NEURAL CDES FOR LONG TIME SERIES VIA THE LOG-ODE METHOD

Anonymous authors
Paper under double-blind review

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 is difficult for long time series. Training takes impractically long, and models may fail to train. Here, we demonstrate that an existing numerical method for the solution of CDEs – the log-ODE method – may in the context of Neural CDEs be used to take integration steps *larger* than the discretisation of the data, whilst depending upon sub-step data through additional terms. Doing so represents making a length/channel trade-off, and is easy to implement with existing tools. We demonstrate efficacy on problems of length up to 17k observations and observe training speed-ups from roughly days to roughly minutes.

1 INTRODUCTION

Neural controlled differential equations (Neural CDEs) (**?**) are the continuous-time analogue to a recurrent neural network (RNN), and provide a natural method for modelling temporal dynamics with neural networks.

Neural CDEs are similar to neural ordinary differential equations (Neural ODEs), as popularised by **?**. 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 changes in the data provoke in the local dynamics of the system.

1.1 CONTROLLED DIFFERENTIAL EQUATIONS

We begin by recalling the definition of a CDE.

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

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

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

where the integral is a Riemann-Stieltjes integral. The notation "$f(Z_s)\mathrm{d}X_s$" denotes a matrix-vector product, and if X is differentiable then $\int_a^t f(Z_s)\mathrm{d}X_s = \int_a^t f(Z_s)\frac{\mathrm{d}X}{\mathrm{d}s}(s)\mathrm{d}s$.

If in equation (**??**), $\mathrm{d}X_s$ 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 changes in X.

1.2 NEURAL CONTROLLED DIFFERENTIAL EQUATIONS

We recall the definition of a Neural CDE as in **?**.

Define 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 $X_{t_i} = (t_i, x_i)$. **?** use natural cubic splines. Here we will actually end up finding piecewise linear interpolation to be a more convenient choice. (We avoid issues with adaptive solvers as discussed in **?**, Appendix A simply by using fixed solvers.)

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. Let $\ell_\theta\colon \mathbb{R}^w \to \mathbb{R}^q$ be linear, for some 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 controlled differential equation* driven by X if

$$Z_{t_0} = \xi_\theta(t_0, x_0), \quad \text{with} \quad Z_t = Z_{t_0} + \int_{t_0}^{t} f_\theta(Z_s)\mathrm{d}X_s \quad \text{and} \quad Y_t = \ell_\theta(Z_t) \quad \text{for } t \in (t_0, t_n]. \tag{2}$$

That is – just like an RNN - we have evolving hidden state Z, which we take a linear map from to produce an output. This formulation is a universal approximator (**?**, 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.

The question remains how to compute the integral of equation (**??**). **?** let

$$g_{\theta,X}(Z, s) = f_\theta(Z)\frac{\mathrm{d}X}{\mathrm{d}s}(s), \tag{3}$$

where the right hand side denotes a matrix multiplication, and then note that the integral can be written as

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

This reduces the CDE to an ODE, so that existing tools for Neural ODEs may be used to evaluate this, and to backpropagate.

1.3 Applications

By moving from the discrete-time formulation of an RNN to the continuous-time formulation of a Neural CDE, then all kinds of time series data are put on the same footing, whether it is regular or irregularly sampled, and whether or not it has missing values. The interpolation procedure for X works regardless. Informative missingness is not sacrificed and can be incorporated by appending extra channels as usual.

Besides this, the continuous-time or differential equation formulation may be useful in applications where such models are explicitly desired, as when modelling physics.

1.4 Contributions

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

Here, we propose to use the "log-ODE method", which is a numerical method for the solution of CDEs. In the CDE literature this is used as a way to handle roughness in X. Here, however, it instead becomes a principled way to trade length for channels. Loosely speaking, long time series are reduced to higher-dimensional short time series, and the problems attendant with length are alleviated. Training times may be improved from roughly days to roughly minutes, model performance is improved, and memory requirements are reduced.

The resulting scheme has two very neat interpretations. In terms of numerical differential equation solvers, this corresponds to taking integration steps larger than the discretisation the data, whilst incorporating sub-step information through additional terms.[1] In terms of machine learning, this corresponds to binning the data prior to running a Neural CDE, with bin statistics carefully chosen to extract precisely the information most relevant to solving a CDE. The method may be implemented as a relatively straightforward data preprocessing step.

[1]For the reader familiar with numerical methods for SDEs, this is akin to the additional correction term in Milstein's method as compared to Euler–Maruyama.

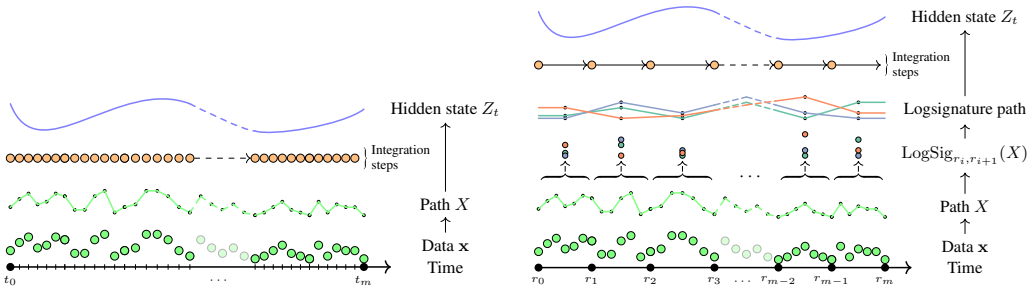


Figure 1: **Left:** The original Neural CDE formulation. The path X_t is quickly varying, meaning a lot of integration steps are needed to resolve it. **Right:** The log-ODE method. The logsignature path is more slowly varying (in a higher dimensional space), and needs fewer integration steps to resolve.

2 THEORY

Here we discuss the motivating theory. Readers more interested in the practical application of the method, and not necessarily the theoretical motivations, may wish to skip to the next section. See Figure **??** for the essential idea.

2.1 SIGNATURES AND LOGSIGNATURES

We begin with an overview of the signature and logsignature transforms.

Signature transform Consider a path $X = (X^1, \ldots, X^v) \colon [a, b] \to \mathbb{R}^v$. We define its *signature transform* as the infinite collection of statistics

$$S_{a,b}(X) = \left(\left\{ S_{a,b}^{i_1}(X) \right\}_{1 \le i_1 \le v}, \left\{ S_{a,b}^{i_1,i_2}(X) \right\}_{1 \le i_1, i_2 \le v}, \left\{ S_{a,b}^{i_1,i_2,i_3}(X) \right\}_{1 \le i_1, i_2, i_3 \le v}, \ldots \right), \tag{5}$$

where each term is a n-fold iterated integral of $\mathbf{x}$ with multi-index $i_1, ..., i_d$:

$$S_{a,b}^{i_1,\ldots,i_d}(X) := \int_a^b \int_a^{t_{d-1}} \cdots \int_a^{t_2} \mathrm{d}X_{t_1}^{i_1} \cdots \mathrm{d}X_{t_d}^{i_d}. \tag{6}$$

This produces a graded sequence of statistics associated with a path. **?** show that these statistics in fact characterise the path up "tree-like equivalence".

As S is an infinite sequence of values, we also define the *depth-N signature transform* as

$$S_{a,b}^N(X) = \left(\left\{ S_{a,b}^{i_1}(X) \right\}_{1 \le i_1 \le v}, \ldots, \left\{ S_{a,b}^{i_1,\ldots,i_N}(X) \right\}_{1 \le i_1, \ldots, i_N \le v} \right). \tag{7}$$

See **?** or **?**, Appendix A.

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

The *logsignature transform* formalises this by implementing a procedure involving throwing out all such redundant terms, and keeping only some minimal collection of terms. Such a minimal collection is nonunique, as we could have thrown out $S_{a,b}^{1,2}(X)$ instead of $S_{a,b}^{2,1}(X)$, for example. We are deliberately light on the (rather involved) details - see Appendix **??**, and **??**.

Starting from the depth-N signature transform and performing this procedure produces the *depth-N logsignature transform*.[2] We denote this $\mathrm{LogSig}_{a,b}^N$, which is a map from Hölder continuous paths

[2]Note that similar terminology such as "step-N logsignature" is also used in the literature.

$[a, b] \to \mathbb{R}^v$ into $\mathbb{R}^{\beta(v,N)}$, where

$$\beta(d, N) = \sum_{k=1}^{N} \frac{1}{k} \sum_{i|k} \mu\left(\frac{k}{i}\right) d^i$$

with μ the Möbius function.

2.2 THE LOG-ODE METHOD

Recall the CDE of equation (**??**), that is: $Z_a = \xi$, and $Z_t = Z_a + \int_a^t f(Z_s)\mathrm{d}X_s$ for $t \in (a, b]$.

Let $N \in \mathbb{N}$ be a fixed depth. Decompose $f \colon \mathbb{R}^w \to \mathbb{R}^{w \times v}$ into vector fields $f_i \colon \mathbb{R}^w \to \mathbb{R}^w$ for $i \in \{1, \ldots, v\}$, and then let $\widehat{f} = (f_{i1 \leq i \leq v}, [f_i, f_j]_{1 \leq i < j \leq d}, \ldots)$ be the collection of all (nested up depth N) Lie brackets whose foliage[3] is a Lyndon word, so that overall $\widehat{f} \colon \mathbb{R}^w \to \mathbb{R}^{w \times \beta(d,N)}$.

Recall for $X \colon [a, b] \to \mathbb{R}^v$ that $\mathrm{LogSig}_{a,b}^N(X) \in \mathbb{R}^{\beta(d,N)}$. Then the log-ODE method states

$$Z_b \approx \widehat{Z}_b \quad \text{where} \quad \widehat{Z}_u = \widehat{Z}_a + \int_a^u \widehat{f}(\widehat{Z}_s)\mathrm{LogSig}_{a,b}^N(X)\mathrm{d}s, \quad \text{and} \quad \widehat{Z}_a = Z_a. \tag{8}$$

The approximation becomes arbitrarily good[4] as $N \to \infty$.

That is, the solution of the CDE may be approximated by the solution to an ODE. In practice, we go further and pick some points r_i such that $a = r_0 < r_1 < \cdots < r_m = b$. We split up the CDE of equation (**??**) 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.

See **??** for other overviews of the log-ODE method. Further information on its theoretical properties and convergence guarantees can be found in **?**. See **???** for applications of the log-ODE method to stochastic differential equations (SDEs).

The log-ODE method is a generalisation of the Magnus expansion for linear differential equations (see **?**). There, the control path is obtained by integrating the time-varying vector field and the log-ODE solve reduces to exponentiating a matrix.

2.3 NEURAL CDES VIA THE LOG-ODE METHOD

We apply two relaxations to make the log-ODE scheme more practical.

Lie bracket structure First, we do not attempt to compute the Lie bracket structure of $\widehat{f}$. Whilst in principle this is possible (even straightforward with autodifferentiation), it is quite computationally expensive. Instead, a cheaper approach is to model $\widehat{f}$ as a neural network directly. This need *not* necessarily exhibit a Lie bracket structure, but as neural networks are universal approximators (**??**) then this approach is at least as general from a modelling perspective.

Hyperparameters Second, we do not attempt to take $r_{i+1} - r_i$ small enough or N large enough such that the error compared to the original Neural CDE is made small. Instead we treat these as hyperparameters, and regard the use of the log-ODE method as a modelling choice rather than an implementation detail. This still produces good results, whilst being cheaper to compute.

Overall Recall how the Neural CDE formulation of equation (**??**) was solved via equations (**??**), (**??**).

Our method is thus to replace equations (**??**), (**??**) with

$$Z_t = Z_{t_0} + \int_{t_0}^{t} \widehat{g}_{\theta,X}(Z_s, s)\mathrm{d}s, \tag{9}$$

[3]The map produced by "ignoring the Lie brackets", for example $[f_1, [[f_2, f_1], f_3]] \mapsto (1, 2, 1, 3)$.

[4]With respect to any norm as $Z_b, \widehat{Z}_1$ exist in finite dimensions.

where

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

is piecewise in s, and $\widehat{f}_\theta \colon \mathbb{R}^w \to \mathbb{R}^{w \times \beta(v,N)}$ is an arbitrary neural network.

Importantly, for implementation purposes: this is actually the same formulation as equation (**??**), with the driving path taken to be piecewise linear in logsignature space. (So that its derivative is piecewise constant.) This is what allows us to make the relatively simple presentation in the next section.

As this formulation is now detached from the original sampling rate of the data, and instead uses the much coarser sampling points r_i, then we may expect to use larger integration steps. The information introduced in $\mathrm{LogSig}^N_{r_i,r_{i+1}}(X)$ for $N > 1$ are higher-order correction terms that compensate for the larger steps.

3 COMPUTATION

We move on to discussing how the method may be implemented, which is straightforward with existing tools. See Figure **??** for the essential idea.

3.1 METHOD

Data 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 pick some 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.

Logsignature transform The depth-N logsignature transform, denoted $\mathrm{LogSig}^N_{a,b}$, is a map from continuous paths $[a, b] \to \mathbb{R}^v$ into $\mathbb{R}^{\beta(v,N)}$. See the previous section for a discussion on the precise nature of the logsignature transform and an equation for β.

We apply the logsignature transform to X on each $[r_i, r_{i+1}]$ for all i. This produces a sequence $(L_1, \ldots, L_m)$ such that

$$L_i = \mathrm{LogSig}^N_{r_{i-1},r_i}(X) \in \mathbb{R}^{\beta(v,N)}.$$

Additionally, let L_0 be the logsignature of the straight-line path from zero to x_0. (Equivalently: just define L_i as above given the input time series $((t_0 - 1, 0), (t_0, x_0), (t_1, x_1), ..., (t_n, x_n))$.)

There are standard libraries available which make computing the logsignature transform easy. We use Signatory (**?**), which is PyTorch-compatible (**?**). Note that efficient algorithms are only known for computing the logsignature of a piecewise linear path; this is what motivates the use of piecewise linear interpolation to construct X.

Neural CDEs Finally, we consider the input time series

$$((r_0, L_0), (r_1, L_0 + L_1), \ldots, (r_{m-1}, \sum_{j=0}^{m-1} L_j), (r_m, \sum_{j=0}^{m} L_j)) \tag{10}$$

and perform the usual Neural CDE procedure. That is, interpolate this data, use it to drive a CDE, and so on. (With one small change: when constructing an interpolation L of this data, we don't need to include time, as that was already done when constructing X. Thus the interpolated path is $L \colon [t_0, t_n] \to \mathbb{R}^{\beta(v,N)}$ with $L(r_i) = \sum_{j=0}^{i} L_j$.)

3.2 DISCUSSION

Binning Essentially all we have done is bin our data, with a carefully chosen bin statistic, given by the logsignature transform. The intuition is that the logsignature transform is a map that extracts the information most relevant for understanding how a path can drive a CDE.

Variation speed The sequence of logsignatures 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.

Length/channel trade-off The length of the input sequence has been reduced from n to m, whilst the dimension of the sequence has been increased from v to $\beta(v, N) > v$.

Ease of implementation The method is straightforward to implement in code: the logsignature transforms may be applied as a data preprocessing step using readily-available libraries.

Memory efficiency One of the classic problems with long sequences is the need for large amounts of memory to perform backpropagation-through-time (BPTT). However, Neural CDEs support memory-efficient backpropagation via the adjoint equations, see **?**. Whilst not strictly a benefit of the log-ODE scheme, for completeness we note that this is an additional reason for Neural CDEs to be an attractive choice for long time series.

3.3 The depth and step hyperparameters

This method introduces two new hyperparameters: depth N and step size $r_{i+1} - r_i$ (which we typically take as the same for all i, for simplicity).

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.

If depth $N = 1$ and steps $r_i = t_i$ are used, then in fact the time series of equation (**??**) is identical to the input time series. Thus the log-ODE method generalises the usual approach.

4 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 was used 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 $N = 1$ corresponds to not using the log-ODE method as per Section **??**, with data subsampled at rate $1/s$. Thus we use NCDE^1_1 as our benchmark: no subsampling, no log-ODE method.

Each model is run three times and we report the mean and standard deviation of the test metrics along with the mean training times and memory usages.

For each task, the hyperparameters were selected by performing a grid search on the NCDE^s_1 model, where s was chosen so that the length of the sequence was 500 steps. This was found to create a reasonable balance between training time and sequence length. (Doing hyperoptimisation on the baseline NCDE^1_1 model would have been more difficult due to the larger training times.)

Precise details of the experiments can be found in Appendices **??** and **??**.

4.1 Classification with EigenWorms

Our first example uses the EigenWorms dataset from the UEA archive from **?**. 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 **??**. 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 addi-

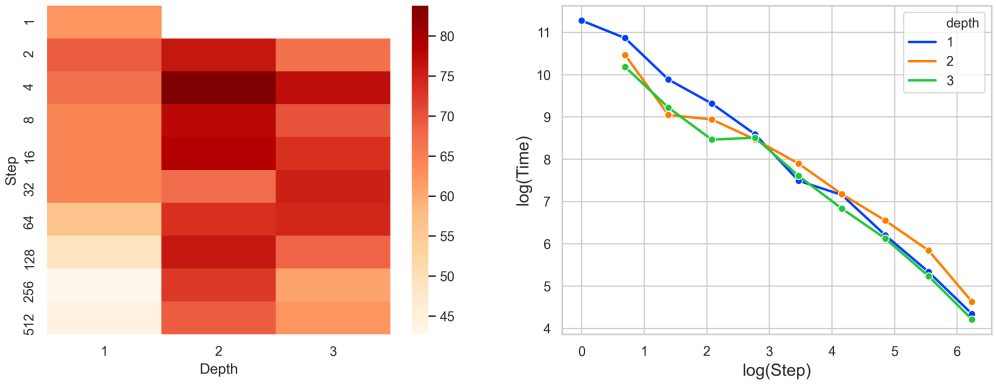


Figure 2: **Left:** Heatmap of accuracies on the EigenWorms dataset for differing step sizes and depths. **Right:** Log-log plot of the elapsed time of the algorithm against the step size.

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	$\mathbf{77.8 \pm 5.9}$	2.1	94.2
	32	67.5 ± 12.1	0.7	28.1
	128	$\mathbf{76.1 \pm 5.9}$	0.2	7.8
NCDE_3	8	70.1 ± 6.5	1.3	460.7
	32	$\mathbf{75.2 \pm 3.0}$	0.6	134.7
	128	68.4 ± 8.2	0.1	53.3

Table 1: Performance on the UEA EigenWorms dataset for depths 1-3 and a selection of step sizes. Bold denotes that the model was the top performer for that step-size.

tionally improves model performance dramatically, and reduces memory usage. Naive subsampling approaches (NCDE_1^8, NCDE_1^{32}, NCDE_1^{128}) only achieve speedups without performance improvements.

See also Figure **??**, in which we summarise results for a larger range of step sizes.

4.2 Estimating Vitals Signs from PPG and ECG Data

Next we consider the problem of estimating vital signs from PPG and ECG data. The data is taken from the TSR archive **?** and involves data from the Beth Israel Deaconess Medical Centre (BIDMC). We consider three tasks whereby we aim to predict a persons respiratory rate (RR), their heart rate (HR), and their oxygen saturation (SpO2). This data is sampled at 125hZ with each series having a length of 4000, 7949 training samples, and a dimension of 3 (including time).

We train a model on each of the three vitals sign prediction tasks using a similar approach to the EigenWorms dataset. The metric used to evaluate performance is the root mean squared error (RMSE) loss which is the standard loss considered in the TSR archive. The results over a range of step sizes are presented in table (**??**). The full results over all step sizes, along with a plots analogous to figure **??** are left to Appendix **??**.

We find that the depth 3 model is the top performer for every task at any step size. Whats more, the top performing depth 3 model from each category also significantly outperforms the NCDE_1^1 model with a significantly reduced training time. This again illustrates that the increased step size not only reduces training time, but can also improve model performance. We believe this is attributable to the log-ODE model being better at learning these long-term dependencies, as the worst scores of the

Depth	Step	RMSE			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	$\mathbf{2.42 \pm 0.19}$	$\mathbf{7.67 \pm 0.40}$	$\mathbf{2.55 \pm 0.13}$	2.9	3.2	3.1	43.3
	32	$\mathbf{1.67 \pm 0.01}$	$\mathbf{4.50 \pm 0.70}$	$\mathbf{1.61 \pm 0.05}$	1.3	1.8	7.3	20.5
	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

Table 2: The RMSE scores on the test set for each of the vitals signs prediction tasks (RR, HR, SpO_2) on the BIDMC dataset. 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.

$NCDE^s_{2,3}$ models are with step sizes 2 and 4 for all tasks where they are on par (or worse) than the corresponding depth 1 model.

Both experiments give show that the depth 2 and 3 models can not only reduce training times with similar levels of performance, but can additionally result in improved performance over these large steps.

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 for low numbers of input channels, as the number of logsignature channels $\beta(v, N)$ grows near-exponentially in v.

6 RELATED WORK

There has been some work on long time series for classic RNN (GRU/LSTM) models. **?** introduce the 'Skip-RNN' model, which extend the RNN by adding an additional learnt component that skips state updates. This reduces the number of computations in the forward pass. However, this model is now not dependent on the within-skip input data. Furthermore, one does not know a priori how many states the model will choose to skip, which can lead to memory difficulties.

Another approach is hierarchical subsampling as in **??**, which operates over multiple training examples at a time as an input, thus reducing the number of total forward computations. Whilst this method is dependent on all the data, it naively uses the raw data when an appropriate summarisation would be more sensible.

This is rectified in **?** where the authors in fact use the logsignature as their summarisation procedure.

However, as is common with all of these models, it describes some some modification of the RNN structure but it has been shown in **?** that the NCDE model subsumes the class of RNN/LSTM/GRU, in that it can model a wider class of functions. Additionally, none of these approaches be solved using ODE methods and thus are not able to utilise the adjoint method, continuous time dynamics, and so on.

7 CONCLUSIONS

We have shown how the integral equation from the Neural CDE model can be solved with high accuracy over intervals larger than the discretisation of the data can be solved using the log-ODE method. We demonstrated that this not only enables us to take longer steps over the data whilst retaining model performance resulting in significantly lower training times, but also that this can result in models that achieve better performance over large step sizes than the original model over any step size, including updating the hidden state at every data point.

AUTHOR CONTRIBUTIONS

ACKNOWLEDGEMENTS

JM was supported by the EPSRC grant EP/L015803/1 in collaboration with Iterex Therapuetics. CS was supported by the EPSRC grant EP/R513295/1. PK was supported by the EPSRC grant EP/L015811/1. 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.

Supplementary material

In sections **??** and **??**, we give a more thorough introduction into solving CDEs via the log-ODE method.

In section **??** we discuss the experimental details such as the choice of network structure, computing infrastructure and hyperparameter selection approach.

In section **??** we give a full breakdown of every experimental result.

A AN INTRODUCTION TO THE LOG-ODE METHOD FOR CONTROLLED DIFFERENTIAL EQUATIONS

The log-ODE method is an effective method for approximating the controlled differential equation:

$$\begin{aligned} dY_t &= f(Y_t)\, dX_t, \\ Y_0 &= \xi, \end{aligned} \tag{11}$$

where $X : [0, T] \to \mathbb{R}^d$ has finite length, $\xi \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to L(\mathbb{R}^d, \mathbb{R}^n)$ is a function with certain smoothness assumptions so that the CDE (**??**) is well posed. Throughout these appendices, $L(U, V)$ denotes the space of linear maps between the vector spaces U and V. In rough path theory, the function f is referred to as the "vector field" of (**??**) and usually assumed to have $Lip(\gamma)$ regularity (see definition 10.2 in **?**). In this section, we assume one of the below conditions on the vector field:

1. f is bounded and has N bounded derivatives.
2. f is linear.

In order to define the log-ODE method, we will first consider the tensor algebra and path signature.

Definition A.1 *We say that $T\big(\mathbb{R}^d\big) := \mathbb{R} \oplus \mathbb{R}^d \oplus (\mathbb{R}^d)^{\otimes 2} \oplus \cdots$ is the **tensor algebra** of $\mathbb{R}^d$ and $T\big(\big(\mathbb{R}^d\big)\big) := \big\{\boldsymbol{a} = \big(a_0, a_1, \cdots\big) : a_k \in \big(\mathbb{R}^d\big)^{\otimes k}\ \forall k \geq 0\big\}$ is the set of formal series of tensors of $\mathbb{R}^d$. Moreover, $T\big(\mathbb{R}^d\big)$ and $T\big(\big(\mathbb{R}^d\big)\big)$ can be endowed with the operations of addition and multiplication. Given $\boldsymbol{a} = (a_0, a_1, \cdots)$ and $\boldsymbol{b} = (b_0, b_1, \cdots)$, we have*

$$\boldsymbol{a} + \boldsymbol{b} = \big(a_0 + b_0, a_1 + b_1, \cdots\big), \tag{12}$$

$$\boldsymbol{a} \otimes \boldsymbol{b} = \big(c_0, c_1, c_2, \cdots\big), \tag{13}$$

where for $n \geq 0$, the n-th term $c_n \in \big(\mathbb{R}^d\big)^{\otimes n}$ can be written using the usual tensor product as

$$c_n := \sum_{k=0}^{n} a_k \otimes b_{n-k}.$$

*The operation $\otimes$ given by (**??**) is often referred to as the "tensor product".*

Definition A.2 *The **signature** of a finite length path $X : [0, T] \to \mathbb{R}^d$ over the interval $[s, t]$ is defined as the following collection of iterated (Riemann-Stieltjes) integrals:*

$$Sig_{s,t}\big(X\big) := \Big(1, X_{s,t}^{(1)}, X_{s,t}^{(2)}, X_{s,t}^{(3)}, \cdots\Big) \in T\big(\big(\mathbb{R}^d\big)\big), \tag{14}$$

where for $n \geq 1$,

$$X_{s,t}^{(n)} := \int\limits_{s<u_1<\cdots<u_n<t} \cdots \int dX_{u_1} \otimes \cdots \otimes dX_{u_n} \in \big(\mathbb{R}^d\big)^{\otimes n}.$$

Similarly, we can define the N-step (or truncated) signature of the path X on $[s, t]$ as

$$Sig_{s,t}^{(N)}\big(X\big) := \left(1, \int_{s<u_1<t} dX_u\,, \cdots, \int\limits_{s<u_1<\cdots<u_N<t} \cdots \int dX_{u_1} \otimes \cdots \otimes dX_{u_N}\right) \in T^N\big(\mathbb{R}^d\big), \tag{15}$$

where $T^N\big(\mathbb{R}^d\big) := \mathbb{R} \oplus \mathbb{R}^d \oplus (\mathbb{R}^d)^{\otimes 2} \oplus \cdots \oplus (\mathbb{R}^d)^{\otimes N}$ denotes the truncated tensor algebra.

The (truncated) signature provides a natural feature set that describes the effects a path X has on systems that can be modelled by (**??**). That said, defining the log-ODE method actually requires the so-called "log-signature" which efficiently encodes the same integral information as the signature. The log-signature is obtained from the path's signature by removing certain algebraic redundancies, such as

$$\int_0^t \int_0^s dX_u^{(i)} dX_s^{(j)} + \int_0^t \int_0^s dX_u^{(j)} dX_s^{(i)} = X_t^{(i)} X_t^{(j)},$$

for $i, j \in \{1, \cdots, d\}$, which follows using the integration by parts formula. To this end, we will define the logarithm map on the N-step truncated tensor algebra $T^N\big(\mathbb{R}^d\big) := \mathbb{R} \oplus \mathbb{R}^d \oplus \cdots \oplus (\mathbb{R}^d)^{\otimes N}$.

Definition A.3 (The logarithm of a formal series) *For $\boldsymbol{a} = (a_0, a_1, \cdots) \in T\big(\big(\mathbb{R}^d\big)\big)$ with $a_0 > 0$, define $\log(\boldsymbol{a})$ to be the element of $T\big(\big(\mathbb{R}^d\big)\big)$ given by the following series:*

$$\log(\boldsymbol{a}) := \log(a_0) + \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \left(\mathbf{1} - \frac{\boldsymbol{a}}{a_0}\right)^{\otimes n}, \tag{16}$$

where $\mathbf{1} = (1, 0, \cdots)$ is the unit element of $T\big(\big(\mathbb{R}^d\big)\big)$ and $\log(a_0)$ is viewed as $\log(a_0)\mathbf{1}$.

Definition A.4 (The logarithm of a truncated series) *For $\boldsymbol{a} = (a_0, a_1, \cdots, a_N) \in T\big(\big(\mathbb{R}^d\big)\big)$ with $a_0 > 0$, define $\log^N(\boldsymbol{a})$ to be the element of $T^N\big(\mathbb{R}^d\big)$ defined from the logarithm map (**??**) as*

$$\log^N(\boldsymbol{a}) := P_N\big(\log(\widetilde{\boldsymbol{a}})\big), \tag{17}$$

where $\widetilde{\boldsymbol{a}} := (a_0, a_1, \cdots, a_N, 0, \cdots) \in T\big(\big(\mathbb{R}^d\big)\big)$ and P_N denotes the standard projection map from $T\big(\big(\mathbb{R}^d\big)\big)$ onto $T^N\big(\mathbb{R}^d\big)$.

Definition A.5 *The **log-signature** of a finite length path $X : [0, T] \to \mathbb{R}^d$ over the interval $[s, t]$ is $LogSig_{s,t}(X) := \log(Sig_{s,t}(X))$, where $Sig_{s,t}(X)$ denotes the path signature of X given by Definition **??**. Likewise, the N-step (or truncated) log-signature of X is defined for each $N \geq 1$ as $LogSig_{s,t}^N(X) := \log^N(Sig_{s,t}^N(X))$.*

The final ingredient we use to define the log-ODE method are the derivatives of the vector field f. It is worth noting that these derivatives also naturally appear in the Taylor expansion of (**??**).

Definition A.6 (Vector field derivatives) *We define $f^{\circ k} : \mathbb{R}^n \to L((\mathbb{R}^d)^{\otimes k}, \mathbb{R}^n)$ recursively by*

$$\begin{aligned} f^{\circ(0)}(y) &:= y, \\ f^{\circ(1)}(y) &:= f(y), \\ f^{\circ(k+1)}(y) &:= D\big(f^{\circ k}\big)(y) f(y), \end{aligned}$$

for $y \in \mathbb{R}^n$, where $D\big(f^{\circ k}\big)$ denotes the Fréchet derivative of $f^{\circ k}$.

Using these definitions, we can describe two closely related numerical methods for the CDE (**??**).

Definition A.7 (The Taylor method) *Given the CDE (**??**), we can use the path signature of X to approximate the solution Y on an interval $[s, t]$ via its truncated Taylor expansion. That is, we use*

$$\textit{Taylor}(Y_s, f, Sig_{s,t}^N(X)) := \sum_{k=0}^{N} f^{\circ k}(Y_s) \pi_k\big(Sig_{s,t}^N(X)\big), \tag{18}$$

as an approximation for Y_t where each $\pi_k : T^N(\mathbb{R}^d) \to (\mathbb{R}^d)^{\otimes k}$ is the projection map onto $\big(\mathbb{R}^d\big)^{\otimes k}$.

Definition A.8 (The Log-ODE method) *Using the Taylor method (**??**), we can define the function $\tilde{f} : \mathbb{R}^n \to L(T^N(\mathbb{R}^d), \mathbb{R}^n)$ by $\tilde{f}(z) := \textit{Taylor}(z, f, \cdot)$. By applying $\tilde{f}$ to the truncated log-signature of the path X over an interval $[s, t]$, we can define the following ODE on $[0, 1]$*

$$\begin{aligned} \frac{dz}{du} &= \tilde{f}(z) LogSig_{s,t}^N(X), \\ z(0) &= Y_s. \end{aligned} \tag{19}$$

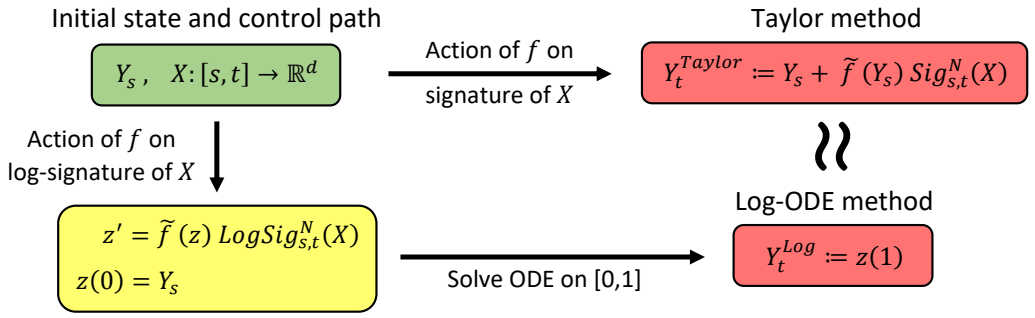


Figure 3: Illustration of the log-ODE and Taylor methods for controlled differential equations.

Then the log-ODE approximation of Y_t (given Y_s and $LogSig_{s,t}^N(X)$) is defined as

$$LogODE(Y_s, f, LogSig_{s,t}^N(X)) := z(1). \tag{20}$$

Remark A.9 *Our assumptions of f ensure that $z \mapsto \tilde{f}(z) LogSig_{s,t}^N(X)$ is either globally bounded and Lipschitz continuous or linear. Hence both the Taylor and log-ODE methods are well defined.*

Remark A.10 *It is well known that the log-signature of a path X lies in a certain free Lie algebra (this is detailed in Section 2.2.4 of **?**). Furthermore, it is also a theorem that the Lie bracket of two vector fields is itself a vector field which doesn't depend on choices of basis. By expressing $LogSig_{s,t}^N(X)$ using a basis of the free Lie algebra, it can be shown that only the vector field f and its (iterated) Lie brackets are required to construct the log-ODE vector field $\tilde{f}(z) LogSig_{s,t}^N(X)$. One of the key advantages of this construction is that is can be extended to define the log-ODE method for CDEs on manifolds. Although this Lie theory is not directly utilized in our framework, we direct the reader to **?** or **?** for further details.*

To illustrate the log-ODE method, we give two examples:

Example A.11 (The "increment-only" log-ODE method) *When $N = 1$, the ODE (**??**) becomes*

$$\begin{aligned} \frac{dz}{du} &= f(z) X_{s,t}, \\ z(0) &= Y_s. \end{aligned}$$

*Therefore we see that this "increment-only" log-ODE method is equivalent to driving the original CDE (**??**) by a piecewise linear approximation of the control path X. This is a classical approach for stochastic differential equations (i.e. when $X_t = (t, W_t)$ with W denoting a Brownian motion) and is an example of a Wong-Zakai approximation (see **?** for further details).*

Example A.12 (An application for SDE simulation) *Consider the following affine SDE,*

$$\begin{aligned} dy_t &= a(b - y_t)\, dt + \sigma\, y_t \circ dW_t, \\ y(0) &= y_0 \in \mathbb{R}_{\geq 0}, \end{aligned} \tag{21}$$

*where $a, b \geq 0$ are the mean reversion parameters, $\sigma \geq 0$ is the volatility and W denotes a standard real-valued Brownian motion. The $\circ$ means that this SDE is understood in the Stratonovich sense. The SDE (**??**) is known in the literature as Inhomogeneous Geometric Brownian Motion (or IGBM). Using the control path $X = \{(t, W_t)\}_{t \geq 0}$ and setting $N = 3$, the log-ODE (**??**) becomes*

$$\begin{aligned} \frac{dz}{du} &= a(b - z_u)h + \sigma\, z_u W_{s,t} - ab\sigma A_{s,t} + ab\sigma^2 L_{s,t}^{(1)} + a^2 b\sigma L_{s,t}^{(2)}, \\ z(0) &= Y_s. \end{aligned}$$

where $h := t - s$ denotes the step size and the random variables $A_{s,t}, L^{(1)}_{s,t}, L^{(2)}_{s,t}$ are given by

$$A_{s,t} := \int_s^t W_{s,r}\, dr - \frac{1}{2} h W_{s,t},$$
$$L^{(1)}_{s,t} := \int_s^t \int_s^r W_{s,v} \circ dW_v\, dr - \frac{1}{2} W_{s,t} A_{s,t} - \frac{1}{6} h W^2_{s,t},$$
$$L^{(2)}_{s,t} := \int_s^t \int_s^r W_{s,v}\, dv\, dr - \frac{1}{2} h A_{s,t} - \frac{1}{6} h^2 W_{s,t}.$$

*In **?**, the 3-step log-signature of $X = \{(t, W_t)\}_{t \geq 0}$ was approximated so that the above log-ODE method became practical and this numerical scheme exhibited state-of-the-art convergence rates. For example, the approximation error produced by 25 steps of the high order log-ODE method was similar to the error of the "increment only" log-ODE method with 1000 steps.*

A.1 The generalized log-ODE method

The log-ODE method provides a natural map on path space. That is, we can view the ODE (**??**) simply as a function of the path X. Ultimately, the vector field of this ODE will be parametrized, and this naturally leads us to consider alternatives to $\tilde{f}$ (such as linear functions or neural networks). To this end, we extend the log-ODE method given by Definition **??** to include general vector fields.

Definition A.13 (The generalized log-ODE) *Given a path $X : [0, T] \to \mathbb{R}^d$ with finite length, we consider the following ODE defined on the interval $[0, 1]$,*

$$\frac{dz}{du} = F(z)\, LogSig^N_{0,T}(X), \tag{22}$$
$$z(0) = z_0,$$

*where $N \geq 1$ and $F : \mathbb{R}^n \to L\big(T^N(\mathbb{R}^d), \mathbb{R}^n\big)$ satisfies the smoothness conditions so that (**??**) is well posed. We define a generalized log-ODE method to be any function on path space that can be obtained by approximating the solution at $u = 1$ of an ODE with the form (**??**).*

Remark A.14 *As one would expect, we can take a partition $\triangle_K = \{0 = t_0 < t_1 < \cdots < t_K = T\}$ of $[0, T]$ and construct a discrete process $\{Y_t\}_{t \in \triangle_K}$ from $Y_0 \in \mathbb{R}^n$ as $Y_{t_{k+1}} := z^k(1)$ for $k \geq 0$, where z^k denotes the solution to the following generalized log-ODE:*

$$\frac{dz^k}{du} = F(z^k)\, LogSig^N_{t_k, t_{k+1}}(X),$$
$$z^k(0) = Y_{t_k}.$$

When the function F comes from a parametrized family, we arrive at the proposed neural differential equation model.

B Convergence of the log-ODE method for rough differential equations

In this section, we shall present "rough path" error estimates for the log-ODE method. In addition, we will discuss the case when the vector fields governing the rough differential equation are linear.

Theorem B.1 (Lemma 15 in ?) *Consider the rough differential equation*

$$dY_t = f(Y_t)\, dX_t, \tag{23}$$
$$Y_0 = \xi,$$

where we make the following assumptions:

- *X is a geometric p-rough path in $\mathbb{R}^d$, that is $X : [0,T] \to T^{\lfloor p \rfloor}(\mathbb{R}^d)$ is a continuous path in the tensor algebra $T^{\lfloor p \rfloor}(\mathbb{R}^d) := \mathbb{R} \oplus \mathbb{R}^d \oplus \left(\mathbb{R}^d\right)^{\otimes 2} \oplus \cdots \oplus \left(\mathbb{R}^d\right)^{\otimes \lfloor p \rfloor}$ with increments*

$$\begin{aligned} X_{s,t} &= \left(1, X^1_{s,t}, X^2_{s,t}, \cdots, X^{\lfloor p \rfloor}_{s,t}\right), \\ X^k_{s,t} &:= \pi_k\big(X_{s,t}\big), \end{aligned} \tag{24}$$

where $\pi_k : T^{\lfloor p \rfloor}\big(\mathbb{R}^d\big) \to \left(\mathbb{R}^d\right)^{\otimes k}$ is the projection map onto $\left(\mathbb{R}^d\right)^{\otimes k}$, such that there exists a sequence of continuous finite variation paths $x_n : [0,T] \to \mathbb{R}^d$ whose truncated signatures converge to X in the p-variation metric:

$$d_p\Big(S^{\lfloor p \rfloor}_{0,T}(x_n), X\Big) \to 0, \tag{25}$$

as $n \to \infty$, where the p-variation between two continuous paths Z^1 and Z^2 in $T^{\lfloor p \rfloor}(\mathbb{R}^d)$ is

$$d_p\big(Z_1, Z_2\big) := \max_{1 \le k \le \lfloor p \rfloor} \sup_{\mathcal{D}} \left(\sum_{t_i \in \mathcal{D}} \left\| \pi_k\big(Z^1_{t_i,t_{i+1}}\big) - \pi_k\big(Z^2_{t_i,t_{i+1}}\big) \right\|^p \right)^{\frac{1}{p}}, \tag{26}$$

where the supremum is taken over all partitions $\mathcal{D}$ of $[0,T]$ and the norms $\|\cdot\|$ must satisfy (up to some constant)

$$\|a \otimes b\| \le \|a\| \|b\|,$$

*for $a \in (\mathbb{R}^d)^{\otimes n}$ and $b \in (\mathbb{R}^d)^{\otimes m}$. For example, we can take $\|\cdot\|$ to be the projective or injective tensor norms (see Propositions 2.1 and 3.1 in **?**).*

- *The solution Y and its initial value ξ both take their values in $\mathbb{R}^n$.*

- *The collection of vector fields $\{f_1, \cdots, f_d\}$ on $\mathbb{R}^n$ are denoted by $f : \mathbb{R}^n \to L(\mathbb{R}^n, \mathbb{R}^d)$, where $L(\mathbb{R}^n, \mathbb{R}^d)$ is the space of linear maps from $\mathbb{R}^n$ to $\mathbb{R}^d$. We will assume that f has $Lip(\gamma)$ regularity with $\gamma > p$. That is, the following norm is finite:*

$$\|f\|_{Lip(\gamma)} := \max_{0 \le k \le \lfloor \gamma \rfloor} \left\| D^k f \right\|_\infty \vee \left\| D^{\lfloor \gamma \rfloor} f \right\|_{(\gamma - \lfloor \gamma \rfloor)\text{-Höl}}, \tag{27}$$

where $D^k f$ is the k-th (Fréchet) derivative of f and $\|\cdot\|_{\alpha\text{-Höl}}$ is the standard α-Hölder norm for $\alpha \in (0,1)$.

- *The RDE (**??**) is defined in the Lyon's sense, so that by the Universal Limit Theorem (see Theorem 5.3 in **?**), there exists a unique solution $Y : [0,T] \to \mathbb{R}^n$.*

We define the log-ODE for approximating the solution Y over an interval $[s,t] \subset [0,T]$ as follows:

1. *Compute the $\lfloor \gamma \rfloor$-step log-signature of the control path X over $[s,t]$. That is, we obtain $\log_{\lfloor \gamma \rfloor}\left(S^{\lfloor \gamma \rfloor}_{s,t}(X)\right) \in T^{\lfloor \gamma \rfloor}(\mathbb{R}^d)$, where $\log_{\lfloor \gamma \rfloor}(\cdot)$ is defined by projecting the standard tensor logarithm map onto $\{a \in T^{\lfloor \gamma \rfloor}(\mathbb{R}^d) : \pi_0(a) > 0\}$.*

2. *Construct the following (well-posed) ODE on the interval $[0,1]$,*

$$\begin{aligned} \frac{dz^{s,t}}{du} &:= F\big(z^{s,t}\big), \\ z^{s,t}_0 &:= Y_s, \end{aligned} \tag{28}$$

where the vector field $F : \mathbb{R}^n \to \mathbb{R}^n$ is defined from the log-signature as

$$F(z) := \sum_{k=1}^{\lfloor \gamma \rfloor} f^{\circ k}(z) \pi_k\Big(\log_{\lfloor \gamma \rfloor}\left(S^{\lfloor \gamma \rfloor}_{s,t}(X)\right)\Big). \tag{29}$$

*Recall that $f^{\circ k} : \mathbb{R}^n \to L((\mathbb{R}^d)^{\otimes k}, \mathbb{R}^n)$ was defined previously in Definition **??**.*

*Then we can approximate Y_t using the $u = 1$ solution of (**??**). Moreover, there exists a universal constant $C_{p,\gamma}$ depending only on p and γ such that*

$$\left\|Y_t - z_1^{s,t}\right\| \leq C_{p,\gamma}\|f\|_{Lip(\gamma)}^{\gamma}\|X\|_{p\text{-}var;[s,t]}^{\gamma}, \tag{30}$$

where $\|\cdot\|_{p\text{-}var;[s,t]}$ is the p-variation norm defined for paths in $T^{\lfloor p \rfloor}(\mathbb{R}^d)$ by

$$\|X\|_{p\text{-}var;[s,t]} := \max_{1\leq k\leq \lfloor p \rfloor} \sup_{\mathcal{D}} \left(\sum_{t_i \in \mathcal{D}} \left\|X_{t_i,t_{i+1}}^k\right\|^p \right)^{\frac{1}{p}}, \tag{31}$$

with the supremum taken over all partitions $\mathcal{D}$ of $[s,t]$.

Remark B.2 *If the vector fields $\{f_1, \cdots, f_d\}$ are linear, then it is a straightforward exercise to show that F is linear.*

Although the above theorem requires some sophisticated theory, it has a simple conclusion - namely that log-ODEs can approximate controlled differential equation. That said, the estimate (**??**) does not directly apply when the vector fields $\{f_i\}$ are linear as they are unbounded. Fortunately, it is well known that solutions of linear RDEs exist and are unique.

Theorem B.3 (Theorem 10.57 in ?) *Consider the linear RDE on $[0,T]$*

$$dY_t = f(Y_t)\,dX_t,$$
$$Y_0 = \xi,$$

where X is a geometric p-rough path in $\mathbb{R}^d$, $\xi \in \mathbb{R}^n$ and the vector fields $\{f_i\}_{1\leq i\leq d}$ take the form $f_i(y) = A_i y + B$ where $\{A_i\}$ and $\{B_i\}$ are $n \times n$ matrices. Let K denote an upper bound on $\max_i(\|A_i\| + \|B_i\|)$. Then a unique solution $Y : [0,T] \to \mathbb{R}^n$ exists. Moreover, it is bounded and there exists a constant C_p depending only on p such that

$$\|Y_t - Y_s\| \leq C_p\big(1 + \|\xi\|\big)K\|X\|_{p\text{-}var;[s,t]} \exp\Big(C_p K^p \|X\|_{p\text{-}var;[s,t]}^p\Big), \tag{32}$$

for all $0 \leq s \leq t \leq T$.

When the vector fields of the RDE (**??**) are linear, then the log-ODE (**??**) also becomes linear. Therefore, the log-ODE solution exists and is explicitly given as the exponential of the matrix F.

Theorem B.4 *Consider the same linear RDE on $[0,T]$ as in Theorem **??**,*

$$dY_t = f(Y_t)\,dX_t,$$
$$Y_0 = \xi.$$

*Then the log-ODE vector field F given by (**??**) is linear and the solution of the associated ODE (**??**) exists and satisfies*

$$\|z_u^{s,t}\| \leq \|Y_s\| \exp\left(\sum_{m=1}^{\lfloor \gamma \rfloor} K^m \left\| \pi_m\Big(\log_{\lfloor \gamma \rfloor}\left(S_{s,t}^{\lfloor \gamma \rfloor}(X)\right)\Big)\right\| \right), \tag{33}$$

for $u \in [0,1]$ and all $0 \leq s \leq t \leq T$.

Proof B.5 *Since F is a linear vector field on $\mathbb{R}^n$, we can view it as an $n \times n$ matrix and so for $u \in [0,1]$,*

$$z_u^{s,t} = \exp(uF) z_0^{s,t},$$

where exp *denotes the matrix exponential. The result now follows by the standard estimate $\|\exp(F)\| \leq \exp(\|F\|)$.*

Remark B.6 *By the boundedness of linear RDEs (**??**) and log-ODEs (**??**), the arguments that established Theorem **??** hold in the linear case as $\|f\|_{Lip(\gamma)}$ would be finite when defined using the domains that the solutions Y and z lie in.*

Given the local error estimate (**??**) for the log-ODE method, we can now consider the approximation error that is exhibited by a log-ODE numerical solution to the RDE (**??**). Fortunately, the analysis required to derive such global error estimates was developed by Greg Gyurkó in his PhD thesis. Thus, the following result is a straightforward application of Theorem 3.2.1 from **?**.

Theorem B.7 *Let X, f and Y satisfy the assumptions given by Theorem* **??** *and suppose that $\{0 = t_0 < t_1 < \cdots < t_N = T\}$ is a partition of $[0, T]$ with $\max_k \|X\|_{p\text{-}var;[t_k,t_{k+1}]}$ sufficiently small. We can construct a numerical solution $\{Y_k^{Log}\}_{0\leq k\leq N}$ of (**??**) by setting $Y_0^{Log} := Y_0$ and for each $k \in \{0, 1, \cdots, N-1\}$, defining Y_{k+1}^{Log} to be the solution at $u = 1$ of the following ODE:*

$$\begin{aligned} \frac{dz^{t_k,t_{k+1}}}{du} &:= F\big(z^{t_k,t_{k+1}}\big), \\ z_0^{t_k,t_{k+1}} &:= Y_k^{Log}, \end{aligned} \tag{34}$$

*where the vector field F is constructed from the log-signature of X over the interval $[t_k, t_{k+1}]$ according to (**??**). Then there exists a constant C depending only on p, γ and $\|f\|_{Lip(\gamma)}$ such that*

$$\big\|Y_{t_k} - Y_k^{Log}\big\| \leq C \sum_{i=0}^{k-1} \|X\|^{\gamma}_{p\text{-}var;[t_i,t_{i+1}]}, \tag{35}$$

for $0 \leq k \leq N$.

Remark B.8 *The above error estimate also holds when the vector field f is linear (by Remark* **??***).*

Since $\lfloor\gamma\rfloor$ is the truncation depth of the log-signatures used to construct each log-ODE vector field, we see that high convergence rates can be achieved through using more terms in each log-signature. It is also unsurprising that the error estimate (**??**) increases with the "roughness" of the control path. So just as in our experiments, we see that the performance of the log-ODE method can be improved by choosing an appropriate step size and depth of log-signature.

C EXPERIMENTAL DETAILS

Code The code to reproduce the experiments is available at `https://github.com/jambo6/neural_rdes`.

Data splits Each dataset was split into a training, validation, and testing dataset with relative sizes 0.70/0.15/0.15.

Normalisation The training splits of each dataset were normalised to zero mean and unit variance using a standard scaling approach. The parameters from the training set were then used to normalise the validation and testing datasets.

Architecture We give graphical description of the architecture used for updating the Neural CDE hidden state in figure **??**.

Activation functions As seen in figure **??**, the final activation function before the ODE solve was the Tanh nonlinearity, with all other nonlinearities being ReLU. Tanh was used to bound the output of the nonlinear hidden layer as we found that otherwise we developed stability problems.

ODE Solver All problems used the 'rk4' solver implemented in torchdiffeq.

Computing infrastructure All EigenWorms experiments were run on a computer equipped with three GeForce RTX 2080 Ti's. All BIDMC experiments were run on a computed with two GeForce RTX 2080 Ti's and two Quadro GP100's.

Optimiser All experiments used the Adam optimiser. The learning rate was initialised at $0.01 * 32$ divided by batch size. The batch size used was 1024 for EigenWorms and 512 for the BIDMC problems. If the validation loss failed to decrease after 15 epochs the learning rate was reduced by a factor of 10. If the validation loss did not decrease after 60 epochs, training was terminated and the model was rolled back to the point at which it achieved the lowest loss on the validation set.

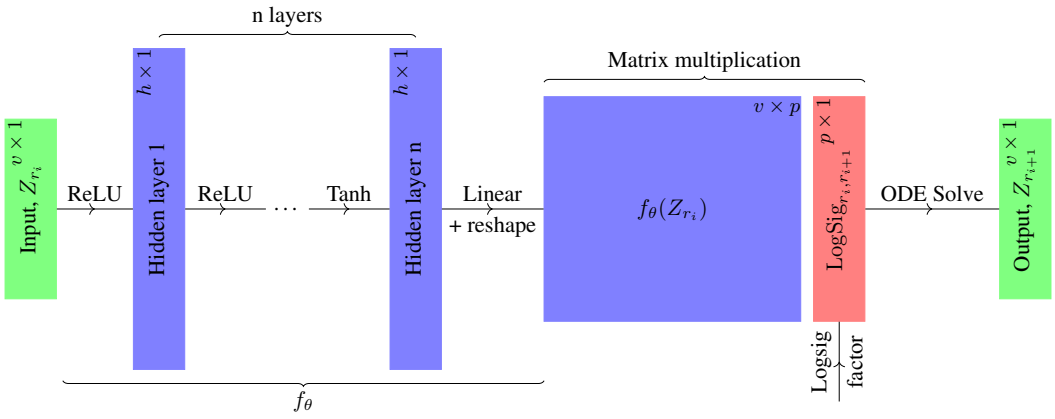


Figure 4: Overview of the hidden state update network structure. We give the dimensions at each layer in the top right hand corner of each box.

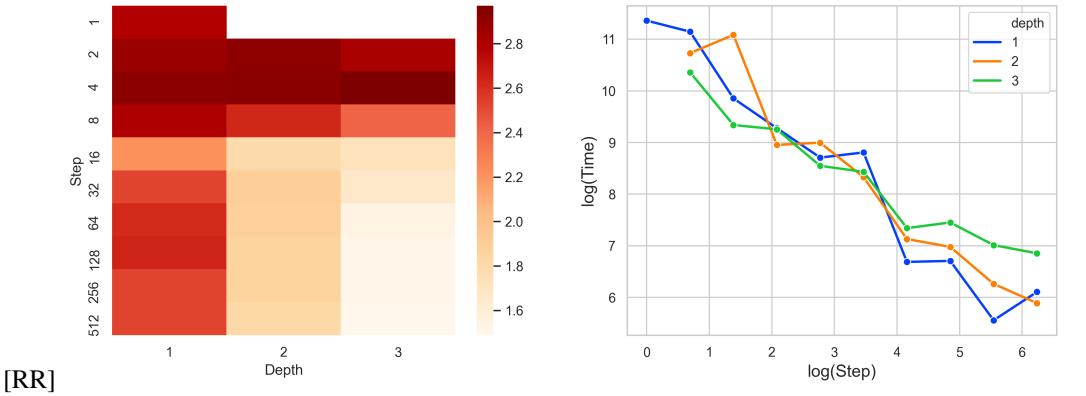
Hyperparameter selection Hyperparameters were selected to optimise the score of the NCDE_1 model on the validation set. For each dataset the search was performed with a step size that meant the total number of hidden state updates was equal to 500, as this represented a good balance between length and speed that allowed us to complete the search in a reasonable time-frame. In particular, this was short enough that we could train using the non-adjoint training method which helped to speed this section up. The hyperparameters that were considered were:

- Hidden dimension: [16, 32, 64] - The dimension of the hidden state Z_t.
- Number of layers: [2, 3, 4] - The number of hidden state layers.
- Hidden hidden multiplier: [1, 2, 3] - Multiplication factor for the hidden hidden state, this being the 'Hidden layer k' in figure **??**. The dimension of each of these 'hidden hidden' layers with be this value multiplied by 'hidden dim'.

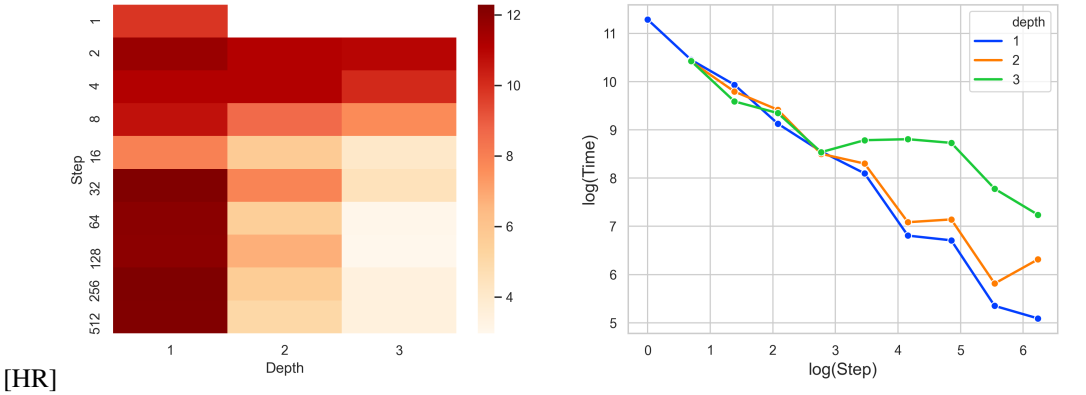
We ran each of these 27 total combinations for every dataset and the parameters that corresponded were used as the parameters when training over the full depth and step grid. The full results from the hyperparameter search are listed in tables (**??**, **??**) with bolded values to show which values were eventually selected.

D Experimental Results

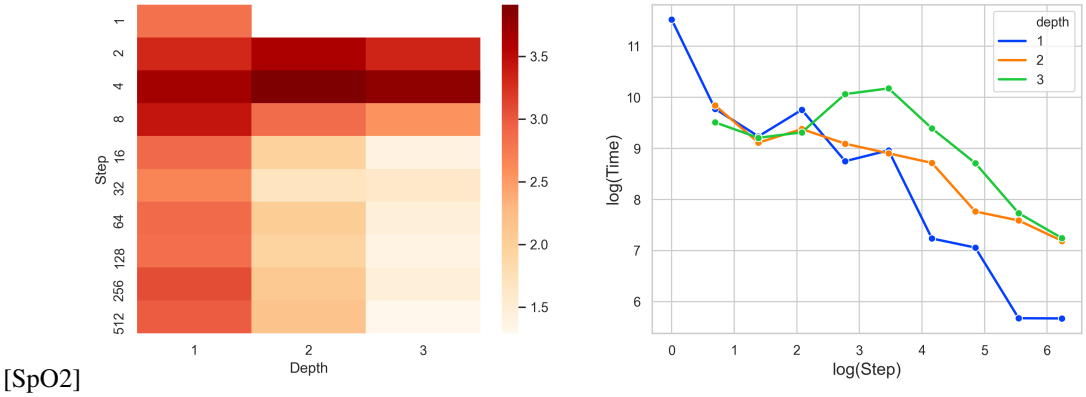
Here we include the full breakdown of all experimental results. Tables **??** and **??** include all results from the EigenWorms and BIDMC datasets respectively. We also give the heatmap style breakdown of the BIDMC results analogous to figure **??** in figure **??**.



[RR]



[HR]



[SpO2]

Figure 5: Heatmaps of the loss on the test set for the all BIDMC problems and the and a log-log plot of the elapsed time of the algorithm against the step size. Recall that lower is better here unlike for EigenWorms.

Validation accuracy	Hidden dim	Num layers	Hidden hidden multiplier	Total params
33.3	16	2	3	5509
43.6	16	2	2	5509
56.4	16	2	1	4453
64.1	16	3	2	8869
38.5	16	3	3	8869
51.3	16	3	1	6517
82.1	16	4	2	12741
35.9	16	4	3	12741
53.8	16	4	1	8581
35.9	32	2	3	21253
74.4	32	2	2	21253
43.6	32	2	1	17093
53.8	32	3	3	34629
87.2	**32**	**3**	**2**	**34629**
64.1	32	3	1	25317
35.9	32	4	3	50053
71.8	32	4	1	33541
79.5	32	4	2	50053
41.0	64	2	3	83461
64.1	64	2	2	83461
48.7	64	3	3	136837
59.0	64	3	2	136837
51.3	64	2	1	66949
56.4	64	4	2	198405
64.1	64	4	3	198405
64.1	64	3	1	99781
51.3	64	4	1	132613

Table 3: Hyperparamter selection results for the EigenWorms dataset. The blue values denote the selected hyperparameters.

Validation loss			**Hidden dim**	**Num layers**	**Hidden hidden multiplier**	**Total params**
RR	HR	SpO2				
1.72	6.10	2.07	16	2	1	2209
1.57	5.58	1.97	16	2	2	3265
1.55	6.10	1.33	16	2	3	3265
1.80	5.16	2.05	16	3	1	3249
1.61	5.22	1.62	16	3	2	5601
1.56	3.34	1.18	16	3	3	5601
1.57	3.86	1.97	16	4	1	4289
1.45	3.54	1.25	16	4	2	8449
1.54	3.93	1.09	16	4	3	8449
1.56	6.81	1.87	32	2	1	8513
1.42	3.11	1.43	32	2	2	12673
1.54	3.60	1.11	32	2	3	12673
1.54	3.52	1.57	32	3	1	12641
1.39	2.96	1.03	32	3	2	21953
1.47	2.95	1.05	32	3	3	21953
1.55	3.00	2.00	32	4	1	16769
1.38	3.20	1.07	32	4	2	33281
1.43	2.58	1.01	32	4	3	33281
1.51	3.21	1.10	64	2	1	33409
1.43	**2.22**	1.00	**64**	**2**	**2**	**49921**
1.51	3.34	0.94	64	2	3	49921
1.55	3.24	2.09	64	3	1	49857
1.32	2.53	0.88	64	3	2	86913
1.25	2.57	**0.73**	**64**	**3**	**3**	**86913**
1.43	5.78	1.43	64	4	1	66305
1.28	2.26	0.93	64	4	2	132097
1.32	2.46	1.15	64	4	3	132097

Table 4: Hyperparameter selection results for each problem of the BIDMC dataset. The bold values denote the selected hyperparameters for each vitals sign problem. Note that RR and SpO2 had the same parameters selected, hence why only two lines are given in bold.

Model	Step	Test Accuracy	Time (Hrs)	Memory (Mb)
NCDE_1	1	62.4 ± 12.1	22.0	176.5
	2	69.2 ± 4.4	14.6	90.6
	4	66.7 ± 11.8	5.5	46.6
	8	64.1 ± 13.3	3.1	24.3
	16	64.1 ± 16.8	1.5	13.4
	32	64.1 ± 14.3	0.5	8.0
	64	56.4 ± 6.8	0.4	5.2
	128	48.7 ± 2.6	0.1	3.9
	256	42.7 ± 3.0	0.1	3.2
	512	44.4 ± 5.3	0.0	2.9
NCDE_2	2	$\mathbf{76.1 \pm 13.2}$	9.8	354.3
	4	$\mathbf{83.8 \pm 3.0}$	2.4	180.0
	8	$\mathbf{77.8 \pm 5.9}$	2.1	94.2
	16	$\mathbf{78.6 \pm 3.9}$	1.3	50.2
	32	67.5 ± 12.1	0.7	28.1
	64	73.5 ± 7.8	0.4	17.2
	128	$\mathbf{76.1 \pm 5.9}$	0.2	7.8
	256	$\mathbf{72.6 \pm 12.1}$	0.1	8.9
	512	$\mathbf{69.2 \pm 11.8}$	0.0	7.6
NCDE_3	2	66.7 ± 4.4	7.4	1766.2
	4	76.9 ± 9.2	2.8	856.8
	8	70.1 ± 6.5	1.3	460.7
	16	73.5 ± 3.0	1.4	243.7
	32	$\mathbf{75.2 \pm 3.0}$	0.6	134.7
	64	$\mathbf{74.4 \pm 11.8}$	0.3	81.0
	128	68.4 ± 8.2	0.1	53.3
	256	60.7 ± 8.2	0.1	40.2
	512	62.4 ± 10.4	0.0	33.1

Table 5: Test set accuracy (in %), memory usage and training time on the UEA 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	RMSE			Time (H)			Memory (Mb)
		RR	HR	SpO_2	RR	HR	SpO_2	
1	1	2.79 ± 0.04	9.82 ± 0.34	2.83 ± 0.27	23.8	22.1	28.1	56.5
$NCDE_1$	2	2.87 ± 0.03	11.69 ± 0.38	$\mathbf{3.31 \pm 0.26}$	19.3	9.6	4.9	32.5
	4	2.92 ± 0.08	11.15 ± 0.49	$\mathbf{3.68 \pm 0.09}$	5.3	5.7	2.8	20.1
	8	2.80 ± 0.06	10.72 ± 0.24	3.43 ± 0.17	3.0	2.6	4.8	14.3
	16	2.22 ± 0.07	7.98 ± 0.61	2.90 ± 0.11	1.7	1.4	1.8	11.8
	32	2.53 ± 0.23	12.23 ± 0.43	2.68 ± 0.12	1.9	0.9	2.2	9.8
	64	2.63 ± 0.11	12.02 ± 0.09	2.88 ± 0.06	0.2	0.3	0.4	9.1
	128	2.64 ± 0.18	11.98 ± 0.37	2.86 ± 0.04	0.2	0.2	0.3	8.7
	256	2.53 ± 0.04	12.29 ± 0.10	3.08 ± 0.10	0.1	0.1	0.1	8.3
	512	2.53 ± 0.03	12.22 ± 0.11	2.98 ± 0.04	0.1	0.0	0.1	8.4
$NCDE_2$	2	2.91 ± 0.10	11.11 ± 0.23	3.63 ± 0.05	12.7	9.3	5.2	58.2
	4	$\mathbf{2.92 \pm 0.04}$	11.14 ± 0.20	3.91 ± 0.24	18.1	5.0	2.5	33.9
	8	2.63 ± 0.12	8.63 ± 0.24	2.88 ± 0.15	2.1	3.4	3.3	21.8
	16	1.80 ± 0.07	5.73 ± 0.45	1.98 ± 0.21	2.2	1.4	2.5	16.0
	32	1.90 ± 0.02	7.90 ± 1.00	1.69 ± 0.20	1.2	1.1	2.0	13.1
	64	1.89 ± 0.04	5.54 ± 0.45	2.04 ± 0.07	0.3	0.3	1.7	11.6
	128	1.86 ± 0.03	6.77 ± 0.42	1.95 ± 0.18	0.3	0.4	0.7	10.9
	256	1.86 ± 0.09	5.64 ± 0.19	2.10 ± 0.19	0.1	0.1	0.5	10.5
	512	1.81 ± 0.02	5.05 ± 0.23	2.17 ± 0.18	0.1	0.2	0.4	10.3
$NCDE_3$	2	$\mathbf{2.82 \pm 0.08}$	$\mathbf{11.01 \pm 0.28}$	4.1 ± 0.72	8.8	9.4	3.7	125.2
	4	2.97 ± 0.23	$\mathbf{10.13 \pm 0.62}$	3.80 ± 0.16	3.2	4.1	2.8	71.6
	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
	16	$\mathbf{1.74 \pm 0.05}$	$\mathbf{4.11 \pm 0.61}$	$\mathbf{1.40 \pm 0.06}$	1.4	1.4	6.5	29.1
	32	$\mathbf{1.67 \pm 0.01}$	$\mathbf{4.50 \pm 0.70}$	$\mathbf{1.61 \pm 0.05}$	1.3	1.8	7.3	20.5
	64	$\mathbf{1.53 \pm 0.08}$	$\mathbf{3.05 \pm 0.36}$	$\mathbf{1.48 \pm 0.14}$	0.4	1.9	3.3	17.9
	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
	256	$\mathbf{1.51 \pm 0.06}$	$\mathbf{3.4 \pm 0.74}$	$\mathbf{1.47 \pm 0.07}$	0.3	0.7	0.6	16.6
	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 6: The RMSE scores on the test set for each of the vitals signs prediction tasks (RR, HR, SpO_2) on the BIDMC dataset. 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.