\dot{X}_s\mathrm{d}s, \quad \text{with } \dot{X}_s = \frac{\mathrm{d}X_r}{\mathrm{d}r}(s). \tag{2}$$

If in equation (1), $\mathrm{d}X_s$ was replaced with $\mathrm{d}s$, then the equation would just be an ODE. Using $\mathrm{d}X_s$ causes the solution to depend continuously on the evolution of X. We say that the solution is "driven by the control X".

Next, we recall the definition of a Neural CDE as introduced in Kidger et al. (2020).

1.2. Neural Controlled Differential Equations

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 $X_{t_i} = (t_i, x_i)$. In Kidger et al. (2020) the authors use natural cubic splines to ensure differentiability of the control X, so as to treat the term "$\mathrm{d}X_s$" in equation (1) as "$\dot{X}_s\mathrm{d}s$".

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 two neural networks and let $\ell_\theta \colon \mathbb{R}^w \to \mathbb{R}^q$ be a linear map, for some

*Equal contribution [1]Mathematical Institute, University of Oxford, UK [2]The Alan Turing Institute, British Library, UK. Correspondence to: James Morrill <morrill@maths.ox.ac.uk>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

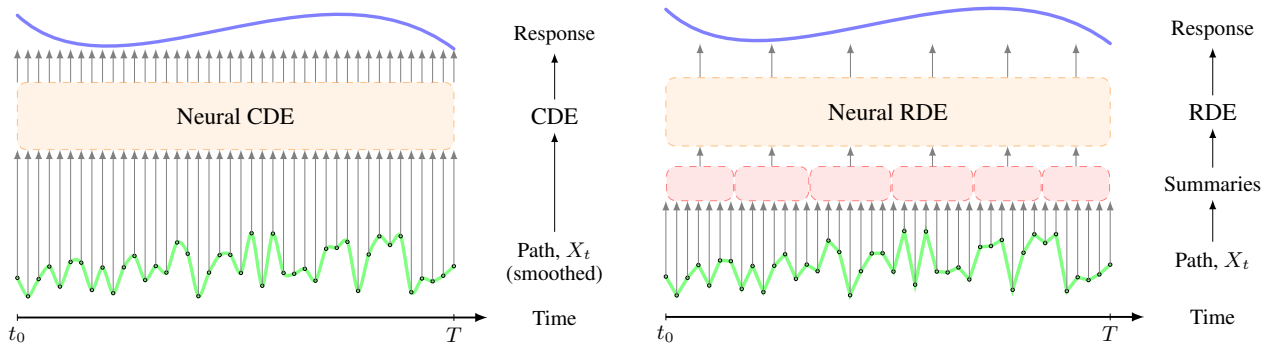

Figure 1. Here we give a high level comparison of the CDE and the RDE formulations. **Left:** The original CDE formulation where the data is smoothly interpolated and pointwise derivative information is used to drive the CDE. **Right:** The corresponding rough approach. Local interval summarisations of the data are computed and used to drive the response over the interval.

output dimension $q \in \mathbb{N}$. Here θ is used to denote dependence on learnable parameters.

We define Z as the hidden state and Y as the output of a Neural CDE driven by X if

$$Z_{t_0} = \xi_\theta(t_0, x_0), \text{ with } Z_t = Z_{t_0} + \int_{t_0}^{t} f_\theta(Z_s)\,\mathrm{d}X_s, \\ \text{and } Y_t = \ell_\theta(Z_t) \text{ for } t \in (t_0, t_n] \quad (3)$$

That is – just like an RNN – we have an evolving hidden state Z, which is fed into a linear map to produce an output Y. This formulation is a universal approximator (Kidger et al., 2020, Appendix B). The output may be either the time-evolving Y_t or just the final Y_{t_n}. This is then fed into a loss function (L^2, cross entropy, ...) and trained via stochastic gradient descent in the usual way.

To compute the integral of equation (3) in Kidger et al. (2020), X is assumed differentiable and the CDE is simply rewritten as an ODE of the form

$$Z_t = Z_{t_0} + \int_{t_0}^{t} g_{\theta,X}(Z_s, s)\,\mathrm{d}s, \quad (4)$$

where

$$g_{\theta,X}(Z, s) = f_\theta(Z)\dot{X}_s. \quad (5)$$

This simple observation allows for incorporating the time-varying data X driving the CDE into the vector field $g_{\theta,X}$ of the equivalent ODE (4). In doing so existing tools for Neural ODEs can be used to carry out the forward pass and backpropagate via adjoint methods.

1.3. Contributions

Neural CDEs, as with RNNs, begin to break down for long time series. Loss/accuracy worsens, and training time becomes prohibitive due to the sheer number of forward operations within each training epoch.

Meanwhile, and at first glance tangentially, it is known in the field of rough path theory (Lyons, 1998; Lyons et al., 2004; Friz & Victoir, 2010) that it is possible to numerically solve CDEs not by pointwise evaluations of the control path (as in the existing Neural CDE approach), but by using a specific summarisation – known as *the log-signature* – of the control path over short time intervals. See Figure 1. A CDE treated in this way is termed a *rough differential equation*, and the numerical method is termed *the log-ODE method*.

The central contribution of this paper is to observe that this latter technique actually offers a way to solve the former problem. The log-ODE method offers a way to update the hidden state of a Neural CDE over large intervals – much larger than would be expected given the sampling rate or length of the data. This dramatically reduces the effective length of the time series. Log-signatures represents a CDE-specific choice of summarisation, which works because closely-spaced samples are often strongly correlated. Additionally, this approach no longer requires differentiability of the control path.

In line with the usual mathematical terminology, we refer to our approach as *neural rough differential equations* (Neural RDEs). Moreover, Neural RDEs are still able to exploit memory-efficient continuous-time adjoint backpropagation. This is of additional benefit as memory pressure becomes increasingly relevant for long time series – indeed many of our experiments could not have been ran without it.

With Neural RDEs, we demonstrate improvements experimentally on real-world problems of length up to 17 000. We report substantial improvements in model performance (by as much as 17% on some classification tasks, reflecting the difficulty inherent in long time series), speed (by roughly a factor of 10), and memory usage (by roughly a factor of 100 compared to models not using the adjoint method).

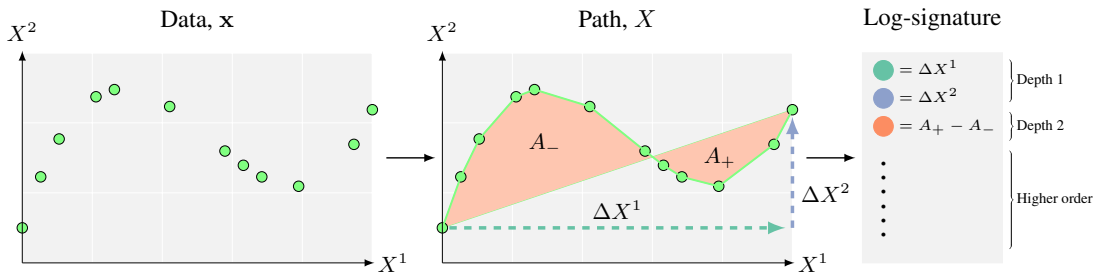

Figure 2. Geometric intuition for the first two levels of the log-signature for a 2-dimensional path. The depth 1 terms correspond to the change in each of the coordinates over the interval. The depth 2 term corresponds to the *Lévy area* of the path, this being the signed area between the curve and the chord joining its start and endpoints.

2. Theory

We begin with an exposition on the motivating theory. Our description here will focus on the high-level intuitions. For a full technical description we refer to the appendices; see also section 7.1 of (Friz & Victoir, 2010).

Readers primarily interested in practical applications should feel free to skip to section 3.

2.1. Signatures and Log-signatures

The signature transform is a map from paths to a vector of real values, specifying a collection of statistics about the path. It is a central component of the theory of controlled differential equations since these statistics describe how the data interacts with dynamical systems. The log-signature is then formed by representing the same information in a compressed format.

Signature transform Let $X = (X^1, ..., X^d) : [0, T] \to \mathbb{R}^d$ be continuous and piecewise differentiable.[1] Letting[2]

$$S^{i_1,...i_k}_{a,b}(X) = \int_{a<t_1<...<t_k<b} ... \int \prod_{j=1}^{k} \frac{\mathrm{d}X^{i_j}}{\mathrm{d}t}(t_j)\mathrm{d}t_j, \quad (6)$$

then the depth-N signature transform of X is given by

$$\mathrm{Sig}^N_{a,b}(X) = \Big(\big\{S^i_{a,b}(X)^{(i)}\big\}^d_{i=1}, \big\{S^{i,j}_{a,b}(X)\big\}^d_{i,j=1}, \ldots, \big\{S^{i_1,...,i_N}_{a,b}(X)\big\}^d_{i_1,...,i_N=1}\Big). \quad (7)$$

This definition is independent of the choice of T and t_i, by change of variables in equation (6).

We see that the signature is a collection of integrals, with each integral defining a real value. It is a graded sequence of statistics that characterise the input time series. In particular, (Hambly & Lyons, 2010) show that under mild conditions, $\mathrm{Sig}^\infty(X)$ completely determines X up to translation, provided time is included as a channel in X.

Log-signature transform The signature transform has some redundancy: a little algebra shows that for example $S^{1,2}_{a,b}(X) + S^{2,1}_{a,b}(X) = S^1_{a,b}(X)S^2_{a,b}(X)$, so that we already know $S^{2,1}_{a,b}(X)$ provided we know the other three quantities.

The *log-signature transform* is then essentially obtained by computing the signature transform, and throwing out redundant terms, to obtain some (nonunique) minimal collection.

Starting from the depth-N signature transform and removing some fixed set of redundancies produces the *depth-N log-signature transform*. We fix some set of redundancies throughout (essentially corresponding to a choice of basis), and denote this $\mathrm{LogSig}^N_{a,b}$. This is a map from Lipschitz continuous paths $[a, b] \to \mathbb{R}^v$ into $\mathbb{R}^{\beta(v,N)}$, where $\beta(v, N)$ denotes the dimension of the log-signature (see Appendix A).

Geometric intuition In figure 2 we provide a geometric intuition for the first two levels of the log-signature, which have natural geometric interpretations.

The depth 1 terms correspond to the changes in each channel over the interval; this is $\Delta X_1, \Delta X_2$ in the figure. The depth 2 term corresponds to the signed area in between the chord joining the endpoints and the path itself; this corresponds to $A_+ - A_-$ in the figure. Higher order terms correspond to higher order integrals and iterated areas in higher dimensional spaces, and become a little more difficult to visualise.

(Log-)Signatures and CDEs In Figure 3 we give the equations for how log-signatures arise in the solution of CDEs. Begin by letting D_f denote the Jacobian of a function f. Now expand equation (1) by linearising the vector field f and neglecting higher order terms.

[1] For our purposes later it will typically be a linear interpolation of a time series.

[2] This is a slightly simplified definition, and the signature is often instead defined using the notation of stochastic calculus; for completeness see Definition A.2.

$$
\begin{aligned}
Z_t &\approx Z_a + \int_a^t \Big(f(Z_a) + D_f(Z_a)(Z_s - Z_a)\Big)\frac{\mathrm{d}X}{\mathrm{d}t}(s)\,\mathrm{d}s \\
&= Z_a + \int_a^t f(Z_a)\frac{\mathrm{d}X}{\mathrm{d}t}(s)\,\mathrm{d}s + \int_a^t \left(D_f(Z_a)\int_a^s f(Z_u)\,\frac{\mathrm{d}X}{\mathrm{d}t}(u)\,\mathrm{d}u\right)\frac{\mathrm{d}X}{\mathrm{d}t}(s)\,\mathrm{d}s \\
&\approx Z_a + f(Z_a)\int_a^t \frac{\mathrm{d}X}{\mathrm{d}t}(s)\,\mathrm{d}s + D_f(Z_a)f(Z_a)\int_a^t\int_a^s \frac{\mathrm{d}X}{\mathrm{d}t}(u)\,\mathrm{d}u\frac{\mathrm{d}X}{\mathrm{d}t}(s)\mathrm{d}s \\
&= Z_a + f(Z_a)\big\{S(X)^{(i)}\big\}_{i=1}^d + D_f(Z_a)f(Z_a)\big\{S(X)^{(i,j)}\big\}_{i,j=1}^d.
\end{aligned}
$$

Figure 3. Signature (Taylor) expansion of a CDE. The action of the vector field f on the depth-N signature is a matrix-vector product and is fully described, for any N, in (Boutaib et al., 2014).

This is simply the Taylor Expansion of the CDE. The Taylor coefficients are precisely these signature terms, thus demonstrating how signatures are intrinsically linked to the solutions of CDEs. Higher order Taylor expansions results in corrections using higher order signature terms.

2.2. The Log-ODE Method

Recall for $X\colon [a,b] \to \mathbb{R}^v$ that $\mathrm{LogSig}^N_{a,b}(X) \in \mathbb{R}^{\beta(v,N)}$. The log-ODE method states that $Z_b \approx \widehat{Z}_b$ where

$$
\widehat{Z}_u = \widehat{Z}_a + \int_a^u \widehat{f}(\widehat{Z}_s)\frac{\mathrm{LogSig}^N_{a,b}(X)}{b-a}\mathrm{d}s \text{ for } u \in (a,b], \tag{8}
$$

and $\widehat{Z}_a = Z_a$. Here Z is the same as in equation (3), and the relationship between $\widehat{f}$ to f is given in Appendix A.

That is, the solution of the CDE may be approximated by the solution to an ODE. This is typically applied locally: pick some points r_i such that $a = r_0 < r_1 < \cdots < r_m = b$, split up the CDE of equation (1) into an integral over $[r_0, r_1]$, an integral over $[r_1, r_2]$, and so on, and apply the log-ODE method to each interval separately. A CDE treated in this way is, for the purposes of this exposition, termed a *rough differential equation*.

See Appendix A for the precise details and Appendix B for a proof of convergence. For the reader familiar with the Magnus expansion for linear differential equations (Blanes et al., 2009), then the log-ODE method is a generalisation.

3. Method

We move on to introducing the neural rough differential equation.

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. The choice and spacing of r_i will be a hyperparameter.

We also pick a depth hyperparameter $N \geq 1$. In section 2 we introduced the depth-N log-signature transform. For $X\colon [t_0, t_n] \to \mathbb{R}^v$ and $t_0 \leq r_i < r_{i+1} \leq t_n$ the log-signature of X over the interval $[r_i, r_{i+1}]$ was defined to be a particular collection of statistics $\mathrm{LogSig}^N_{r_i,r_{i+1}}(X) \in \mathbb{R}^{\beta(v,N)}$; specifically those statistics that best describe how X drives the CDE equation (1).

3.1. The Rough Hidden State Update

Recall how the Neural CDE formulation of equation (3) was solved via equations (4), (5). For the rough approach we begin by replacing (5) with the piecewise

$$
\widehat{g}_{\theta,X}(Z,s) = \widehat{f}_\theta(Z)\frac{\mathrm{LogSig}^N_{r_i,r_{i+1}}(X)}{r_{i+1}-r_i} \quad \text{for } s \in [r_i, r_{i+1}), \tag{9}
$$

where $\widehat{f}_\theta\colon \mathbb{R}^w \to \mathbb{R}^{w\times\beta(v,N)}$ is an arbitrary neural network, and the right hand side denotes a matrix-vector product between $\widehat{f}_\theta$ and the log-signature. Equation (4) then becomes

$$
Z_t = Z_{t_0} + \int_{t_0}^t \widehat{g}_{\theta,X}(Z_s, s)\mathrm{d}s. \tag{10}
$$

This may now be solved as a (neural) ODE using standard ODE solvers.

We give an overview of this process in figure 4. The left hand side represents a single step method, as in the existing approach to Neural CDEs. The right hand side depicts a rough approach that takes steps larger than the discretisation of the data in exchange for additional terms of the log-signature.

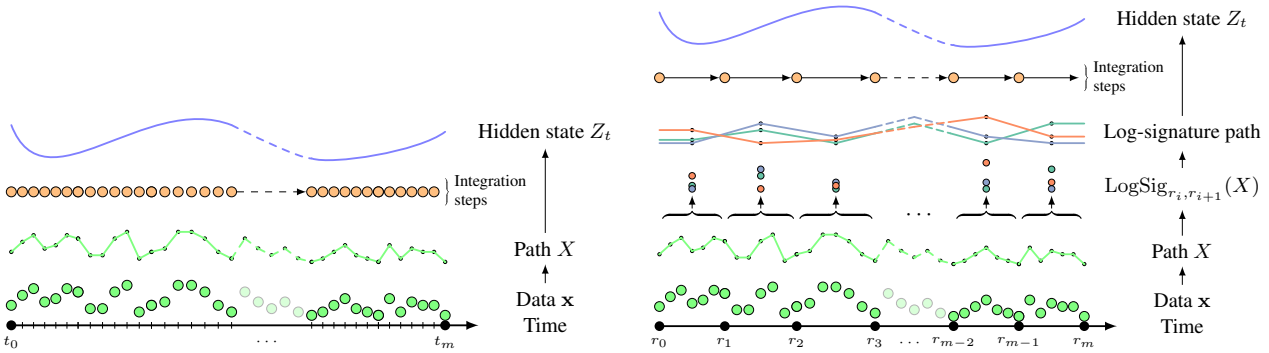

Figure 4. An overview of the log-ODE method applied to Neural RDEs. **Left:** A single step (CDE or RDE) model. The path X is quickly varying, meaning a lot of integration steps are needed to resolve it. **Right:** The Neural RDE utilising the log-ODE method with integration steps larger than the discretisation of the data. The path of log-signatures is more slowly varying (in a higher dimensional space), and needs fewer integration steps to resolve.

3.2. Neural RDEs Generalise Neural CDEs

Suppose we happened to choose $r_i = t_i$ and $r_{i+1} = t_{i+1}$. Then the log-signature term is

$$\frac{\mathrm{LogSig}^N_{t_i,t_{i+1}}(X)}{t_{i+1} - t_i}$$

Recall that the depth 1 log-signature is just the increment of the path over the interval. So this becomes

$$\frac{\Delta X_{[t_i,t_{i+1}]}}{t_{i+1} - t_i} = \frac{\mathrm{d}X^{\mathrm{linear}}}{\mathrm{d}t}(s) \quad \text{for } s \in [t_i, t_{i+1}),$$

that is to say the same as obtained via the original method if using linear interpolation. In this way the Neural RDE approach generalises the existing Neural CDE approach.

3.3. Discussion

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 10.

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. If the vector field f_θ requires $\mathcal{O}(H)$ memory, and the time series is of total length T, then backpropagating through the solver requires $\mathcal{O}(HT)$ memory whilst the adjoint method requires only $\mathcal{O}(H + T)$; see Kidger et al. (2020).

The Log-signature as a Preprocessing Step When training a model in practice, the log-signatures need only be computed once and thus the computation can be performed as part of data preprocessing. Log-signatures can also be easily computed in an online fashion, making the model suitable for such problems.

Structure of $\widehat{f}$ The description here aligns with the log-ODE scheme described in equation (8). There is one discrepancy: we do not attempt to model the specific structure of $\widehat{f}$. This is in principle possible, but is computationally expensive. Instead, we model $\widehat{f}$ as a neural network directly. This need *not* necessarily exhibit the requisite structure, but as neural networks are universal approximators (Pinkus, 1999; Kidger & Lyons, 2020a) then this approach is at least as general from a modelling perspective.

Ease of Implementation This method is straightforward to implement using pre-existing tools.

There are standard libraries available for computing the log-signature transform: we use Signatory (Kidger & Lyons, 2020b). As equation (10) is an ODE, it may be solved directly using tools such as `torchdiffeq` (Chen, 2018).

As an alternative, we note that the form of equation (9) is that of equation (5), with the driving path taken to be piecewise linear in log-signature space. Computation of the log-signatures can therefore be considered as a preprocessing step, producing a sequence of log-signatures. From this we may construct a path in log-signature space, and apply existing tools for neural CDEs. (Rather than tools for neural ODEs.) This idea is summarised in figure 4. We make this approach available in the [redacted] open source project.

Applications In principle, a Neural RDE may be applied to solve any Neural CDE problem. However, we typically

observe limited benefit on relatively short time series: the original Neural CDE formulation works well enough, and there is little room to see either speed or loss/accuracy improvements via this approach.

The situation changes for long time series. Here, the existing approach struggles as the length of the time series grows. Performance worsens, and speed drops due to the sheer number of forward evaluations. This is the same behaviour as for RNNs.

Now, the reduction in length (from n to $m \ll n$) is highly beneficial. Moreover, the compression performed by the log-signature is also of benefit: closely-sampled points will be typically be strongly correlated, and there is little to be gained by treating them all individually.

In addition, there are two advantages shared by both Neural CDEs and Neural RDEs, that make them suitable for long time series. The first is the sharply reduced memory requirements of the adjoint method. For example (chosen arbitrarily without cherry-picking) in one experiment we see a reduction in memory usage from 3.6GB to just 47MB.

The second is that as both operate in continuous time, the steps in the numerical solver may be decoupled from the sampling rate of the data: steps are taken with respect to the complexity of the data, not just its sampling rate. In particular a slowly-varying but densely-sampled path would still be fast without requiring many integration steps.

The Depth and Step Hyperparameters To solve a Neural RDE accurately via the log-ODE method, we should be prepared to take the depth N suitably large, or the intervals $r_{i+1} - r_i$ suitably small. Accomplishing this would often require that they are taken relatively large or relatively small, respectively. Instead, we treat these as hyperparameters. This makes use of the log-ODE method a modelling choice rather than an implementation detail.

Increasing step size will lead to faster (but less informative) training by reducing the number of operations in the forward pass. Increasing depth will lead to slower (but more informative) training, as more information about each local interval is used in each update.

4. Experiments

We run experiments applying Neural RDEs to four real-world datasets. Every problem was chosen for its long length. The lengths are sufficiently long that adjoint-based backpropagation (Chen et al., 2018) was often needed simply to avoid running out of memory at any reasonable batch size. Every problem is regularly sampled, so we take $t_i = i$.

Recall that the Neural RDE approach features two hyperparameters, corresponding to log-signature depth and step size. Good choices will turn out to have a dramatic positive effect on performance. Accordingly for every experiment we run Neural RDEs for all depths in $N = 2, 3$ and all step sizes in $2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$. Depth 1 and step 1 are not considered as both reduce onto the Neural CDE model, as discussed in section 3.2. In practice, when choosing a final model, one would choose that with depth and step values that minimise the validation loss, as in any hyperparamter value selection.

We compare against two baseline models. The first is a Neural CDE; as the model we are extending then comparisons to this are our primary concern. For context we also additionally include a baseline against the ODE-RNN introduced in Rubanova et al. (2019). For both of these models, we also run experiments on the full range of step sizes described above.

For the Neural CDE model, increased step sizes correspond to naïve subsampling of the data (in accordance with section 3.2). For the ODE-RNN model, we instead fold the time dimension into the feature dimension, so that at each step the ODE-RNN model sees several adjacent time points. This represents an alternate technique for dealing with long time series, so as to provide a reasonable benchmark.

For each model, and each hyperparameter combination, we run the experiment three times and report the mean and standard deviation of the test metrics. We additionally report mean training times and memory usages.

Precise details of hyperparameter selection, optimisers, normalisation, and so on can be found in Appendix C. For brevity, we provide results for only some of the step sizes here. The full results are described in Appendix D.

4.1. Classifying EigenWorms

Our first example uses the EigenWorms dataset from the UEA archive from Bagnall et al. (2017). This consists of time series of length 17 984 and 6 channels (including time), 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.

Results are shown in Table 1. We begin by seeing that the step-1 Neural CDE model takes roughly a day to train. Switching to Neural RDEs speeds this up by an order of magnitude, to roughly two hours. Moreover doing so dramatically improves accuracy, by up to 17%, reflecting the classical difficulty of learning from long time series.

Meanwhile naïve subsampling approaches for the Neural CDE method only achieve speed-ups without performance improvements. The folded ODE-RNN model performs poorly, attaining the worst score for any step size whilst imposing a significantly higher memory burden.

Model	Step	Accuracy (%)	Time (Hrs)	Mem (Mb)
ODE-RNN (folded)	1	–	–	–
	4	35.0 ± 1.5	0.8	3629.3
	32	32.5 ± 1.5	0.1	532.2
	128	47.9 ± 5.3	0.0	200.8
NCDE	1	62.4 ± 12.1	22.0	176.5
	4	66.7 ± 11.8	5.5	46.6
	32	64.1 ± 14.3	0.5	8.0
	128	48.7 ± 2.6	0.1	3.9
NRDE (depth 2)	4	**83.8 ± 3.0***	2.4	180.0
	32	67.5 ± 12.1	0.7	28.1
	128	**76.1 ± 5.9**	0.2	7.8
NRDE (depth 3)	4	76.9 ± 9.2	2.8	856.8
	32	**75.2 ± 3.0**	0.6	134.7
	128	68.4 ± 8.2	0.1	53.3

Table 1. EigenWorms dataset: mean ± standard deviation of test set accuracy measured over three repeats. Also reported are the mean memory usage and training time. For all models a variety of step sizes are considered. For the Neural RDE we additionally investigate varying depths. (Recalling that the NCDE is a depth-1 NRDE.) '–' denotes that the model could not be run within GPU memory. Bold denotes the best model score for a given step size, and * denotes that the score was the best achieved over all models and step sizes.

Results across all step sizes may be found in Appendix D.

4.2. Estimating Vitals Signs from PPG and ECG data

Next we consider three separate problems, using data from the TSR archive (Tan et al., 2020), coming originally from the Beth Israel Deaconess Medical Centre (BIDMC).

We aim to predict a person's respiratory rate (RR), their heart rate (HR), or their oxygen saturation (SpO2) at the end of the sample, having observed PPG and ECG data over the length of the sample. The data is sampled at 125Hz and each series has length 4 000. There are 3 channels (including time). We evaluate performance with the L^2 loss.

The results are shown in table 2.

We find that the depth 3 Neural RDE is the top performer for every task at every step size, reducing test loss by 30–59% versus the Neural CDE. Moreover, it does so with roughly an order of magnitude less training time.

We attribute the improved test loss to the Neural RDE model being better able to learn long-term dependencies due to the reduced sequence length. Note that the performance of the rough models actually improves as the step size is increased. This is in contrast to Neural CDE, which sees a degradation in performance.

The ODE-RNN model, besides using significantly more memory, struggles to train effectively when the sequence length is long. Training improves as the sequence size is shortened, but still produces results substantially worse than those achieved by the Neural RDE.

As a visual summary of these results, including the full range of step sizes, we also provide heatmaps in Figure 5.

The full results across the full range of step sizes may be found in Appendix D.

5. Limitations

Number of hyperparameters Two new hyperparameters – truncation depth and step size – with substantial effects on training time and memory usage must now also be tuned.

Number of input channels The log-ODE method is most feasible with few input channels, as the number of log-signature channels $\beta(v, N)$ grows exponentially in v. For larger v then the available parallelism may become saturated.

6. Related Work

CNNs and Transformers have been shown to offer improvements over RNNs for modelling long-term dependencies (Bai et al., 2018; Li et al., 2019), although the latter in particular have typically focused on language modelling. On a more practical note, Transformers are famously $\mathcal{O}(L^2)$ in the length of the time series L. Several approaches have been introduced to reduce this, for example Li et al. (2019) reduce this to $\mathcal{O}(L(\log L)^2)$. Extensions specifically to long sequences do exist (Sourkov, 2018), but again these typically focus on language modelling rather than multivariate time series data.

There has also been some work on long time series for classic RNN (GRU/LSTM) models.

Wisdom et al. (2016); Jing et al. (2019) show that unitary or orthogonal RNNs can mitigate the vanishing/exploding gradients problem. However, they are expensive to train due to the need to compute a matrix inversion at each training step. Chang et al. (2017) introduce dilated RNNs with skip connections between RNN states, which help improve training speed and learning of long-term dependencies. Campos et al. (2017) introduce the 'Skip-RNN' model, which extend the RNN by adding an additional learnt component that skips state updates. Li et al. (2018) introduce the 'IndRNN' model, with particular structure tailored to learning long time series.

One meaningful comparison is to hierarchical subsampling as in Graves (2012); De Mulder et al. (2015). There the data is split into windows, an RNN is run over each window, and then an additional RNN is run over the first RNN's

Model	Step	L^2			Time (Hrs)			Memory (Mb)
		RR	HR	SpO_2	RR	HR	SpO_2	
ODE-RNN (folded)	1	–	13.06 ± 0.0	–	–	10.5	–	3653.0
	8	2.47 ± 0.35	13.06 ± 0.00	3.3 ± 0.00	1.5	1.2	0.9	917.2
	128	1.62 ± 0.07	13.06 ± 0.00	3.3 ± 0.00	0.2	0.1	0.1	81.9
	512	1.66 ± 0.06	6.75 ± 0.9	1.98 ± 0.31	0.0	0.1	0.1	40.4
NCDE	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
	128	2.64 ± 0.18	11.98 ± 0.37	2.86 ± 0.04	0.2	0.2	0.3	8.7
	512	2.53 ± 0.03	12.22 ± 0.11	2.98 ± 0.04	0.1	0.0	0.1	8.4
NRDE (depth 2)	8	2.63 ± 0.12	8.63 ± 0.24	2.88 ± 0.15	2.1	3.4	3.3	21.8
	128	1.86 ± 0.03	6.77 ± 0.42	1.95 ± 0.18	0.3	0.4	0.7	10.9
	512	1.81 ± 0.02	5.05 ± 0.23	2.17 ± 0.18	0.1	0.2	0.4	10.3
NRDE (depth 3)	8	$\mathbf{2.42 \pm 0.19}$	$\mathbf{7.67 \pm 0.40}$	$\mathbf{2.55 \pm 0.13}$	2.9	3.2	3.1	43.3
	128	$\mathbf{1.51 \pm 0.08}$	$\mathbf{2.97 \pm 0.45}^*$	$\mathbf{1.37 \pm 0.22}$	0.5	1.7	1.7	17.3
	512	$\mathbf{1.49 \pm 0.08}^*$	$\mathbf{3.46 \pm 0.13}$	$\mathbf{1.29 \pm 0.15}^*$	0.3	0.4	0.4	15.4

Table 2. The three experiments on BIDMC datasets: mean $\pm$ standard deviation of test set L^2 loss, measured over three repeats, over each of three different vital signs prediction tasks (RR, HR, SpO_2). Also reported are the memory usage and training time. Only mean times are shown for space. For all models a variety of step sizes are considered. For the Neural RDE we additionally investigate varying depths. (Recalling that the NCDE is a depth-1 NRDE.) '–' denotes that the model could not be run within GPU memory. Bold denotes the best model score for a given step size, and * denotes that the score was the best achieved over all models and step sizes.

Figure 5. Heatmap depicting normalised losses on the three BIDMC datasets for differing step sizes and depths. We can see that the point of lowest MSE (deepest red) has step > 1 and depth > 1, and that performance worsens for very long steps. This represents the depth/step tradeoff for long length time series.

outputs; we may describe this as an RNN/RNN pair. Liao et al. (2019) then perform the equivalent operation with a log-signature/RNN pair. In this context, our use of log-ODE method is analogous to an log-signature/NCDE pair.

In comparison to Liao et al. (2019), this means moving from an inspired choice of pre-processing to an actual implementation of the log-ODE method. In doing so the differential equation structure is preserved. Moreover this takes advantage of the synergy between log-signatures (which extract statistics on how data drives differential equations), and the controlled differential equation it then drives. Broadly speaking these connections are natural: at least within the signature/CDE/rough path community, it is a well-known but poorly-published fact that RNNs, (log-)signatures, and (Neural) CDEs are all related; see for example Kidger et al. (2020) for a little exposition on this.

De Brouwer et al. (2019); Lechner & Hasani (2020) amongst others consider continuous time modifications to GRUs and LSTMs, improving the learning of long-term dependencies. Voelker et al. (2019); Gu et al. (2020) consider links with ODEs and approximation theory, with the goal of improving the long-term memory capacity of RNNs. Given the differential equation structure both they and we consider, a hybridisation of these techniques seems like a promising line of future inquiry.

7. Conclusion

We have introduced *neural rough differential equations* as an approach to continuous-time time series modelling. These extend Neural CDEs, driving the hidden state not by point evaluations but by interval summarisations of the underlying time series or control path. Neural RDEs may still be solved via ODE methods, and thus retain both adjoint backpropagation and continuous dynamics. As they additionally reduce the effective length of the control path, we observe substantial practical benefits in applying Neural RDEs to long time series. In this regime we report significant training speed-ups, model performance improvements, and reduced memory requirements, on problems of length up to 17 000.

Acknowledgements

We would like to thank Terry Lyons as the main supervisor of this project. His name has not appeared on the final version of this paper only due to an administrative error on our part.

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 were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

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

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Blanes, S., Casas, F., Oteo, J.-A., and Ros, J. The Magnus expansion and some of its applications. *Physics Reports*, 470(5-6):151–238, 2009.

Boutaib, Y., Gyurkó, L. G., Lyons, T., and Yang, D. Dimension-free Euler estimates of rough differential equations. *Revue Roumaine de Mathmatiques Pures et Appliques*, 59, 2014.

Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., and Chang, S.-F. Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks. *arXiv preprint arXiv:1708.06834*, 2017.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Chen, R. T. Q. `torchdiffeq`, 2018. https://github.com/rtqichen/torchdiffeq.

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

De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pp. 7379–7390, 2019.

De Mulder, W., Bethard, S., and Moens, M.-F. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30 (1):61–98, 2015.

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

Friz, P. K. and Victoir, N. B. *Multidimensional stochastic processes as rough paths: theory and applications*, volume 120. Cambridge University Press, 2010.

Graves, A. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pp. 5–13. Springer, 2012.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. HiPPO: Recurrent Memory with Optimal Polynomial Projections. *arXiv:2008.07669*, 2020.

Gyurkó, L. G. *Numerical methods for approximating solutions to rough differential equations*. DPhil thesis, University of Oxford, 2008.

Hambly, B. and Lyons, T. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, 171, 2010.

Jing, L., Gulcehre, C., Peurifoy, J., Shen, Y., Tegmark, M., Soljacic, M., and Bengio, Y. Gated orthogonal recurrent units: On learning to forget. *Neural computation*, 31(4): 765–783, 2019.

Kidger, P. and Lyons, T. Universal Approximation with Deep Narrow Networks. *COLT 2020*, 2020a.

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

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

Lechner, M. and Hasani, R. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466, 2018.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. Enhancing the locality and breaking the

memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, pp. 5243–5253, 2019.

Liao, S., Lyons, T., Yang, W., and Ni, H. Learning stochastic differential equations using RNN with log signature features. *arXiv preprint arXiv:1908.08286*, 2019.

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

Lyons, T., Michael, C., and Thierry, L. *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.

Lyons, T. J. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14(2):215–310, 1998.

Lyons, T. J., Caruana, M., Lévy, T., and Picard, J. Differential equations driven by rough paths. *Ecole d'été de Probabilités de Saint-Flour*, 34:1–93, 2004.

Pinkus, A. Approximation theory of the MLP model in neural networks. *Acta Numer.*, 8:143–195, 1999.

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

Rubanova, Y., Chen, R. T., and Duvenaud, D. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

Ryan, R. A. *Introduction to Tensor Products of Banach Spaces*. Springer Monographs in Mathematics, Springer, 2002.

Sourkov, V. Igloo: Slicing the features space to represent sequences. *arXiv preprint arXiv:1807.03402*, 2018.

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

Voelker, A., Kajić, I., and Eliasmith, C. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. Full-capacity unitary recurrent neural networks. In *Advances in neural information processing systems*, pp. 4880–4888, 2016.

Wong, E. and Zakai, M. On the Convergence of Ordinary Integrals to Stochastic Integrals. *Annals of Mathematical Statistics*, 36(5):1560–1564, 1965.