Machine Learning Applications of Controlled Differential Equations to Streamed Data

James Morrill

Mansfield College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Contents

List of Figures

List of Tables

List of Source Code

Abstract

The amalgamation of rough path theory and machine learning for sequential data has been a topic of increasing interest over the last ten or so years. The unity of these two subject areas is a natural one: rough path theory provides us with the language to describe the solutions to differential equations driven by multidimensional (and potentially highly irregular) signals, and machine learning provides the tools to learn such solutions from data.

The central aim of rough path theory is to provide a general mathematical framework for answering questions about the effect a stream of data can have on a system. One common example of such data is a time series, pervasive in all areas of life (and will be the type of streams we consider most frequently in this thesis); as such, framing problems in the language of rough paths provides us with models that have genuine utility in the real-world.

The aim of this thesis is to provide an accessible introduction to the field of rough path theory for application in machine learning, and then to provide an account of recent and effective contributions that further connect the two fields.

Topics covered in this thesis include: *neural controlled differential equations* (neural CDEs) – extensions of neural ordinary differential equation that can incorporate changes in external data processes; *neural rough differential equations* (neural RDEs) – a rough path extension to neural CDEs that leads to benefits for long or high-frequency time series; the *generalised signature method* – a collection of feature extraction techniques for multivariate time series; and finally real-world applications of the signature method on sepsis and stress detection.

Acknowledgements

First and foremost, I would like to express my deepest thanks to my supervisors, Terry Lyons and Sam Howison.

Three years ago I appeared unannounced at Sam's door, with no supervisor and only a sketch of an idea for a three-year project. Over the coming weeks, with no obligation to do so, Sam dedicated several hours from his free time to help develop the ideas for my thesis and identify a potential supervisor – even ending up becoming my secondary supervisor in the process.

To Terry Lyons, thank you for being everything I could hope for as a PhD supervisor. Your enthusiasm for new ideas, vision for the project, generosity with time – and not to mention back-catalogue of mathematical contributions that I have had the privilege of being able to work on – have made me into the researcher and person that I am today.

I would also like to express my greatest thanks to Sumanth Swaminathan, for the initial research proposal, and for acting as my industrial supervisor for the duration of my PhD. My memories of times spent at the company in New York are some of the fondest of my life. I feel extremely privileged to have been able to visit during my PhD, and I thank you greatly for your generosity financially, with your time, and with your enduring advice about research and life in general.

Thank you to all my wonderful collaborators: Patrick Kidger, Cristopher Salvi, Adeline Fermanian, Andrey Kormilitzin, James Foster, and Lingyi Yang.

Thank you to my examiners Sam Cohen and Lars Ruthotto for an engaging viva and for providing detailed feedback, improvements, and guidance on future research directions.

To Chris and Colin, thank you for InFoMM and everything that comes with it, including your continual support and help to shape us into the researchers we all are today. Special thanks to Nabil and Ferran for their welcoming nature and support during the early years; and to Sheng, Zhen, Harry, Tom, Giuseppe, Rahil and the rest of cohort 4.

Finally, I would like to thank my parents, Oli, Anna, Max, and Andre.

Originality

Statement The majority of Chapters in this thesis (Chapters 3 to 7) are based on work that has already been accepted for publication. If I was lead author producing the main draft of the paper, text has sometimes been taken directly from the paper – this text will contain certain edits from collaborators. If this was not the lead author, everything has been rewritten from scratch. A detailed breakdown of my contributions to each paper is given below.

Publications and contributions Below are the publications that have been produced as a result of work in this DPhil. We will mark at the beginning of each chapter which (if any) of the papers the chapter is based on.

- Kidger, P., **Morrill, J.**, Foster, J., and Lyons, T. (2020a). "Neural Controlled Differential Equations for Irregular Time Series." *Advances in Neural Information Processing Systems (2020).*

- **Morrill, J.**, Kidger, P., Yang, L., and Lyons, T. (2021a). "Neural controlled differential equations for online prediction tasks." *arXiv preprint arXiv:2106.11028.*

- **Morrill, J.**, Salvi, C., Kidger, P., Foster, J., and Lyons, T. (2021c). "Neural rough differential equations for long time series." *International Conference on Machine Learning (2021).*

- **Morrill, J.**, Fermanian, A., Kidger, P., and Lyons, T. (2020a). "A generalised signature method for time series." *arXiv preprint arXiv:2006.00873.*

- **Morrill, J.**, Kormilitzin, A., Nevado-Holgado, A., Swaminathan, S., Howison, S.,and Lyons, T. (2019). "The signature-based model for early detection of sepsis from electronic health records in the intensive care unit." *2019 Computing in Cardiology (CinC), pages Page–1. IEEE.*

- **Morrill, J.**, Kormilitzin, A., Nevado-Holgado, A. J., Swaminathan, S., Howison, S. D., and Lyons, T. J. (2020b). "Utilization of the signature method to identify the early onset of sepsis from multivariate physiological time series in critical care monitoring." *Critical Care Medicine, 48(10):e976–e981.*

The individual breakdowns are as follows:

● **Neural CDEs** - This is based on the paper *Neural controlled differential equations for irregular time series* which was the work of Patrick Kidger. Cristopher Salvi, James Foster, and myself were working on similar ideas at the same time which we turned into the Neural RDEs paper. The writing up of the work in this document is my own, but the experimental results are taken (with permission) from Patrick.

● **Online Neural CDEs** - Using rectilinear control paths to enable Neural CDEs to operate in an online fashion was an idea Terry Lyons and I had been talking about for a long time. Patrick Kidger had done some earlier work regarding theoretical conditions and I proposed merging the two into a single paper. Lingyi Yang assisted me with the curation of the MIMIC-IV database at the beginning, and from that point on, ideas and direction of the paper was developed jointly by the three of us. Implementation and experiments were handled by myself, I completed the main draft of the paper which was improved with heavy edits by Patrick and Lingyi.

● **Neural RDEs** - The core idea, using the log-ODE method, was thought up by Cristopher Salvi. We then worked jointly to develop the idea and understand where it fitted in the landscape of Neural DEs and time series methods. James Foster represented a continual source of knowledge on rough methodologies and we owe him considerable thanks to the many MS teams calls he endured during the course of this paper. I handled the implementation and experiments. The first draft was completed by me with edits made by everyone. The first abstract section giving a background to rough paths was done by James Foster.

● **Generalised signature method** - The three of us independently had the idea of doing something similar, so we decided to pool ideas and do one together. Ideas were developed jointly, first draft was completed by Patrick and Adeline, edits were made by everyone, I handled the implementation and experiments. The open source sktime software implementation was also developed by me.

● ● **Sepsis papers** - The work in these papers was completed by me, with Andrey acting as a co-supervisor with Terry. The remainder of the authors aided in providing edits to the two write-ups. I would like to add particular thanks to Andrey, without his enthusiasm and encouragement these papers would not have existed at all.

Terry Lyons acted as lead supervisor for all of these papers. He was extremely influential in the development and direction of each and every paper, I only apologise I could only help to follow this very small subset of his proposed ideas through the course of this PhD.

Below we also give two miscellaneous papers that were completed as part of this PhD but are somewhat tangential to the narrative of the thesis so have been left out of the final draft.

● Kidger, P., **Morrill, J.**, and Lyons, T. (2020b). "Generalised interpretable shapelets for irregular time series." *arXiv preprint arXiv:2005.13948.*

● **Morrill, J.**, Qirko, K., Kelly, J., Ambrosy, A., Toro, B., Smith, T., Wysham, N., Fudim, M., and Swaminathan, S. (2021b). "A machine learning methodology for identification and triage of heart failure exacerbations." *Journal of Cardiovascular Translational Research, pages 1–13.*

Part I

Introduction and Preliminaries

Chapter 1

Introduction

The primary aim of this thesis is to demonstrate that the union of rough path theory and machine learning results in powerful models that can be used to solve a range of real-world problems.

Rough path theory and machine learning at first glance may appear to be two disparate branches of research with little common ground. Rough path theory is a branch of pure mathematics that provides us with language to model evolving systems and describe how they interact with one another. Machine learning on the other hand, is a tool that can be used to perform high-capacity function approximation from data, generally requiring large amounts of computational power.

When taken in combination, rough path theory enables us to write down the formulation of a general system evolving under some external influence, and machine learning can then be utilised to fit the as-yet-unknown functions to real-world observations.

When we write *streamed data*, we are employing this as a general, abstract term to describe any ordered sequential data that can affect a system – we defer a formal mathematical definition to Chapter 2. Some examples of streamed data are:

- A piece of handwritten text drawn on a tablet.
- The ordered collection of words in a book.
- A video of a person moving.
- The evolution of emotions of a person during the course of a conversation.
- A changing stock price.
- Vitals signs and laboratory test results of a patient in an ICU.

We give a plot of some such examples in Figure 1.0.1.

These all fit our definition of *streamed data* since they can all have measurable, evolving, effects on things. Again, we leaving the idea of 'an effect' to be highly general

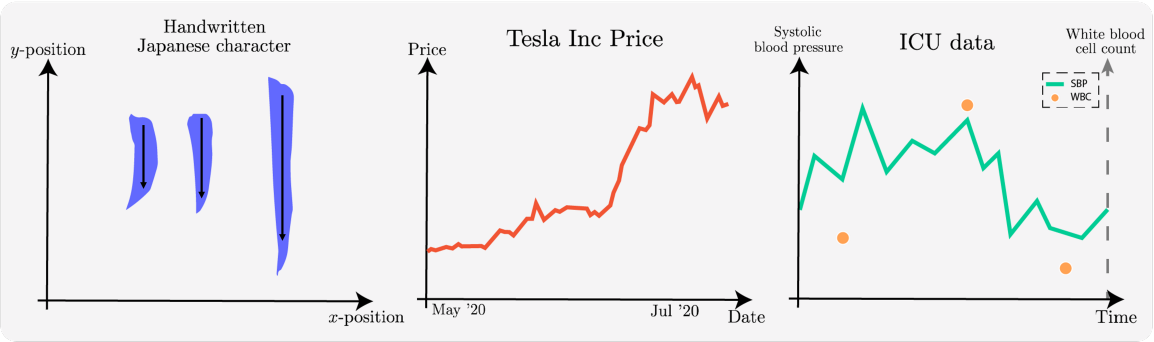


Figure 1.0.1 – Some examples of streamed data. **Left:** A Japanese character sketched on a phone. The data consists of the x, y locations of each of three strokes. **Middle:** a month of Tesla stock prices. **Right:** ICU patient data with frequently observed systolic blood pressure (SBP) and infrequently observed white blood cell (WBC) levels.

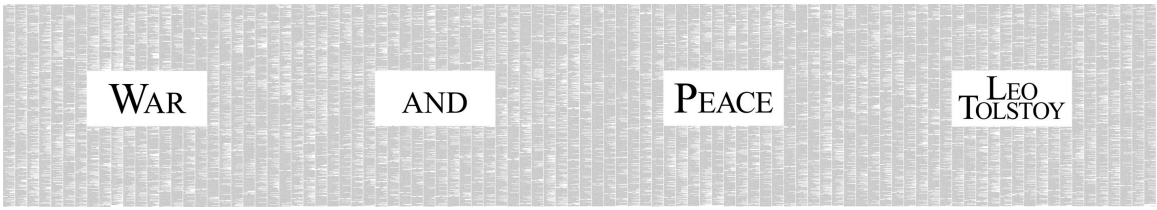


Figure 1.0.2 – The full text of war and peace. This represents a slightly different form of streamed data, but a no less valid one.

on purpose, as theoretically, this effect really can be almost anything. For example, differing streams of handwritten text can change what character is perceived by the person reading it; the order of words in a book changes the readers mood as they read it; and a changing stock price changes the value of ones portfolio.

Rough path theory provides us with a framework for answering questions about streamed data, and their effects, aiming at this generality. At the heart of the mathematics lies the controlled differential equation (CDE)

$$\mathrm{d}z_t = f(z_t)\mathrm{d}X_t, \quad z_{t_0} = \zeta. \tag{1.1}$$

Here $X_t \in \mathbb{R}^d$ and represents the streamed data under consideration – known as the control – $z_t \in \mathbb{R}^v$ is the response of the system, $\zeta \in \mathbb{R}^v$ is an initial condition, and $f \colon \mathbb{R}^v \to \mathbb{R}^{v \times d}$ determines how the response z_t behaves under the influence of X_t. "$f(z_t)\mathrm{d}X_t$" can then be seen to denote a matrix-vector product.

For those unfamiliar with CDEs, it may help to consider the setting where the control path X_t is differentiable. In this instance, the equation can be written

$$\mathrm{d}z_t = f(z_t)\frac{\mathrm{d}X_t}{\mathrm{d}t}\mathrm{d}t, \quad z_{t_0} = \zeta, \tag{1.2}$$

where "$f(z_t)\frac{\mathrm{d}X_t}{\mathrm{d}t}$" is now the matrix-vector product. The response z_t is modified via f in the direction of X_t; hence, we see why this is called a "controlled" differential

equation, since the response is "controlled" by the behaviour of the streamed data X_t.

The CDE is our primary model of consideration when learning systems driven by streamed data. It will be introduced in more detail in 2, but suffice it to say that this relatively simple equation is also a powerful one for modelling functions on streams. In particular, it is known that controlled differential equations are universal approximators on streamed data (Liao et al., 2019) – any continuous function on streamed data can be uniformly approximated on compact sets by a CDE.

To break this down more explicitly, suppose that we are tracking patients in an ICU and wish to model the likelihood that this person has sepsis. We let X_t represent everything we know about the patient up to time t, this can include their vitals signs, laboratory test results, and demographic information. Let z_t denote the probability that the patient under consideration has sepsis given their information up to time t (if a binary indicator is desired, this can be achieved in the standard way via $y_t = \mathbb{1}_{\text{sigmoid}(z_t)>0.5}$). By the universal approximation theorem for CDEs, if there exists some relationship between X_t and z_t (potentially highly nonlinear), then the relationship can be approximated arbitrarily well by a CDE.

The process for the remainder of this thesis will in general be something like this:

1. Examine an evolving system that evolves under the influence of some changing data stream.
2. Model the system as a CDE, with X_t being this observed data stream.
3. Use the tools from rough path theory to write down the solution to the system.
4. Learn any unknown functions that arise via machine learning methods.

The result of this is then a CDE system that maps the underlying data stream onto something that, if all things go to plan, accurately models our effect of interest.

Throughout the remainder of this thesis we will see a range of different methods, models, and applications, that all broadly follow the structure listed above. It is our goal to persuade the reader that learning functions on streamed data through the language of CDEs and rough path theory, powered by machine learning, produces models and results that are both interesting, and useful.

1.1 Plan

The remainder of this thesis will be presented in almost exactly the wrong chronological order. In reality, the chapters were (approximately) completed in the order $8 \to 7 \to 6 \to 3 \to 5 \to 4$. Chapters 7 and 8 contain some applied work that utilise existing tools in novel ways to real-world datasets – they most interesting for anyone who is directly interested in the prediction tasks themselves, these being the prediction of stress and sepsis. Chapter 6 is somewhat of a review article, summarising a

lot of the previous contributions that had been made in the rough path theory and machine learning space. The earlier chapters, Chapters 3 to 5, document a novel methodology

Going into a bit more detail, the chapters are grouped into two parts: Chapters 3 to 5 and Chapters 6 to 8 – with an extra part being this introduction and the preliminaries in Chapter 2 if we count that.

Chapters 3 to 5 covers our more recent work developing *Neural Controlled Differential Equations* (Neural CDEs). These are continuous time models that extend existing Neural ODE ideas but use the CDE (as in Eq. (1.1)) as our differential equation of choice, resulting in a more general model that works on time series due to the X_t term. We demonstrate that they have a number of real, practical benefits over many existing time series models, such as the fact they apply in a very natural way to data that has been irregularly sampled.

Chapters 6 to 8 outline our work on *the signature method*. The signature is a transformation central to the study of CDEs and their solutions (it will be introduced formally in Chapter 2). In brief, the signature applied to a stream of data X_t returns a vector of real-valued features that are most relevant for solving a CDE. As mentioned, Chapter 6 is partly a review article, but also provides novel insights into how all the proposed variations can be classified, and ends by proposing a high-performance canonical pipeline that provides an excellent starting point from which to experiment. Finally, Chapters 7 and 8 follow on naturally as they represent individual applications of the generalised approach devised in Chapter 6.

Chapter 2

Preliminaries

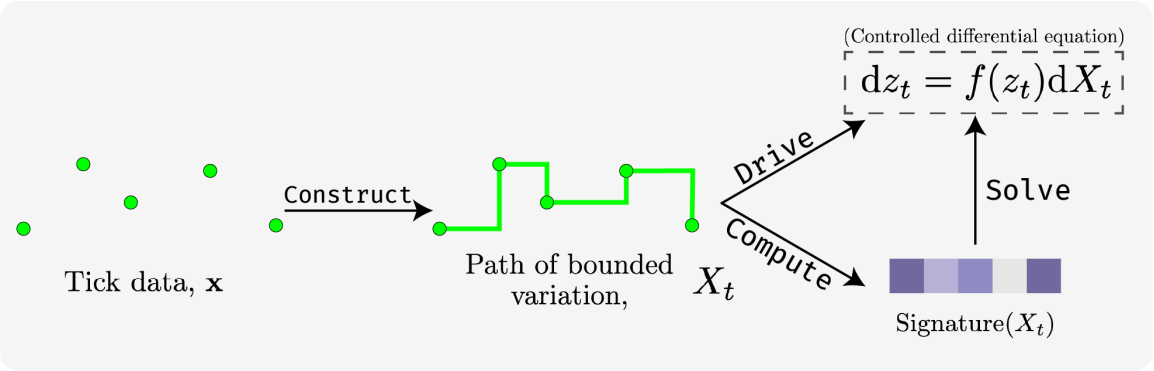


Figure 2.0.1 – A (very) high level overview of the aims of this chapter. Raw tick data is converted to a continuous path of bounded variation that can be used to drive a controlled differential equation (CDE) (top right). The solution to the equation can be estimated as a linear functional of the signature of the path – an vector of real values that describe how to update a CDE (to be introduced formally in this chapter) – making signature coefficients a natural basis in which to express the solutions to CDEs.

2.1 Controlled differential equations

Consider the following ordinary differential equation (ODE)

$$\mathrm{d}z_t = f(z_t)\mathrm{d}t, \quad z(t_0) = \zeta. \tag{2.1}$$

Here $f\colon \mathbb{R}^v \to \mathbb{R}^v$ is a vector field, $z_t \in \mathbb{R}^v$ represents the solution (or response), and $\zeta \in \mathbb{R}^v$ is an initial condition. The solution z_t is updated from its initial value ζ at t_0 as we vary t, in accordance with the behaviour defined by the vector field f.

Often, our system of interest evolves under the influence of significantly more complex behaviour than incremental changes in a single variable, in this case t. For example, suppose we are interested in modelling patient health in an ICU, where we can observe

various characteristics about said patient such as their vitals signs. In such cases, it is appropriate to extend the ODE to something more general that can handle multi-dimensional driving signals.

Definition 2.1.1 (Controlled differential equation (CDE))**.** *Let $X_t \in \mathbb{R}^d$ be a continuous path of bounded variation, $z_t \in \mathbb{R}^v$, $f\colon \mathbb{R}^v \to \mathbb{R}^{v\times d}$, and $\zeta \in \mathbb{R}^v$. Suppose we have that*

$$\mathrm{d}z_t = f(z_t)\mathrm{d}X_t, \quad z(t_0) = \zeta. \tag{2.2}$$

We call the solution z_t of Equation (2.2) the response of the system f to the control X_t from the initial condition ζ. Note that in general, $v > 1$, and $f(z_t)\mathrm{d}X_t$ denotes matrix-vector multiplication.

The CDE is an important generalisation of the ODE stated in Equation (2.1), with the modification $\mathrm{d}t \to \mathrm{d}X_t$ allowing z_t to evolve in multiple directions according to the behaviour of X. The CDE is more general since it allows the equation to be driven by signals that exist in multiple dimensions. If $X_t = t$, then we reproduce the ODE from Equation (2.1) which shows that it is indeed a generalisation.

Equation (2.2) is sufficiently general to model our ICU patients in a way that simultaneously incorporates the vitals signs, laboratory data, and demographic information. Further, the universality property of CDEs – that is, linear functionals of CDEs being dense in the space of continuous functions on a compact set of controls (Lyons, 1998) – shows that the model is also theoretically rich enough to approximate the functional relationship between the input data and the response variable[1].

Supposing that X_t records changing patient observables, the CDE formulation enables us to model evolving functions of the patient's state; for example, a patient's sepsis risk can be modelled via a CDE with hidden state driven by X_t.

2.1.1 Mathematical formalism

To give more concreteness to the classes of controls we can consider, we introduce the notion of a path, a control path, and a path of bounded variation.

Definition 2.1.2 (Continuous path)**.** *If $X : [a, b] \to \mathbb{R}^d$ and each co-ordinate $(X^1, \ldots, X^d)$ is a real-valued continuous mapping $X^i : [a, b] \to \mathbb{R}$, then we call X_t a continuous path.*

Definition 2.1.3 (Control path)**.** *A control path X_t is a geometric rough path (as in (Lyons, 1998)) which we emphasise can be used to drive CDEs.*

In this thesis, the paths we are interested in are specifically those of bounded variation.

[1]Like all universality properties that arise in the field of machine learning, in the real-world this comes with the rather sizeable assumption of infinite data, memory, and time (or algorithm efficiency). In reality, the universality property is just hinting that this theoretically has the desired flexibility for modelling functions on streams, a theory that can only be proven to be useful through empirical evaluation – this evaluation will come in the following chapters.

Definition 2.1.4 (Path of bounded variation (path of finite length))**.** *A path* $X : [a,b] \to \mathbb{R}^d$ *is said to have bounded variation if the quantity*

$$V_a^b(X) = \sup_{\mathcal{D}\in[a,b]} \sum_{k=0}^{n-1} \left\| X_{t_{k+1}} - X_{t_k} \right\|, \tag{2.3}$$

is finite, with the supremum being taken over all finite partitions $\mathcal{D}$ *of the interval* $[a,b]$*.*

Assuming the vector fields are Lipschitz, Equation (2.2) is well-defined for controls that are smooth paths, and more generally for continuous paths of bounded variation.

The CDE contains a natural time reparameterisation invariance property.

Remark 2.1.1 (Reparameterisation invariance)**.** *Let* z_t *be the solution of the following CDE driven by* X_t

$$\mathrm{d}z_t = f(z_t)\mathrm{d}X_t, \quad z_{t_0} = \zeta. \tag{2.4}$$

Suppose further that $\phi : \mathbb{R} \to \mathbb{R}$ *is smooth, then*

$$\mathrm{d}z_{\phi(t)} = f(z_{\phi(t)})\mathrm{d}X_{\phi(t)}, \quad z_{\phi(t_0)} = \zeta. \tag{2.5}$$

This means that if we change the parameterisation (or speed) at which we traverse the control X, then we only change the speed at which the solution is traversed, not the solution itself.

When data arrives discretely (as in generally the case in the real-world) observations can be considered as impulses (or jumps) in our system. CDEs are undefined at jumps, however there is a canonical way in which they can be dealt with considering Remark 2.1.1.

Remark 2.1.2 (Dealing with jumps)**.** *Suppose we have a jump in* X *at* t_0*. We can introduce a reparameterisation* ϕ *that incorporates "virtual time" between* t_0^- *and* t_0^+ *over which we linearly interpolate* X *and solve*

$$\mathrm{d}z_{\phi(t)} = f(z_{\phi(t)})\mathrm{d}X_{\phi(t)}, \quad z_{\phi(t_0)} = \zeta. \tag{2.6}$$

We can then speed up the movement over the virtual time and return to the original parameterisation following the jump.

In the following sections we will see a more concrete example of how this would be achieved in the context of a real discrete data process.

2.2 Tick data

Whilst paths of bounded variation are suitable to drive CDEs, we cannot directly observe them in the real-world. Much real world data is discrete, for example if we

are monitoring traffic to a website we see hits only at discrete time points. Even with data that is truly continuous, such as the evolving blood pressure of a patient, we are restricted to the frequency of our measurement implements which, however good, will always be discrete at some level of granularity.

This section focuses solely on data representation, prior to modelling and choices for f.

2.2.1 Types of tick data

We thus introduce the idea of *tick data*, which we use to represent data obtained from real-world processes.

Definition 2.2.1 (Tick data)**.** *Let $d \in \mathbb{N}$ and $x_i \in (\mathbb{R} \cup \{*\})^d$, where $*$ is used to represent missing data. We denote the space of tick data to be*

$$\mathcal{S}(\mathbb{R}^d) = \{(x_0, \dots, x_n) \mid n \in \mathbb{N},\ n \geq 1\}, \tag{2.7}$$

and denote elements of $\mathcal{S}(\mathbb{R}^d)$ by a lowercase bold letter ($\mathbf{x}$ will be seen most frequently).

Keeping open the possibility for features (or channels) to be missing is important. Often when observing real-world data we do not see all features simultaneously; for example, when patient laboratory tests are returned in an ICU we only see those that were requested by the doctor, with other information being missing.

In many cases, the observation times themselves are valuable information. In such cases it is sensible to include the observation time stamps as features. When tick data includes the observation times as a one of the feature dimensions we call the tick data a *time series*.

Definition 2.2.2 (Time series)**.** *Let $\mathbf{x} \in \mathcal{S}(\mathbb{R}^d)$ be some tick data and let $t_i \in \mathbb{R}$ with $t_0 < \ldots < t_n$ denote the observation times of the observations $\mathbf{x}$. If $(t_0, \dots, t_n)$ is one of the components in $\mathbf{x}$, then we call $\mathbf{x}$ a time series. Typically the times will be included as the first component of $\mathbf{x}$.*

A time series is a subset of tick data, however it is so often useful to include the time stamps with the observation vector that we include this additional definition. We will sometimes denote the set of timestamps with $\mathbf{t} = (t_0, \dots, t_n)$.

We use the terms 'components', 'channels', and 'features', interchangeably to refer to the the dimensions of the input X_t. From hereon, we will tend to use features or channels as they are more commonly used in the machine learning community.

We now give a few additional definitions that we will make use of frequently in later chapters.

Definition 2.2.3 (Fully observed time series)**.** *If $\mathbf{x}$ is a time series, and no values are missing (take the value '$*$'), then we call the data fully observed.*

Definition 2.2.4 (Partially observed time series)**.** *If* $\mathbf{x}$ *is a time series, and at least one value is missing, then we call the stream partially observed.*

This definition covers two common scenarios involving real-world data:

1. A channel was expected to update, but due to some measurement failing or otherwise, did not.
2. A channel was not expected to update, but due to the update of some other channel, results in a missing value. For example, the second channel may be measured at a higher frequency than the first.

Definition 2.2.5 (Regular time series)**.** *Suppose* $\mathbf{x}$ *is a time series and the time stamps* t_i *are such that* $t_{i+1} - t_i = t_{j+1} - t_j \quad \forall i, j \in \{1, \dots, n\}$*. Then we call* $\mathbf{x}$ *a regular time series.*

Definition 2.2.6 (Irregular time series)**.** *Suppose* $\mathbf{x}$ *is a time series and the time stamps* t_i *are such that there is at least one* $i, j \in \{1, \dots, n\}$ *such that* $t_{i+1} - t_i \neq t_{j+1} - t_j$*. Then we call* $\mathbf{x}$ *an irregular time series.*

The 'fully/partially observed' definitions apply equally to tick data, however we will almost exclusively use these definitions with respect to time series, hence the reasons for the definitions being given on time series'.

Deep learning time series models, such as the RNN, tend to struggle with data that is either 'partially observed' or 'irregular'. The models fare significantly better on fully observed data that occurs at regular intervals. You may notice however, that the terms 'partially observed' or 'irregular' are properties of discrete observations, having no natural analogue for signals that are continuous across all channels. This can result in significantly easier handling of data that contains such irregularities.

The missingness of data is often highly valuable information that should be fed into a model; see for example Che et al. (2018). Returning to the ICU example, variables that require laboratory are sparsely observed, since they are only returned upon request of a doctor. Monitoring the rate at which certain tests are observed is therefore likely to hold important information, as it gives an indication into the doctors thought processes. For this reason we introduce the idea of an *observational frequency vector.*

Definition 2.2.7 (Observational frequency)**.** *Supposing* $\mathbf{x} = \big((x_0, c_0), \dots, (x_n, c_n)\big)$ *is a such that* $x_i \in (\mathbb{R} \cup \{*\})^v$ *and* $c_{i,j} = \sum_0^i \mathbb{1}_{\{x_{i,j} \neq *\}}$*. Then we call the* c_i*'s the observational frequencies and we say that* $\mathbf{x}$ *is augmented with observational frequencies.*

Here $\mathbb{1}$ is the indicator function and $\sum_0^i \mathbb{1}_{\{x_{i,j} \neq *\}}$ returns a count of the number of times the feature has been observed, up until the current observation.

2.2.2 Bounded variation paths from time series

We now have a clearer definition for the type of data that we are likely to encounter in real-world time series problems; however, time series as defined in Definition 2.2.2 does

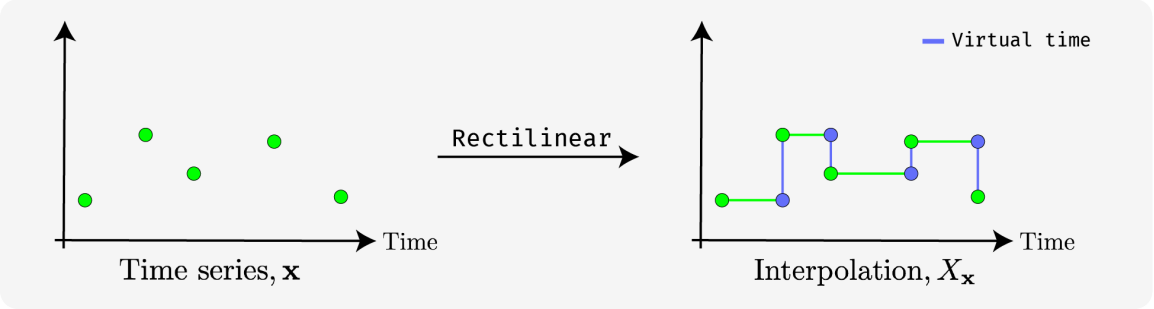


Figure 2.2.1 – Example of the rectilinear interpolation scheme. We introduce a parameterisation of the data that increments by 1 along every green line and every blue line. After an observation (from t_i^+ to t_{i+1}^-, green lines) we increase time as normal, at an observation (from t_i^- to t_i^+, blue line) time is held fixed but our underlying parameterisation is still changing continuously. We sometimes say that we have introduced a "virtual time" in the parameterisation along the blue line.

not yet define a control path that can be used to drive a CDE as in Equation (2.2). Thankfully, it is straightforward to produce a path of bounded variation from tick data, for example, a simple linear interpolation between the points would suffice.

The general procedure will be to construct a continuous path of bounded variation, $X_{\mathbf{x}} : [a, b] \to \mathbb{R}^d$, from tick data $\mathbf{x}$; the subscript being used to denote dependence on the tick data $\mathbf{x}$. We deliberately avoid going into details about how this is done here as much time will be devoted in the best way to do this in 4, however, we provide a natural example to demonstrate that any time series can also be thought of as a continuous path of bounded variation.

Definition 2.2.8 (Rectilinear interpolation)**.** *Let* $\mathbf{x} = \big((t_0, x_0), (t_1, x_1), \dots (t_n, x_n)\big)$ *be some time series. Let* $\widetilde{x}_i$ *denote the forward fill of* x_i *(that is,* x_i *with any missing values filled by the most recent observed value). If*

$$\widetilde{\mathbf{x}} = \big((t_0, \widetilde{x}_0), (t_1, \widetilde{x}_0), (t_1, \widetilde{x}_1), \dots (t_{n-1}, \widetilde{x}_{n-1}), (t_n, \widetilde{x}_{n-1}), (t_n, \widetilde{x}_n)\big), \tag{2.8}$$

and $X_{\widetilde{\mathbf{x}}} : [0, 2n] \to \mathbb{R}^d$ *denotes the linear interpolation of* $\widetilde{\mathbf{x}}$ *such that* $X_{\widetilde{\mathbf{x}}}(2i) = (t_i, \widetilde{x}_i)$ *and* $X_{\widetilde{\mathbf{x}}}(2i + 1) = (t_i, \widetilde{x}_{i-1})$ *for* $i \in \{1, \dots, n\}$*, then we call* $X_{\widetilde{\mathbf{x}}}$ *the rectilinear interpolation of* $\mathbf{x}$*.*

We refer the reader to Figure 2.2.1 from which this definition is significantly easier to understand. The figure depicts how one can construct a rectilinear interpolation from discrete time series data. The crucial point to be aware of here is that **this path is not parameterised by time**; instead, we have introduced a new parameterisation along which we alternate between varying time, and varying the feature dimension.

To show this more explicitly, note from definition 2.2.8 that our parameterisation runs from 0 to $2n$. Time is increased from $2i \to 2i + 1$ and the features are linearly updated ($x_i \to x_{i+1}$) over $2i + 1 \to 2i + 2$. The locations of the first few green and

blue circles from the figure are

$$\begin{aligned} X_{\mathbf{x}}(0) &= (t_0, x_0) \\ X_{\mathbf{x}}(1) &= (t_1, x_0) \\ X_{\mathbf{x}}(2) &= (t_1, x_1) \\ X_{\mathbf{x}}(3) &= (t_2, x_1) \\ X_{\mathbf{x}}(4) &= (t_2, x_2) \\ &\dots \end{aligned}$$

The fact we can jump from $(t_1, x_0) \to (t_1, x_1)$ is allowed due to the introduction of our new parameterisation along $[0, 2n]$. The result is a continuous path that has piecewise continuous derivatives with respect to the underlying parameterisation. There are discontinuities in the derivatives at $X_{\mathbf{x}}(i)$ $\forall i \in \mathbb{N}$, but as we will see in Chapter 4, this does not pose a problem for our applications.

Rectilinear interpolation results in a continuous embedding that faithfully represents the underlying discrete measurements. We say faithful since it represents our observations in reality; time is increased between measurements, and measurements are jumped instantaneously (from the standpoint of time) when we see a new observation.

There are of course many ways we can convert a time series onto a path of bounded variation, and this will indeed be the topic of much discussion in this thesis. This section merely aims to convince the reader that the fact data in the real-world is observed discretely does not pose an issue for modelling such data with CDEs that require control paths that operate in continuous time.

2.3 Rough paths

Rough path theory, introduced by Lyons (1998), provides a complete description for the solution of CDEs driven by highly irregular and oscillatory signals. It gives deterministic meaning to the CDE (as in Eq. (2.2)) when X_t has the roughness of a Brownian motion, and for paths of greater irregularity still.

The theory of rough paths diverts from the traditional calculus of Newton and Riemann in its consideration of the path X_t. Previous focus was placed on the consideration of the control path at particular instances in time, which breaks down as the path becomes increasingly complex or oscillatory. Instead, rough path theory considers the path over intervals and gives a description of the path over the interval that is sufficient to describe its effect on the response, z_t, at the end of the interval.

This description is called the signature of the path and will be introduced in the following section, but at a high-level, the signature picks out the information that is required to estimate the movement of the response over the interval, without the need to consider the fine-scale behaviour of the control path. Rough path theory can

be thought of as the study of converting $X_{[r,s)}$ onto actionable information that can be used to predict how z_s evolves from z_r.

In this thesis, we will be sticking exclusively to paths of bounded variation. Paths of bounded variation are dense in the metrics of rough paths; as such, the theory of rough paths applies to paths of bounded variation.[2]. We will not be delving far into the rich mathematical landscape of rough paths; however, we think it is important to highlight that the novelties that appear in this work are almost exclusively based on the ideas that arise from rough path theory.

We direct the interested reader to Lyons et al. (2007b) for an introduction, and to Friz and Hairer (2020) or Friz and Victoir (2010) for a more comprehensive textbook.

2.4 The signature

The signature is an object central to the study of rough path theory and controlled differential equations. The signature consumes a path and returns a collection of real values each of which contains information about the structure of the path; it can be useful to think of these as analogous to summary statistics about a path, only these are the summary statistics that are most relevant for solving a CDE.

To define the signature, we begin by defining a *k-fold iterated integral*.

Definition 2.4.1 (k-fold iterated integral)**.** *Let $a, b \in \mathbb{R}$, and $X : [a, b] \to \mathbb{R}^d$ be a continuous smooth path. We define the k-fold iterated integral of X along the indices $i_1, \dots, i_k$ as*

$$S_{a,t}^{i_1,\dots,i_k}(X) = \int_{a<t_k<t} \dots \int_{a<t_1<t_2} \mathrm{d}X_{t_1}^{i_1} \dots \mathrm{d}X_{t_k}^{i_k}, \tag{2.9}$$

We are now in a position to give a definition to what it means to be the signature of a path X.

Definition 2.4.2 (The signature)**.** *The signature of a path of bounded variation is the infinite sequence of real values*

$$S_{a,b}(X) = \big(1, S_{a,b}^1(X), \dots, S_{a,b}^d(X), S_{a,b}^{1,1}(X), S_{a,b}^{1,2}(X), \dots\big), \tag{2.10}$$

where $S_{a,b}^{i_1,\dots,i_k}(X)$ are the k-fold iterated integrals of X as in Definition 2.4.1.

Example 2.4.1 (Signature computation)**.** *Consider the path $X_t = (t, \sin(t))$ for*

[2] A common misconception is that rough path theory applies only to these highly irregular "rough paths"; in reality, rough path theory does indeed provide us with the language to describe solutions for these irregular paths, but absolutely still applies to paths of higher regularity.

$t \in (0, \pi)$. *We have up to depth 2*

$$
\begin{aligned}
S^1_{0,\pi}(X_t) &= \int_0^\pi \mathrm{d}t_1 = \pi \\
S^2_{0,\pi}(X_t) &= \int_0^\pi \mathrm{d}(\sin(t_1)) = 0 \\
S^{1,1}_{0,\pi}(X_t) &= \int_0^\pi \int_0^{t_2} \mathrm{d}t_1 \mathrm{d}t_2 = \int_0^\pi t_2 \mathrm{d}t_2 = \frac{\pi^2}{2} \\
S^{1,2}_{0,\pi}(X_t) &= \int_0^\pi \int_0^{t_2} \mathrm{d}t_1 \mathrm{d}(\sin(t_2)) = \int_0^\pi t_2 \mathrm{d}(\sin(t_2)) = \int_0^\pi t_2 \cos(t_2) \mathrm{d}t_2 = 2 \\
S^{2,1}_{0,\pi}(X_t) &= \int_0^\pi \int_0^{t_2} \mathrm{d}(\sin(t_1)) \mathrm{d}t_2 = \int_0^\pi \sin(t_2) \mathrm{d}t_2 = 2 \\
S^{2,2}_{0,\pi}(X_t) &= \int_0^\pi \int_0^{t_2} \mathrm{d}(\sin(t_1)) \mathrm{d}(\sin(t_2)) = \int_0^\pi \sin(t_2) \cos(t_2) \mathrm{d}t_2 = 0 \\
S^{1,1,1}_{0,\pi}(X_t) &= \int_0^\pi \int_0^{t_3} \int_0^{t_2} \mathrm{d}t_1 \mathrm{d}t_2 \mathrm{d}t_3 = \dots \\
&\dots
\end{aligned}
$$

Collecting the terms we have $S(X_t) = (1, \pi, 0, \pi^2/2, 2, 2, 2, 0, \dots)$.

We also provide the following, more succinct definition.

Definition 2.4.3 (The signature, alternative notation)**.** *Let* $a, b \in \mathbb{R}$*, and* $X : [a, b] \to \mathbb{R}^d$ *be a continuous smooth path.*

$$
S_{a,b}(X) = \left(\left(\int_{a<t_k<t} \dots \int_{a<t_1<t_2} \mathrm{d}X^{i_1}_{t_1} \dots \mathrm{d}X^{i_k}_{t_k} \right)_{(i_1,\dots,i_k)\in\mathcal{I}_k} \right)_{k\geq 0} \tag{2.11}
$$

where $\mathcal{I}_k$ *denotes all subsets of length* k *of the indices* $(1, \dots, d)$.

From this definition we can see how terms are collected into 'depths' that depend on the number of integrals that were performed to produce the output. For example collecting example 2.4.1 up to depth 2 we have $\big((1), (\pi, 0), (\pi^2/2, 2, 2, 0)\big)$.

As has already been alluded, signatures are central to the study of CDEs. One clear way to see this is by performing a Taylor expansion of a generic CDE, and we see that the Taylor coefficients are precisely the signature features.

Example 2.4.2 (Taylor expansion of CDEs)**.** *To generate a solution to the CDE equation, one obvious tactic is to apply a Taylor expansion. By converting Equa-*

tion (2.2) *to its integral form, expanding, and neglecting higher order terms we find*

$$z_b = z_a + \int_a^b f(z_{t_1})\mathrm{d}X_{t_1}, \tag{2.12}$$

$$= z_a + \int_a^b f(z_a) + D_f(z_a)(z_{t_1} - z_a)\mathrm{d}X_{t_1} + \dots, \tag{2.13}$$

$$= z_a + f(z_a)\int_a^b \mathrm{d}X_{t_1} + D_f(z_a)\int_a^b (z_{t_1} - z_a)\mathrm{d}X_{t_1} + \dots, \tag{2.14}$$

where D_f is that Jacobian of the function f. Now substituting $z_s - z_a$ from the first line, and again expanding and ignoring higher order terms we find that

$$= z_a + f(z_a)\int_a^b \mathrm{d}X_{t_1} + D_f(z_a)\int_a^b \int_a^{t_2} f(z_{t_1})\mathrm{d}X_{t_1}\mathrm{d}X_{t_2} + \dots, \tag{2.15}$$

$$= z_a + f(z_a)\int_a^b \mathrm{d}X_{t_1} + D_f(z_a)f(z_a)\int_a^b \int_a^{t_2} \mathrm{d}X_{t_1}\mathrm{d}X_{t_2} + \dots, \tag{2.16}$$

More explicitly, for the i'th co-ordinate of z_t up to depth 2 we have

$$z_b^i = z_a^i + \sum_{j=1}^d f(z_a)_{i,j}\int_b^a \mathrm{d}X_{t_1}^j + \sum_{j=1}^d\sum_{k=1}^d \big(D_f(z_a)f(z_a)\big)_{i,j,k}\int_a^b \int_a^{t_2} \mathrm{d}X_{t_1}^j\mathrm{d}X_{t_2}^k \tag{2.17}$$

$$= z_a^i + \sum_{j=1}^d f(z_a)_{i,j}S_{a,b}^{(j)}(X_t) + \sum_{j=1}^d\sum_{k=1}^d \big(D_f(z_a)f(z_a)\big)_{i,j,k}S_{a,b}^{(j,k)}(X_t) + \dots \tag{2.18}$$

We see that the Taylor coefficients are precisely the terms of the signature.

2.4.1 Properties of the signature

Our first fundamental property of the signature is *Chen's identity*, which states that the signature of two concatenated paths is equivalent to the product of their signatures.

Theorem 2.4.1 (Chen's identity; Chen (1954))**.** *Let $X : [a, b] \to \mathbb{R}^d$ and $Y : [b, c] \to \mathbb{R}^d$ be two continuous paths of bounded variation, letting $\star$ denote path concatenation and S the signature transform we have*

$$S(X \star Y) = S(X) \otimes S(Y). \tag{2.19}$$

We see from the citation dating back to 1954 in Chen's identity; Chen (1954) (Theorem 2.4.1), that signatures go way back. Signatures were not discovered alongside rough path theory, rather, rough path theory was what interwove them with analysis to establish a new calculus.

The two most important properties (for us at least) are *uniqueness* and *universal nonlinearity*.

The uniqueness property proves that the signature is a faithful representation of that path; every path has a unique signature (up to some null set that we need not concern ourselves with). The universal nonlinearity property demonstrates the centrality of signatures to the study of CDEs; it tells us that the solution to any (linear or nonlinear) CDE can be written as a linear function on the signature of the control. In this way, the signature can be thought of as unravelling the non-linearities inherent in the CDE equation, and can be used as a natural function basis for modelling the response of CDEs to driving signals.

Theorem 2.4.2 (Uniqueness of the Signature; Hambly and Lyons (2010))**.** *Let $X : [a, b] \to \mathbb{R}^d$ be a continuous path of bounded variation. Then $S(X)$ uniquely determines X up to parameterisation and translation (and an appropriate null set)*[3]*.*

Theorem 2.4.3 (Universal nonlinearity; Lyons (1998); Bonnier et al. (2019))**.** *Let F be a real-valued continuous function on continuous paths of bounded variation in $\mathbb{R}^d$ and let K be a compact set of such paths*[4]*. Let $\epsilon > 0$. Then there exists a linear function L such that for all $X \in K$,*

$$\left|F(X) - L\big(S(\hat{X})\big)\right| < \epsilon \tag{2.20}$$

where $\hat{X} = (t, X) \in \mathbb{R}^{d+1}$ denotes that path X with time included as an extra dimension.

Another important property, especially when considering the signature in machine learning applications, is the *factorial decay* of the signature. As we take the signature to higher depths (more iterated integrals) the size of the terms exhibits factorial decay; this is very important to be aware of in machine learning as often learning algorithms are highly sensitive to feature scale. More time will be devoted to this in Chapter 6

Proposition 2.4.4 (Factorial decay; Lyons (1998))**.** *Let $X : [a, b] \to \mathbb{R}^d$ be a continuous piecewise smooth path. Then*

$$\left\| \int_{a<t_k<t} \dots \int_{a<t_k<t_1} \mathrm{d}X_{t_1} \otimes \dots \otimes \mathrm{d}X_{t_k} \right\| \leq \frac{L^k}{k!} \tag{2.21}$$

where L is the length of the path X and $\|\cdot\|$ is any tensor norm on $(\mathbb{R}^d)^{\otimes k}$.

Finally, we define the notion of the *truncated signature*. The signature is an infinite collection of iterated integrals which, given we are limited by a world of finite resources, we can never expect to compute all of. For this reason we generally compute only the first few iterated integrals and use these features in our models.

[3]Technically this definition should address the issue of the null sets that Hambly and Lyons introduce which are called tree-like pieces. However, we will find that for real-world time series data such pieces are never likely to occur, a simple inclusion of the monotonically increasing time channel for instance always guarantees the path will not be tree-like. We therefore do not concern ourselves with the null set here, though some discussion is given in Chapter 3

[4]The definition of both continuity and compactness depend on the topology of the set of paths. See Lyons (2014a) for details.

Definition 2.4.4 (Depth-N truncated signature)**.** *The truncated signature to depth N is defined as*

$$S^N_{a,b}(X) = \left(\left(\int_{a<t_k<b} \dots \int_{a<t_k<t_1} \mathrm{d}X^{i_1}_{t_1} \otimes \dots \otimes \mathrm{d}X^{i_k}_{t_k} \right)_{(i_1,\dots,i_k)\in\mathcal{I}_k} \right)_{0\le k\le N}$$
$$= \big(1, S^1_{a,b}(X), \dots, S^d_{a,b}(X), \dots, S^{\overbrace{1,\dots,1}^{N \text{ times}}}_{a,b}(X), \dots, S^{\overbrace{d,\dots,d}^{N \text{ times}}}_{a,b}(X)\big).$$

2.4.2 The log-signature

The signature transform has some redundancy. A little algebra shows that

$$S^{1,2}_{a,b}(X) + S^{2,1}_{a,b}(X) = S^1_{a,b}(X) S^2_{a,b}(X), \tag{2.22}$$

and so we already know $S^{1,2}$ if we know three other quantities. This brings us to the introduction of a different but related transform known as *the log-signature*. The log-signature holds the same information as the signature, but without these redundancies; one can think of it as a compressed version of the signature.

As its name suggests, the log-signature can be obtained by taking the natural logarithm of the signature (in a lie theoretic sense (Reizenstein, 2017)). The process is not particularly illuminating but can be seen in Lyons (2014b). For our purposes, it is sufficient to know the log-signature is a compressed version of the signature and the log-signature truncated to depth N is denoted $\mathrm{LogSig}^N_{a,b}(X)$.

It is also worth noting that the log-signature loses the universal nonlinearity property from Universal nonlinearity; Lyons (1998); Bonnier et al. (2019) (Theorem 2.4.3) in favour of this more compressed data format. When utilising the log-signature in machine learning applications it is therefore important to utilise a non-linear model, since a linear model on the log-signature does not have the same guarantees as for the signature.

2.4.3 Signature intuition and examples

Example 2.4.3 (Depth 1 (log-)signature)**.** *The depth 1 (log-)signature describes the path's total displacement vector over the interval. To see this note*

$$S^1_{a,b}(X) = \int_a^b dX_t = \int_a^b \dot{X}_t dt = X_b - X_a = \Delta X_{[a,b]}. \tag{2.23}$$

This can be seen in Figure 2.4.1 by looking at the $\Delta X_1, \Delta X_2$ terms.

Example 2.4.4 (Depth 2 (log-)signature)**.** *In Figure 2.4.1 we give a visualisation of depth 2 terms for the signature and log-signature in two dimensions.*

The $S^{1,2}_{a,b}(X), S^{2,1}_{a,b}(X)$ terms in the signature can be seen as the blue and orange areas of the left hand plot respectively. It is also straightforward to show that the $S^{i,i}_{a,b}(X)$

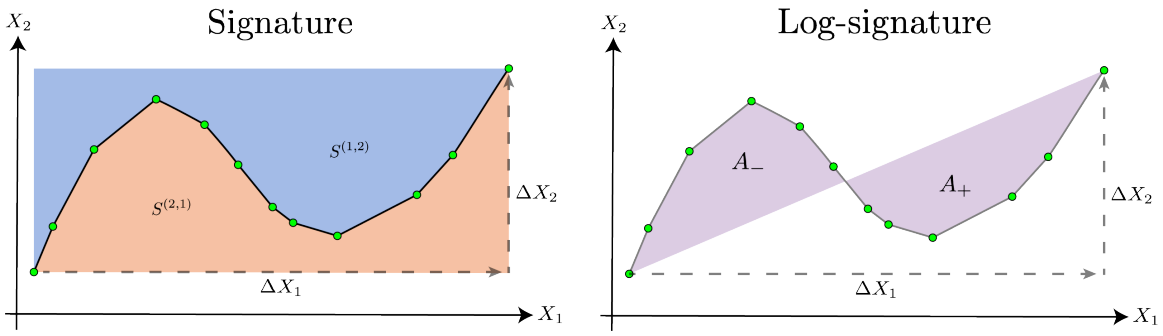


Figure 2.4.1 – Depths 1 and 2 term visualisation for the signature (left) and log-signature (right).

terms is half the square of the increment.

$$S^{i,i}_{1,b}(X) = \int_a^b \int_a^{t_2} \mathrm{d}X_1^i \mathrm{d}X_2^i = \frac{(X_b^i - X_a^i)^2}{2}. \tag{2.24}$$

The log-signature for a two-dimensional has a single term at depth 2 and it is equal to the Lévy area of the path, that is

$$A = A_+ - A_- = \frac{1}{2}\big(S^{1,2}_{a,b}(X) - S^{2,1}_{a,b}(X)\big). \tag{2.25}$$

Higher order signature visualisations become increasingly more difficult, however, in Chapter 8 we give some third order visualisations of the signature that have intuitive meaning with respect to a real data stream.

Part II

Controlled Differential Equations

Chapter 3

Neural Controlled Differential Equations[1]

Neural controlled differential equations (Neural CDEs) are the continuous time limit of RNNs, and are particularly well suited to modelling functions on irregular time series. They are an extension of Neural ODEs that enable the hidden state to be continuously updated based on some incoming observations through utilisation of CDE theory. We demonstrate that the model achieves state of the art performance on a range of benchmarks and metrics.

- *Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural Controlled Differential Equations for Irregular Time Series.* Advances in Neural Information Processing Systems

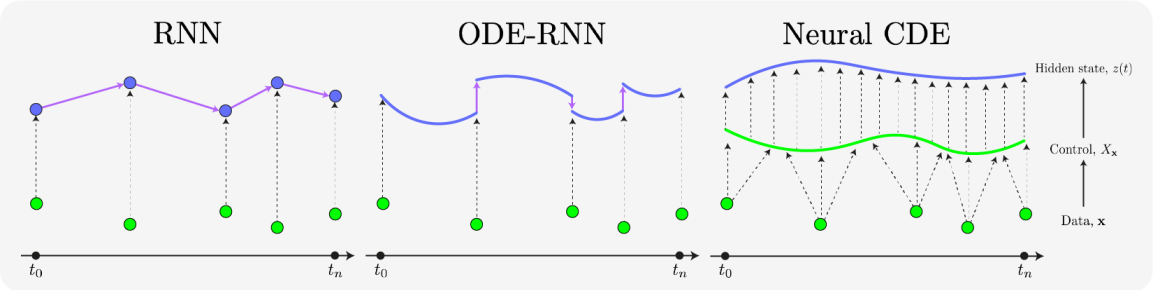


Figure 3.0.1 – Visual comparison on the RNN, ODE-RNN and Neural CDE methods. **Left:** the RNN updates the hidden state only at observation times and is undefined in-between. **Middle:** The ODE-RNN is defined between observations and modifies the hidden state at each observation. **Right:** The Neural CDE model first constructs a continuous time path control $X_{\mathbf{x}}$ resulting in a true continuous time interpretation of the hidden state.

[1]Joint work with Patrick Kidger, Cristopher Salvi, and James Foster

3.1 Introduction

Since their inception, Recurrent Neural Networks (RNNs) have been a dominant modelling paradigm for all types of sequential data modelling. They have, at one time or another, been a crucial component in state-of-the-art models in language modelling (Sutskever et al., 2011; Mikolov et al., 2010; Graves and Jaitly, 2014), EHR data (Choi et al., 2016), ICU prediction (Fagerström et al., 2019), and almost any other area that deals with large quantities of data exhibiting a sequential-like structure. Whilst RNNs have recently found themselves outpaced by transformer-style architectures in certain areas (Vaswani et al., 2017), they still remain an extremely performant in time series applications.

One issue that arises in RNNs with regards to time series applications is that they tend to breakdown when the data is irregularly sampled. Often, practitioners first divide the data into equally sized 'bins' along the time dimension, and perform some aggregation over the intervals (for example a simple mean function) before running the data through an RNN (Fagerström et al., 2019). Doing such preprocessing fundamentally destroys information that may be useful for prediction. With medical data for example, it may be crucial to know whether a laboratory test was ordered because of a rise in blood pressure, or the other way around; this is an example of information that could be obfuscated by equal time binning.

Neural differential equations Chen et al. (2018) represents a promising area of research in this regard. Being differential equations, they can define a model with a latent-state that is defined at all times, making them an appealing model class for modelling irregular time series. Recent examples include Rubanova et al. (2019); Jia and Benson (2019); De Brouwer et al. (2019).

In this chapter, we introduce the Neural Controlled Differential Equation (Neural CDE) model, which can be thought of as a continuous time extension of an RNN. As we will see, using a CDE over an ODE is what will allow our model to have continuous dependence on the underlying data, something a Neural ODE cannot do.

The structure of this chapter is as follows: first, we overview the Neural ODE model, since Neural CDEs can be understood naturally as an extension of this framework; second, we describe the Neural CDE model and outline how it is solved for piecewise differentiable control paths; third, we outline the comparison with the more traditional RNN model, in particular, we demonstrate that an RNN is actually a specific discretisation of the Neural CDE; finally, we give experimental results that establish the Neural CDE as state-of-the-art for time series ODE modelling.

3.2 Neural Ordinary Differential Equations

We being by outlining the Residual Neural Network (ResNet) architecture as introduced by He et al. (2016). Neural ODEs can be viewed as the continuous time limit

of a ResNets and deriving this from the ResNet equation is a natural way in which to introduce the Neural ODE model.

Suppose that we wish to predict some output $y \in \mathbb{R}^w$ from initial data $x_0 \in \mathbb{R}^d$. With the ResNet model we learn a function $f_\theta \colon \mathbb{R}^v \mapsto \mathbb{R}^v$ and linear maps $\ell^1_\theta \colon \mathbb{R}^d \mapsto \mathbb{R}^v, \ell^2_\theta \colon \mathbb{R}^v \mapsto \mathbb{R}^w$ such that

$$y \approx \ell^2_\theta(z_n) \quad \text{where} \quad z_n = z_{n-1} + f_\theta(z_{n-1}) \quad \text{and} \quad z_0 = \ell^1_\theta(x_0). \tag{3.1}$$

This is analogous to a feedforward neural network but for the addition of the z_{n-1} term in the hidden state update; this inclusion is what is called the *residual connection* and therefore why we call it a ResNet. This inclusion allows for improved gradient backpropagation through layers which enables effective training of much deeper networks.

Neural ODEs are not a recent development, they have been around since at least the early 90's as seen in work such as Chu and Shoureshi (1991); Rico-Martinez et al. (1994); González-García et al. (1998). They have seen a revival more recently (Weinan, 2017; Haber and Ruthotto, 2017; Haber et al., 2018; Chen et al., 2018; Chang et al., 2018). In Haber and Ruthotto (2017) the authors show that the ResNet (He et al., 2016) may be derived from an Euler discretisation of a corresponding ODE, and use this to motivate new stable architectures using ideas from ODE theory that improve gradient propagation. In Chang et al. (2018) the authors show how reversible ODE architectures can be used to derive both stable and memory efficient neural network architectures. Finally, Chen et al. (2018) coined the term 'Neural ODEs' with their paper *Neural Ordinary Differential Equations* which captured the imagination of many practitioners and has resulted in an explosion of papers in the field.

We can rewrite the ResNet from Eq. (3.1) as

$$\frac{z_n - z_{n-1}}{\Delta t} = f_\theta(z_{n-1}), \tag{3.2}$$

where $\Delta t = 1$. We can see that this is in precisely the same form as an Euler solve of the following ODE

$$\frac{\mathrm{d}z}{\mathrm{d}t} = f_\theta(z). \tag{3.3}$$

Therefore, by taking the continuous limit of the ResNet, the hidden state update can be achieved via

$$z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z)\mathrm{d}t. \tag{3.4}$$

That is, the propagation of information through layers in a ResNet can actually be considered as a discretisation of the ODE in Equation (3.4).

Neural ODEs provide us with an interface between the fields of machine learning and differential equation modelling. They allow us to combine modern tools from ODE research alongside the high flexibility and powerful function approximation properties from machine learning.

3.2.1 Benefits of neural ODEs

Here we outline some of the main benefits that arise through the combination of ODE modelling with neural networks.

Portability of existing theory By considering the full ODE equation, we can utilise existing theory of ODEs to improve various aspects of the neural architecture. Important examples include the construction of forwards and backwards stable architectures, as well as memory efficient architectures (Haber and Ruthotto, 2017; Haber et al., 2018; Chang et al., 2018).

Memory efficiency Viewing the underlying hidden state propagation equation as an ODE has aided in the construction of memory efficient architectures. A general feedforward neural network requires $\mathcal{O}(HL)$ where H is the size of the hidden state and L is the number of layers. Different methods using ideas from ODE theory have reduced this to $\mathcal{O}(H)$, enabling the training of arbitrarily deep networks.

We give more discussion to this below in Section 3.2.2 as it is an important topic with a number of finer points.

Learning physical systems Much of physics-based modelling is based on ODEs. Neural ODEs allow for modelling to occur without the need for prior information regarding the ODE structure to be defined through expert knowledge, this being due to the high-capacity function approximation properties of neural networks.

Continuous time modelling Being ODEs, they can produce solutions that are defined for continuous time. This is in direct contrast to something such as a ResNet which would only produce updates on a discrete grid and not provide us with the mechanism to make inference at arbitrary points in-between.

Time series Continuous data cannot be irregularly sampled, since there is no analogous concept of sampling rate for a continuous path. Neural CDEs (as we will see in this chapter) operate on continuous paths; as such, modelling of the data can be simplified. This comes with the rather big caveat that we can find a 'faithful' continuous embedding of discretely observed data process, but in cases where we can, the handling of such data can be simplified from a modelling standpoint.

Continuous normalizing flows A class of methods (normalizing flows) for constructing generative models have been shown to be derived from differential equations (Chen et al., 2018; Onken and Ruthotto, 2020).

3.2.2 Backpropagation through neural ODEs

In this section we will outline the two main ways in which people backpropagate through the ODE solves to optimise the parameters of the neural network. These

are

- **Discretise-then-optimise** First discretise the continuous problem and then solve a finite-dimensional optimisation problem on the discretised grid (Haber and Ruthotto, 2017; Chang et al., 2018; Haber et al., 2018).
- **Optimise-then-discretise** Optimise the continuous ODE and discretise the continuous dynamics after training (Chen et al., 2018; Grathwohl et al., 2018; Rackauckas et al., 2019).

There are a number of points to be aware of when choosing the optimisation scheme, we will overview the main ones here but refer the reader to Gholami et al. (2019); Onken and Ruthotto (2020); Kidger (2022) for more details.

3.2.2.1 Discretise-then-optimise

In this case we formulate a discretisation of our ODE and optimise over the discretisation. This is analogous to the standard procedure used in neural network optimisation whereby backpropagation amounts to reverse-mode automatic differentiation through the ODE solves along the same grid that the forward propagation was performed on.

Advantages of this method are

- **Accuracy of gradients** Reverse-mode differentiation provides exact values of the gradients for the discretisation being used. We will see this is in contrast to the optimise-then-discretise approach which only provides us with approximations to gradients in the backwards pass.
- **Speed** Since the grid is specified in advance, no adaptive time-stepping is required and the backpropagation can rely on existing autodifferentiation software that is highly optimised.

Disadvantages are

- **Memory inefficiency** Hidden states from the forward propagation must be held in memory to compute the gradients in the backwards pass. This results in a total memory footprint of $\mathcal{O}(HL)$ where H is the memory of the hidden state, and L the total number of layers.

  However, as we will see below, reversible solvers and be used to reduce this to $\mathcal{O}(H)$. As such, this does not really become a disadvantage provided we are prepared to take some extra care to setup a reversible solver.
- **Grid specification** This method requires specification of a grid, which one could consider akin to another hyperparameter to tune. The finer the grid, the more accurate the solution and the better convergence, but at a cost of increased training times and additional function evaluations; as such, there is a trade-off that must be optimised for.

3.2.2.2 Checkpointing

One way memory inefficiencies can be mitigated is by checkpointing the model (Rojas et al., 2020). This involves storing a subset of the hidden states during the forward pass such that during the backwards pass, the hidden states can be recomputed from the checkpointed hidden states when they are required. The memory cost of this is then $\mathcal{O}(HC)$ where C is the number of checkpointed layers. The more checkpoints we have, the faster the hidden states can be recomputed, but the higher the memory cost.

Checkpointing is also architecturally more challenging as it requires a number of additional steps to perform an update of all the weights in the model, so there is an additional trade-off here to be considered.

3.2.2.3 Reversible architectures

An even neater way to mitigate the issues of memory consumption is to use a reversible architecture (recommended reading: Chang et al. (2018)). An architecture is called reversible is it allows the construction of activations going from the end to the beginning. In such cases, we only need the final (or final few) activations, and we can successively reconstruct the activations as we backpropagate through the network. This results in memory of just $\mathcal{O}(H)$ and removes the need for checkpointing.

Example 3.2.1 (The midpoint network)**.** *In this example we consider solving the ODE via the midpoint approach. That is,*

$$\frac{z_{n+1} - z_{n-1}}{2\Delta t} = f_\theta(z_{t_n}). \tag{3.5}$$

Here we are assuming equal length gaps between observations, but this could easily be modified. Therefore, in the forward pass we have

$$z_{n+1} = z_{n-1} + 2\Delta t f_\theta(z_{t_n}). \tag{3.6}$$

Now provided we store $z_{t_{n+1}}$ and z_{t_n}, we can reconstruct $z_{t_{n-1}}$ via

$$z_{n-1} = z_{n+1} - 2\Delta t f_\theta(z_{t_n}). \tag{3.7}$$

Thus, we only need to store two hidden states at any given time, computing the next hidden state when needed and dropping the z_{n+1} state from memory. This results in just $\mathcal{O}(H)$ memory usage.

Training time using a reversible method will of course increase due to the increased number of forward and backward computations that must be performed to reconstruct the information.

There are many finer points associated with this process, in particular regarding the stability of the methods, we refer the reader to Onken and Ruthotto (2020) for a more in depth discussion.

We highlight that a reversible architecture need not be motivated from an ODE model. (Gomez et al., 2017) for example introduce a reversible architecture for a ResNets that is not explicitly motivated via an ODE. We therefore want to be careful in noting this is not specifically a benefit of ODE modelling, rather, it is an approach from which results from ODE theory are extremely useful.

It is worth noting however that the RevNet model introduced by (Gomez et al., 2017) is not able to be made backwards stable. The only stable and reversible architectures that have been found have all been through considerations of ODE theory.

3.2.2.4 Optimise-then-discretise

This is the approach taken by Chen et al. (2018) whereby the gradients are computed numerically by solving the adjoint equations Pontryagin (1987).

The primary advantage of this model would be that it is memory efficient. The adjoint method means hidden states are recomputed on the backwards pass and thus do not need to be held in memory making this just $\mathcal{O}(H)$ memory, the same as when using reversible solvers.

However, we defer from explicitly stating it as a benefit since really one would require both forwards and backwards stability for this to be an option. Sadly, this is not the case which causes a number of issues that would generally make this approach a bad choice.

There are a number of disadvantages of this approach to be aware of, including

- **Numerical instabilities** For general activation operators, reversibility of the ODE may be unstable due to ill-conditioning of the reverse-mode solver. This leads to inaccuracies of gradients and poor convergence, or even divergence of training.
- **Inaccuracies of gradients** Another way gradient inaccuracies can occur (other than that listed above) is since the forward and backwards pass are not solved over the same grid. In such cases, inaccurate gradients can arise when the solution time steps differ between the solver and the solver tolerance is insufficient.
- **Speed** This method is significantly slower than the discretise then optimise approach. The method uses adaptive-time stepping which results in more calls to the solver.

  For time series we will often have discontinuous gradients with respect to our parameterisation which results in long training times due to the need to resolve the discontinuities. On the other hand, discretise-then-optimise approaches can be used without issue provided we match the timesteps up with the discontinuities in gradient.

Further details and experiments regarding the differences between the two approaches can be found in Gholami et al. (2019); Onken and Ruthotto (2020).

The work in the remainder of this chapter uses the optimise-then-discrete approach. In Chapters 4 and 5 we instead utilise the discretise-then-optimise approach since we found that it resulted in significantly better performance and were able to fit our model in memory.

3.2.3 Key drawback of neural ODEs for time series

The definition of the Neural ODE as stated in Equation (3.4) cannot be straightforwardly applied to time series data. The key issue is that a Neural ODE defines an initial value problem; as such, all information must be encoded into the hidden state z_0 prior to running the Neural ODE model. This is an unnatural way to process sequential data, and almost certainly suboptimal. In particular, if data is being processed in an online fashion, the Neural ODE provides no mechanism with which to update the solution trajectory based on the incoming, unseen data.

3.3 The ODE-RNN model

ODE-RNNs were one of the first successful extensions of neural differential equations to irregular time series data. Introduced by Rubanova et al. (2019) they utilise a neural ODE to propagate the hidden state between observations, then use an RNN to jump the hidden state upon receiving a new observation.

Specifically, between observations they let the hidden state $z \in \mathbb{R}^v$ be the solution to an ODE

$$\widehat{z}_i = \text{ODESolve}(f_\theta, z_{i-1}, (t_{i-1}, t_i)) \tag{3.8}$$

and then at each observation, the hidden state is updated via a standard RNN update

$$z_i = \text{RNNCell}(\widehat{z}_i, x_i). \tag{3.9}$$

The hidden state is, unlike and RNN, defined at every time point that the underlying time series is defined. One of the major downsides of this model is that the RNN requires an interruption of the neural ODE at each observation, which means the model cannot utilise the adjoint backpropagation property – a major upside of neural ODE modelling.

3.4 Neural Controlled Differential Equations

Neural CDEs are an elegant solution to the aforementioned problem with Neural ODEs; that being that Neural ODEs define an initial value problem and have no mechanism for updating their solution trajectory based on incoming data. The idea with Neural CDEs is simple, instead of modelling the data with an ODE and training the vector field with data, model with a CDE instead.

Suppose that we observe a time series as in Definition 2.2.2. To recap, we have $\mathbf{x} = \big((t_0, x_0), \ldots, (t_n, x_n)\big)$ with $t_i \in \mathbb{R}$ denoting the timestamp of the observation vector $x_i \in (\mathbb{R} \cup \{*\})^{d-1}$, where $*$ is used to denote that some information may be missing, and $t_0 < \ldots < t_n$.

Let $X_{\mathbf{x}} \colon [0, n] \to \mathbb{R}^d$ be a (continuous, piecewise differentiable, bounded variation) interpolation such that $X_{\mathbf{x}}(i) = (t_i, x_i)$, where '=' denotes equality up to missing data. Here $X_{\mathbf{x}}$ will be the control path for our Neural CDE equation.

We then let $\ell_\theta^1, \ell_\theta^2$ be as defined in Section 3.2, but now $f_\theta \colon \mathbb{R}^v \mapsto \mathbb{R}^{v \times d}$. We then say the solution $z \in \mathbb{R}^v$ to

$$y_t \approx \ell_\theta^2(z_{t_n}) \quad \text{where} \quad z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z_s) \mathrm{d}X_{\mathbf{x}} \quad \text{and} \quad z_{t_0} = \ell_\theta^1(t_0, x_0), \tag{3.10}$$

is the solution of a *neural controlled differential equation*. Here "$f_\theta(z_s)\mathrm{d}X_{\mathbf{x}}$" denotes a matrix-vector product.

Comparing Equations (3.4) and (3.10), we see that the only difference is that $\mathrm{d}t$ is replaced with $\mathrm{d}X_{\mathbf{x}}$. This small modification is what allows the Neural CDE model to change continuously based on incoming data, since the path $X_{\mathbf{x}}$ adapts the dynamics of the system via the $\mathrm{d}X_{\mathbf{x}}$ term.

3.4.1 Evaluating the Neural CDE model

Before proceeding, we must first understand how to evaluate the Neural CDE equation from Equation (3.10). Whilst the modification of $\mathrm{d}t \to \mathrm{d}X_{\mathbf{x}}$ may seem a simple one, we are left now with a path integral for which the method of solution is not immediately clear.

In this chapter and Chapter 4 we will assume that the control path $X_{\mathbf{x}}$ is at least piecewise differentiable. This will be relaxed in Chapter 5, but for now this means that we can write

$$X_{\mathbf{x}} = \frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}s}(s)\mathrm{d}s, \tag{3.11}$$

(except at finitely many points) and thus we define

$$g_{\theta, X_{\mathbf{x}}}(z, s) = f_\theta(z)\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}s}(s) \tag{3.12}$$

and then we can write

$$z(t) = z(t_0) + \int_{t_0}^{t} f_\theta(z)\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}s}(s)\mathrm{d}s = z(t_0) + \int_{t_0}^{t} g_{\theta, X_{\mathbf{x}}}(z, s)\mathrm{d}s. \tag{3.13}$$

We can see that, under the assumption of piecewise differentiability, the solution $z(t)$ can be written as the solution of an ODE equation. This ODE, given in Equation (3.13), is almost identical to that of Equation (3.4) but modified by the derivative

term $\mathrm{d}X_{\mathbf{x}}/\mathrm{d}s$. This modification is the CDE modification that allows the solution to exhibit continuous dependence upon the data process $X_{\mathbf{x}}$.

If $X_{\mathbf{x}}$ is piecewise differentiable, but not differentiable, then Equation (3.4) can be solved provided we take care to align the integration steps with the jumps.

3.4.2 Modelling irregular time series

The Neural CDE model depends on a continuous time embedding of the discrete data $\mathbf{x}$. This means that messy data (irregular and partially observed) is placed on the same footing as regular data, since they both result in a continuous time control path in $\mathbb{R}^d$.

This is in contrast to an RNN-type model, where such data is not immediately applicable in the same pipeline. One must think carefully about how to represent the missing data, and how to provide information about the irregularities in sampling, to the RNN. With a Neural CDE, no such difference in processing or handling is required, which is why we regard Neural CDEs to be a particularly good fit for messy time series.

3.4.3 Optimisation approach for neural CDEs

In this thesis we will always convert the neural CDE onto a corresponding ODE, either via the derivative of a smooth control or through the paths log-signature (as we will see Chapter 5). As such, the notes on what optimisation method to choose follow those outlined for neural ODEs in Section 3.2.2.

In this chapter we use the optimise-then-discretise approach, however in Chapters 4 and 5 we use the discretise-then-optimise approach since we found the performance to be significantly better with a drastic improvement in training time.

It would be interesting to utilise some of the recently developed ideas on stable and reversible operators Haber and Ruthotto (2017); Onken and Ruthotto (2020); however, we only came across the literature after the experiments in Chapters 3 to 5 were completed, as such, it is left for future work to experiment on such things with neural CDEs.

3.5 Comparison to RNN

An RNN seeks to learn a network $f_\theta \colon \mathbb{R}^{d+v} \to \mathbb{R}^v$ via

$$z_n = f_\theta(x_n, z_{n-1}). \tag{3.14}$$

This results in a hidden state z that is defined only at discrete observation times.

A Neural CDE learns $f_\theta \colon \mathbb{R}^v \mapsto \mathbb{R}^{d \times v}$ such that

$$z_n = f_\theta(z_{n-1})\mathrm{d}X_{\mathbf{x}}, \tag{3.15}$$

where $X_{\mathbf{x}}$ is a continuous time representation of the discrete data $\mathbf{x}$.

We see that $z(t) \in \mathbb{R}^v$ is analogous to the hidden state of the RNN equation, much like the neural ODE solution is analogous to the intermediate hidden layers of a ResNet.

RNNs are a special case of Neural CDEs It is easily shown that RNNs are a special case of Neural CDEs. An RNN of the form

$$z_{i+1} = h_\theta(z_i, x_i)$$

is an explicit Euler discretisation with step unit length of

$$z(t) = z(t_0) + \int_{t_0}^{t} h_\theta\big(z(s), X_{\mathbf{x}}(s)\big) - z(s)\ \mathrm{d}s,$$

which is a special case of a Neural CDE (Kidger et al., 2020, Theorem C.1).

A comparative picture between the RNN, ODE-RNN, and Neural CDE is given at the beginning of this section in Figure 3.0.1. We see that the RNN has a discretely defined hidden state; the ODE-RNN evolves continuously between observation, and jumps at observations; and the neural CDE results in a continuously defined hidden state due to it evolving based on a continuous time embedding of the discrete data.

3.6 Experimental evaluation

In this section we will overview the performance of the Neural CDE model with a natural cubic spline interpolation for the control path. Natural cubic splines were used in the original Neural CDE paper by Kidger et al. (2020) and make a good choice because they are differentiable and slowly varying making them fast to integrate numerically.

In the following chapter, we will perform an extensive study into what it means to be a good control signal for Neural CDEs. In particular, we will examine where natural cubic splines fail (most notably in online prediction problems) and offer up alternatives that improve upon it in certain situations. However, we choose to first present results from the original implementation of Neural CDEs before we go on to examine improvements.

3.6.1 Irregular time series modelling with CharacterTrajectories

As noted in Section 3.4.2, Neural CDEs are a natural model for handling irregular time series data.

We verify this empirically by examining their performance on the CharacterTrajectories dataset from the UEA archive Bagnall et al. (2018). This is a dataset of 2858

Model	Test accuracy			Mem (MB)
	30% dropped	50% dropped	70% dropped	
GRU-ODE	92.6 ± 1.6	86.7 ± 3.9	89.9 ± 3.7	1.5
GRU-Δt	93.6 ± 2.0	91.3 ± 2.1	90.4 ± 0.8	15.8
GRU-D	94.2 ± 2.1	90.2 ± 4.8	91.9 ± 1.7	17.0
ODE-RNN	95.4 ± 0.6	96.0 ± 0.3	95.3 ± 0.6	14.8
Neural CDE	**98.7 ± 0.8**	**98.8 ± 0.2**	**98.6 ± 0.4**	**1.3**

Table 3.6.1 – Test accuracy (mean ± std, computed across five runs) and memory usage on CharacterTrajectories. Memory usage is independent of repeats and of amount of data dropped. These results are courtesy of Patrick Kidger.

time series, each of length 182, consisting of the x, y position and pen tip force whilst writing a Latin alphabet character in a single stroke. The goal is to classify which of 20 different characters are written. The excluded characters were (f, i, j, k, t, x) which require two strokes.

Three experiments are run in which either 30%, 50%, or 70% of the data is dropped from each sample, thus making the data progressively more irregular.

The results are shown in Table 3.6.1. We see that the Neural CDE outperforms all models in each category, including memory. In particular, note that other models (except the ODE-RNN) see significant reductions in performance as the problem becomes more irregular. Not only does the Neural CDE retain top performance for all levels of dropped data, but it is also more robust to increased irregularity of the data.

3.6.2 PhysioNet sepsis prediction task

We use data from the PhysioNet 2019 challenge on sepsis prediction [32, 33]. This is a dataset of 40335 time series of variable length, describing the stay of patients within an ICU. Measurements are made of 5 static features such as age, and 34 time-dependent features such as respiration rate or creatinine concentration in the blood, down to an hourly resolution. Most values are missing; only 10.3% of values are observed. We consider the first 72 hours of a patient's stay, and consider the binary classification problem of predicting whether they develop sepsis over the course of their entire stay (which is as long as a month for some patients).

Two experiments were run for each model, one with the observational frequencies c_i, and one without.

The results are presented in Table 3.6.2. We see that upon inclusion of the observational frequency mask, the Neural CDE model achieves both the top overall performance and has the lowest memory usage.

Model	Test AUC		Memory (MB)	
	c_i	No c_i	c_i	No c_i
GRU-ODE	0.852 ± 0.010	0.771 ± 0.024	454	273
GRU-Δt	0.878 ± 0.006	0.840 ± 0.007	837	826
GRU-D	0.871 ± 0.022	**0.850 ± 0.013**	889	878
ODE-RNN	0.874 ± 0.016	0.833 ± 0.020	696	686
Neural CDE	**0.880 ± 0.006**	0.776 ± 0.009	**244**	**122**

Table 3.6.2 – Test AUC (mean ± std, computed across five runs) and memory usage on PhysioNet sepsis prediction. 'c_i' refers to the additional inclusion of the observational frequencies.

3.7 Limitations

Parameters The map $f_\theta \colon \mathbb{R}^v \to \mathbb{R}^{v \times d}$ is trilinear since it requires $\mathcal{O}(v^2 d)$ parameters – currently a major bottleneck for neural CDEs.

One would expect that the matrix $\mathbb{R}^{v \times d}$ to be low-rank. The situation is reminiscent of transformers where recent work has shown that low-rank approximations of the self-attention matrix can result in similarly performant models at an order of magnitude lower memory (Wang et al., 2020b).

To learn a low-rank approximation we can instead let f_θ map to matrices $A_1 \in \mathbb{R}^{v \times m}$ and $A_2 \in \mathbb{R}^{m \times d}$ such that $m \ll \min(v, d)$. From this we can construct $f_\theta(z) \coloneqq A_1 A_2 \in \mathbb{R}^{v \times d}$ which will be a low-rank approximation. However, initial work on this has not been promising.

An alternative is to use a sparse network layer, of which many implementations exist with autograd support. This has achieved good results in some initial work, with retainment of performance at over 95% sparsity for some problems.

More work on both of these options is needed to claim anything conclusive.

Choice of interpolation scheme The natural handling of irregularly sampled data does not come for free. We must first create a continuous embedding of the data that is 'faithful' to the original data. We go into much more detail on this front in the following chapter, but it is worth noting that the choice of interpolation scheme to use is fundamentally a modelling choice and there is not always a simple option.

Lack of gating Gating in RNNs (e.g. GRU/LSTM) is known to improve performance in most cases. However, there does not seem to be a particularly natural way to enable gating in neural CDEs. In future work we would like to examine potential gating strategies to increase the expressiveness of the model.

3.8 Conclusions

We have introduced the neural controlled differential equation that can be considered as a continuous time equivalent for the RNN. The model is particularly well suited to modelling functions on irregular time time series due to it interacting with the data through a continuous time embedding of the discrete time series which place all time series on the same playing field. Finally, we demonstrated that the model achieves state-of-the-art performance against similar models in empirical studies across different datasets.

Chapter 4

Understanding Control Signals for Neural CDEs[1]

The Neural CDE implementation introduced in Chapter 3 relies on a non-causal interpolation of the data for continuous time data interpretation. This means that Neural CDEs are not suitable for use in online prediction tasks, where predictions need to be made in real-time: a major use case for recurrent networks. Here, we show how this limitation may be rectified. First, we identify several theoretical conditions that interpolation schemes for Neural CDEs should satisfy, such as boundedness and uniqueness. Second, we use these to motivate the introduction of new schemes that address these conditions, offering in particular measurability (for online prediction). Third, we empirically benchmark our online Neural CDE model on three continuous monitoring tasks from the MIMIC-IV medical database demonstrating improved performance on a range of ODE and non-ODE benchmarks.

● *Morrill, J., Kidger, P., Yang, L., and Lyons, T. (2021a). Neural controlled differential equations for online prediction tasks.* arXiv:2106.11028

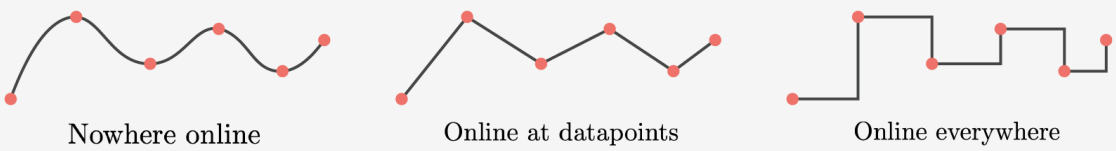


Figure 4.0.1 – An illustration of three different data interpolation schemes. **Left:** A natural cubic interpolation scheme, offline as all points depend on all datapoints (including future points). **Middle:** Linear interpolation, offline in-between datapoints as the scheme depends on the following datapoint. **Right:** Rectilinear scheme which is online everywhere as all interpolated values depend only on previous data.

[1]Joint work with Patrick Kidger and Lingyi Yang

4.1 Introduction

Neural differential equations are an elegant formulation combining continuous-time differential equations with the high-capacity function approximation of neural networks. This makes them an appealing methodology for handling irregular time series. Recent examples include Rubanova et al. (2019); Jia and Benson (2019); De Brouwer et al. (2019); Kidger et al. (2020); Herrera et al. (2021); Morrill et al. (2021b); Kidger et al. (2021) amongst others.

In this chapter, we extend the Neural Controlled differential equation introduced in Chapter 3 Kidger et al. (2020). These were introduced as the general continuous-time limit of arbitrary RNNs. Besides these appealing theoretical connections – and indeed recent work on RNNs has often explicitly designed them around differential-equation-like structures (Chang et al., 2019) – Neural CDEs have additionally been shown to demonstrate excellent empirical performance. In particular, they have been shown to outperform similar Neural ODE or RNN models at modelling functions of irregular time series in offline prediction tasks, where all data is observed in advance (Kidger et al., 2020; Bellot and van der Schaar, 2021).

However despite these appealing properties, Neural CDEs as introduced in the previous chapter are not able to be used to learn and predict in real-time (where new data arrives during inference), due to the solution trajectory exhibiting dependence on future data. In contrast, other Neural ODE variants such as the ODE-RNN (Rubanova et al., 2019) can already handle online processing.

In this chapter, we show how this may be rectified, so that Neural CDEs may be adapted to apply to online problems.

4.1.1 Recapping Neural CDEs

Provided $X_{\mathbf{x}} \in \mathbb{R}^d$ is piecewise continuously differentiable (as will always be the case for us), then the Neural CDE model is defined as the solution $z \in \mathbb{R}^v$ to

$$z(t_0) = \ell_\theta(t_0, x_0), \quad z(t) = z(t_0) + \int_{t_0}^{t} f_\theta(z(s))\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}s}\mathrm{d}s \quad \text{for } t \in (t_0, t_n], \tag{4.1}$$

and as such, the model can be interpreted and solved as an ordinary differential equation. Here "$f_{\theta_1}(z(s))\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}s}$" denotes a matrix-vector product. The solution z is said to be the response of a CDE *driven or controlled by* $X_{\mathbf{x}}$.

We highlight here that the integration does not need to be done with respect to time, and can be done with respect to some alternative parameterisation such as that introduced with the rectilinear interpolation scheme in definition 2.2.8. In such cases we include time as the first dimension of $X_{\mathbf{x}}$ so that $X_{\mathbf{x}} \in \mathbb{R}^{d+1}$ and define a fictional parameterisation over $X_{\mathbf{x}}$ that we integrate with respect to. For example in the rectilinear scheme from definition 2.2.8 we define our parameterisation to run from $[0, 2n]$ where n is the number of observations and we alternate between incrementing time and incrementing the feature dimension.

The evolving $z(t) \in \mathbb{R}^v$ is analogous to the hidden state in an RNN, now operating in continuous time. Typically, the output of the model will be a linear map on this hidden state: either applied to $z(t)$ for all times $t \in [t_0, t_n]$ to produce a time-dependent output path, or on just $z(t_n)$ for a single output such as for classification.

4.1.2 Continuous time control signals for Neural CDEs

Neural CDEs act on and require a continuous-time embedding $X_{\mathbf{x}}$ of the observed data $\mathbf{x}$. This provides a number of benefits. Firstly, this simplifies the handling of 'messy' (irregularly sampled with missing data) time series, by enabling it to be interpreted in the same way as regular data. Additionally, as this results in an ODE-like model, the model produces a continuously defined solution, has memory-efficient continuous adjoint methods, and the utilisation of modern ODE solvers offers trade-offs between error and computation.

The previous chapter took the map $\mathbf{x} \to X_{\mathbf{x}}$ to be a natural cubic spline which notably cannot be used in an online fashion. This is because predictions at time t_i depend on future, as-yet-unobserved, data at $t > t_i$.

4.2 What makes a good control signal?

We now introduce four conditions that a good control signal should satisfy.

4.2.1 Adapted measurability

A model which can learn and predict in real-time is often referred to as an *online model* in machine learning. This is analogous to the concept of an *adapted measurable process* in the language of probability theory (Williams, 1991). This is the primary property of interest to us here, which is not satisfied by existing implementations of Neural CDEs. Fulfilment of this property is what will enable Neural CDEs to be deployed in real-world real-time scenarios, such as continuous monitoring in ICU settings.

$\mathcal{T}-$measurable Let $\mathcal{T} \subseteq [t_0, t_n]$. We say that the Neural CDE solution $z(t)$ (of Equation (3.10)) is $\mathcal{T}-$measurable if for all $t \in \mathcal{T}$ we have that $z(t)$ is a function of only (t_i, x_i) for those $t_i \in [t_0, t]$. That is, at times $\mathcal{T}$ it is possible to obtain a prediction in an online setting.

Continuously online If the solution is $\mathcal{T}-$measurable for $\mathcal{T} = [t_0, t_n]$, we say that is is *continuously online*.

Discretely online Suppose the solution $z(t)$ is $\mathcal{T}-$measurable for $\mathcal{T} = \{t_0, \ldots, t_n\}$ (that is, at the observation times). For t in (t_i, t_{i+1}) between the observation times, suppose $z(t)$ depends on the data up to one observation ahead $((t_0, x_0), \ldots, (t_{i+1}, x_{i+1}))$.

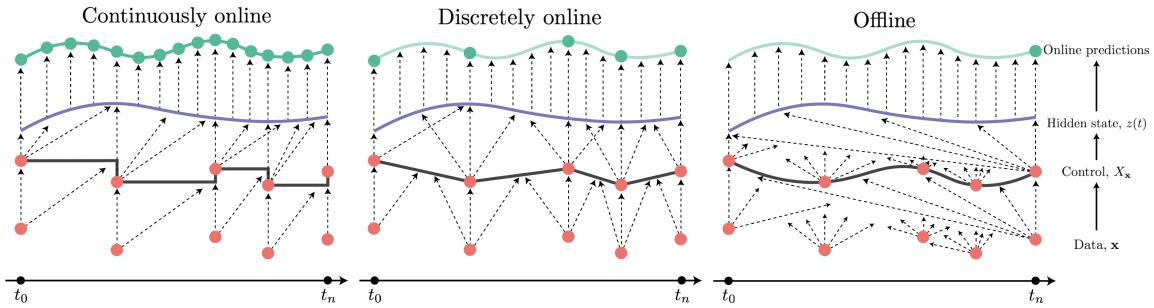


Figure 4.2.1 – Graphical description of the measurability definition for Neural CDEs. Arrows indicate the direction of time datapoints can influence. ● indicates an online prediction can be made at that point. **Left:** a continuously online model where no information is passed backwards in time, resulting in an online solution at all points in time. **Middle:** a discretely online scheme where information can be passed backwards, but not further than the preceding observation. **Right:** an offline scheme where information is passed backwards in time further than the preceeding observation.

Then we call the solution *discretely online*. In other words, the solution is online only at the discrete observation times.

Offline If the solution is not at least discretely online then we say it is *offline*.

We choose to sub-categorise the definitions in this way since it well separates existing models. For example, standard RNNs are discretely online, an ODE-RNN (Rubanova et al., 2019) is continuously online, and existing Neural CDEs are offline.

A graphical depiction of this is shown in Figure 4.2.1.

4.2.2 Smoothness

To be able to apply numerical solvers to the integral in Equation (3.10) we require that the integrand be sufficiently smooth, and thus that the control be sufficiently smooth. As a minimum (using Euler's method), we require that $X_{\mathbf{x}}$ be piecewise twice continuously differentiable with bounded second derivative. In practice to use a higher-order numerical method, such as Dormand–Prince, then additional smoothness is desirable.

When using an adaptive solver and a control that is piecewise smooth, but not smooth, the solver should be informed about the jumps between pieces so that its integration steps may align with them. Without this the solver must locate the discontinuities on its own, slow down to resolve them, and then speed up again, which is a numerically expensive procedure. For example this may be done using the `jump_t` argument for `torchdiffeq` (Chen et al., 2018), or the `d_discontinuities` argument for DifferentialEquations.jl (Rackauckas and Nie, 2017).

4.2.3 Boundedness

We require that $X_{\mathbf{x}}$ should behave "reasonably". It should not introduce any spurious oscillations or grow unboundedly from bounded data. This condition ensures that $X_{\mathbf{x}}$ is a good representative of $\mathbf{x}$, so that the Neural CDE can learn from the data.

Formally, let $\tau_i = t_{i+1} - t_i$ for each i. Then we require that there exists some continuous $\omega \colon \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that

$$\|X_{\mathbf{x}}\|_\infty + \left\| \frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t} \right\|_\infty + \left| \frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t} \right|_{BV} < \omega(\max_i \tau_i, \min_i \tau_i, \max_i |x_i|), \tag{4.2}$$

where $|\cdot|_{BV}$ denotes the bounded variation seminorm.

This condition is required to attain the universal approximation property for Neural CDEs. A formal proof is given in Appendix A.1, but the main idea is as follows. Consider a collection of time series $\mathcal{X}$ for which $\sup_{\mathbf{x} \in \mathcal{X}} \omega(\max_i \tau_i, \min_i \tau_i, \max_i |x_i|) < \infty$. Then (4.2) implies that $\mathfrak{X} = \{X_{\mathbf{x}}\}\, \mathbf{x} \in \mathcal{X}$ is relatively compact with respect to the topology generated by $\|X\|_\infty + \left\| {}^{\mathrm{d}X}/_{\mathrm{d}t} \right\|_1$, and hence also with respect to the topology generated by $\|X\|_\infty + |X|_{BV}$. This then satisfies the compactness condition of Kidger et al. (2020, Theorem B.7), implying that Neural CDEs driven by $X \in \mathfrak{X}$ are universal approximators on $\mathbf{x} \in \mathcal{X}$.

We note that this property is non-trivial: for example, quadratic splines exhibit a resonance property that may result in unbounded oscillations as time progresses.

4.2.4 Control signal uniqueness

Given a collection of time series $\mathcal{X}$, we say that a control $X_{\mathbf{x}}$ is unique with respect to $\mathcal{X}$ if

$$\mathbf{x} \to (x_0, \text{Signature}(X_{\mathbf{x}})) \text{ is injective with respect to } \mathcal{X}. \tag{4.3}$$

This is required, along with the boundedness property, for universal approximation with Neural CDEs to hold. Signature denotes the signature transform, which is central to the study of CDEs (Lyons, 1998; Lyons et al., 2007b; Bonnier et al., 2019; Morrill et al., 2020a; Kidger and Lyons, 2021).

For the reader unfamiliar with signature transforms, this may intuitively be approximated by the simplified (but not completely accurate) condition

$$\mathbf{x} \to X_{\mathbf{x}} \text{ is injective with respect to } \mathcal{X}. \tag{4.4}$$

In situations where we do not have tree-like pieces (Hambly and Lyons, 2010), then Section 4.2.4 $\Leftrightarrow$ Eq. (4.4). The simple inclusion of a time dimension is sufficient for no tree-like pieces to exist.

4.3 Control signals for Neural CDEs

As mentioned in Section 4.2, previous work has not explored the choice of control in much detail. Here we overview existing controls and identify the theoretical properties from Section 4.2 that each possesses. We will then introduce two new controls – cubic Hermite splines with backward differences, and rectilinear controls – that address issues with existing control schemes.

Natural cubic splines Natural cubic splines were used in the original Neural CDE paper by Kidger et al. (2020). This control signal requires the full time series to be available prior to construction, as $z(t)$ depends on all datapoints $\big((t_0, x_0), \ldots (t_n, x_n)\big)$. Any change in one datapoint has a small effect on the entire construction, even at earlier times. As such, natural cubic splines cannot be used in an online fashion. If an offline scheme is sufficient, then these do still make a good choice: they are relatively smooth and slowly varying, making them fast to integrate numerically.

Linear control This is arguably the simplest and most natural control signal, whereby we apply linear interpolation between observations.

For fully observed data, the linear control defines a discretely online control path, and thus has the same online properties as an RNN. If on the other hand there exists missing data then it cannot be used even discretely online. To see why this is the case, consider the following data observations

$$\mathbf{x} = \big((t_0, x_0), (t_1, *), (t_2, x_2)\big), \tag{4.5}$$

with $*$ denoting missing data. The linear control at time t_1 is dependent on the value x_2 at t_2, and as such, it cannot define an online solution at t_1.

Whilst it has better online properties, the linear control is generally computationally slower than natural cubic splines. This is due to the need to resolve derivative discontinuities at the knots (t_i, x_i) (assuming the data is sampled more finely than required for the adaptive time integrator), as in Section 4.2.2.

Cubic Hermite splines with backward differences This scheme smooths the discontinuities in the linear control, whilst retaining the same online properties. We achieve this by joining adjacent timepoints with a cubic spline where the additional degrees of freedom are used to smooth gradient discontinuities. This leads to faster integration times than linear controls.

This differs from natural cubic splines as it solves a single equation on each $[i, i + 1)$ piece independently. As a result, it is more quickly varying than natural cubic splines (see Figure 4.3.1) and so produces slower integration times than natural cubic splines.

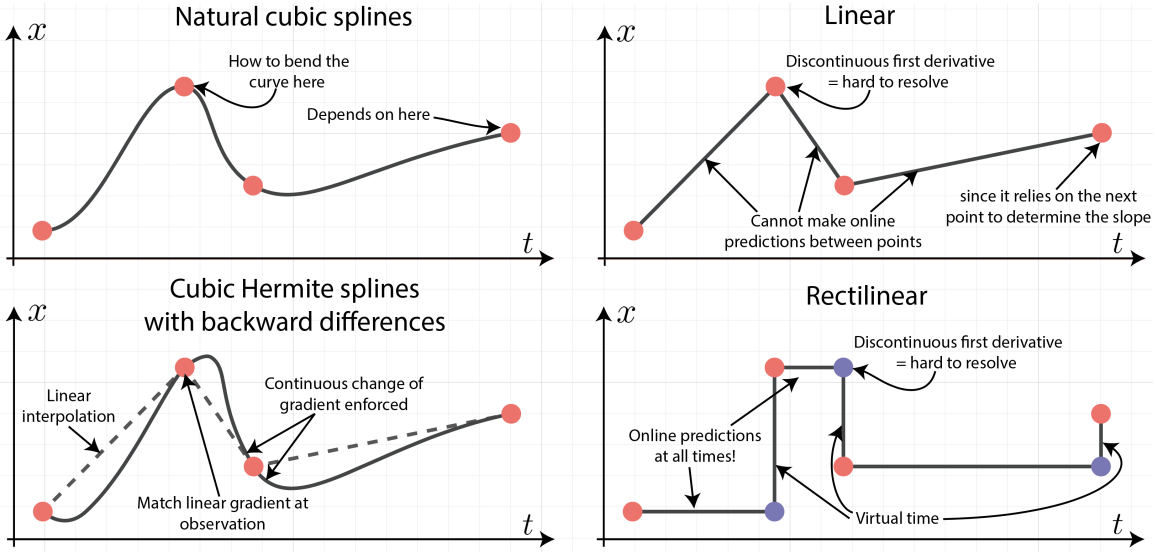


Figure 4.3.1 – Graphical comparison of the four control signals: natural cubic splines (top left), linear control (top right), cubic Hermite splines with backward differences (bottom left), rectilinear control (bottom right).

For each interval $[i, i+1)$, we ensure that $X_{\mathbf{x}}(i) = (t_i, x_i)$, $X_{\mathbf{x}}(i+1) = (t_{i+1}, x_{i+1})$, and that the gradient at each node matches the backward finite difference

$$\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t}(i) = x_i - x_{i-1},$$
$$\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t}(i+1) = x_{i+1} - x_i.$$

The result of this is a control signal with continuous derivatives that is discretely online.

Rectilinear control Each of the previous schemes need to look at least one time-step ahead to construct the control between time points. This means they are not continuously online.

To resolve this, we turn to a control construction that has already been introduced in Chapter 2, the *rectilinear interpolation* from Definition 2.2.8. We begin by letting $\widetilde{x}_i$ denote the forward fill of x_i with respect to the missing data $*$. We then let $X_{\mathbf{x}}\colon [0, 2n] \to \mathbb{R}^v$ be piecewise linear such that $X_{\mathbf{x}}(2i) = (t_i, \widetilde{x}_i, c_i)$ for $i \in \{0, \ldots, n\}$ and $X_{\mathbf{x}}(2i-1) = (t_{i+1}, \widetilde{x}_i, c_i)$ for $i \in \{1, \ldots, n\}$.

The produces a control signal – the *rectilinear control* – that updates the time and feature channels separately in lead-lag fashion. While channels are not observed, time is increased as normal. When a channel is observed, then we interpolate between channel values with time held fixed. This is shown pictorially in the bottom right of Figure 4.3.1. ODE-RNNs (Rubanova et al., 2019) may be seen as a special case of a Neural CDE with rectilinear control.

	Properties			
Control signal	Measurability	Smoothness	Boundedness	Uniqueness
Natural cubic	✗	✔	✔	$\sim$
Linear	(discrete)*	(piecewise)	✔	✔(with c_i)
Cubic Hermite	(discrete)*	✔	✔	✔(with c_i)
Rectilinear	✔	(piecewise)	✔	✔(with c_i)

Table 4.3.1 – Summary of the interpolation schemes and the properties they hold for irregular and partially observed data. A superscript $*$ denotes that the property holds only if no data is missing, and $\sim$ if the property is unknown. Proofs for previously unknown properties are given in Appendix A.1

The downside of this scheme is that the parameterisation is twice as long (with domain $[0, 2n]$ rather than $[0, n]$), with correspondingly many derivative discontinuities. This means that the model takes longer to evaluate, and to train.

In Table 4.3.1 we summarise each of the aforementioned schemes, and the properties from Section 4.2 that each holds. Corresponding proofs are given in Appendix A.1.

4.4 Experiments

We begin by evaluating each of the controls across a range of both regularly sampled and irregularly sampled datasets. We then go on to benchmark the online Neural CDE against a collection of benchmarks on MIMIC-IV prediction tasks.

Each model used hyperparameters that were chosen by a Bayesian optimisation strategy with 20 trials. The ranges of values the hyperparameters were optimised over as well as the final configuration for every model is given in Appendix A.2. Full experimental details including optimisers, learning rates, normalisation, architectures and so on can be found in Appendix A.2. The code for reproducing all results is available at `github.com/jambo6/online-neural-cdes`.

4.4.1 Datasets

First we overview the datasets used in our analysis. They all had a 70%/15%/15% train/validation/test split, with (as required by the Neural CDE formulation) time included as a channel.

We begin by with four regularly sampled and fully observed datasets.

BeijingPM2.5, BeijingPM10: Both contain 10 channels and 16966 total samples. The aim is to predict the PM2.5/PM10 level (two pollutant indexes) at 12 different air-quality monitoring sites in Beijing (Tan et al., 2020). The performance metric is taken to be the RMSE against the true value.

SpeechCommands: Contains 11 channels and 34975 samples of one-second audio recordings of individual spoken words such as 'yes', 'no', 'left', and 'right' (Warden, 2018). The performance metric is taken to be classification accuracy of the spoken words.

CharacterTrajectories: Contains 4 channels and 2872 samples of the x, y position and the force applied by the pen tip whilst writing a letter from the Latin alphabet in a single stroke (Bagnall et al., 2018). The performance metric used is the classification accuracy for the different characters.

In addition to these, we additionally examine three tasks for the MIMIC-IV database (Johnson et al., 2021; Goldberger et al., 2000) related to continuous patient health monitoring. The database contains 76540 de-identified admissions to intensive care units at the Beth Israel Deaconess Medical Center. The data is highly irregular and channels have lots of missing data. Each of the three tasks has its own set of exclusion criteria; these are outlined in full in Appendix A.2.3.

Mortality: We predict the likelihood of mortality within an ICU stay from some initial data. The performance metric is the AUC of prediction of eventual mortality.

LOS: We estimate the length of stay of a patient given their first 24 hours of data. The performance metric is the RMSE against the true value in days.

Sepsis: We predict the risk of sepsis along a patient's stay. The performance metric is the AUC against the true state of sepsis that is defined according to the Sepsis-3 definition (Singer et al., 2016) (full details are given in Appendix A.2.3).

In terms of metrics, lower is better for BeijingPM2.5, BeijingPM10, and LOS, whilst higher is better for the other tasks.

4.4.2 Empirical study on control signals

In Tables 4.4.1 to 4.4.4 we compare the performance of the control signals on the regularly sampled datasets.

The NFEs column (Number of Function Evaluations per epoch) shows that natural cubic splines are uniformly the fastest choice. For example on the SpeechCommands dataset, natural cubic splines require an average of 2.55×10^4 evaluations per epoch, piecewise cubic nearly double on 4.16×10^4, linear nearly triple on 6.35×10^4, and rectilinear significantly more on 1.08×10^5.

However, the linear, cubic Hermite, and rectilinear schemes all produce better RMSE and accuracies, generally similar to each other. As they are the fastest from this group, we thus recommend cubic Hermite splines with backward differences as the best choice on regular datasets.

We see a similar story on the irregularly sampled MIMIC-IV tasks in Tables 4.4.5 and 4.4.6 (see also Appendix A.2.3 for results on the sepsis task). Natural cubic

Tables 4.4.1 to 4.4.4: Control signal comparison for the *regularly sampled* datasets using the 'dopri5' adaptive solver. The top performer for each dataset is given in bold. NFEs represents the number of function evaluations per epoch.

Control	RMSE	NFEs $\times 10^3$
Natural cubic	53.3 ± 0.3	**4.6 ± 0.1**
Linear	**51.7 ± 0.8**	5.4 ± 0.3
Cubic Hermite	52.5 ± 0.7	4.8 ± 0.1
Rectilinear	52.0 ± 1.3	16.3 ± 0.4

Table 4.4.1 – BeijingPM2.5

Control	RMSE	NFEs$\times 10^3$
Natural cubic	77.6 ± 1.7	**4.5 ± 0.1**
Linear	**77.5 ± 2.4**	5.7 ± 0.1
Cubic Hermite	78.5 ± 1.6	4.7 ± 0.1
Rectilinear	79.0 ± 0.7	15.7 ± 0.4

Table 4.4.2 – BeijingPM10

Control	ACC (%)	NFEs$\times 10^3$
Natural cubic	83.6 ± 6.1	**1.0 ± 0.2**
Linear	97.6 ± 1.5	2.2 ± 0.1
Cubic Hermite	**99.3 ± 0.0**	1.9 ± 0.0
Rectilinear	98.6 ± 0.5	7.9 ± 0.4

Table 4.4.3 – CharacterTrajectories

Control	ACC (%)	NFEs$\times 10^4$
Natural cubic	92.9 ± 0.3	**2.67 ± 0.36**
Linear	93.3 ± 0.7	6.35 ± 0.50
Cubic Hermite	92.5 ± 0.4	4.16 ± 0.03
Rectilinear	**93.7 ± 0.8**	11.3 ± 0.67

Table 4.4.4 – SpeechCommands

Tables 4.4.5 and 4.4.6: Control signal comparison for the *irregularly sampled* Mortality and LOS MIMIC-IV tasks using the `'dopri5'` adaptive solver. The top performer for each dataset is given in bold. NFEs represents the number of function evaluations per epoch.

Control	AUC	NFEs$\times 10^3$
Natural cubic	0.859 ± 0.003	14.5 ± 0.1
Linear	**0.910 ± 0.003**	36.9 ± 0.8
Cubic Hermite	0.909 ± 0.002	26.0 ± 0.6
Rectilinear	0.906 ± 0.002	96.3 ± 0.1

Table 4.4.5 – Mortality

Control	RMSE	NFEs$\times 10^3$
Natural cubic	0.241 ± 0.005	4.0 ± 0.0
Linear	0.149 ± 0.037	7.8 ± 0.2
Cubic Hermite	0.138 ± 0.025	4.2 ± 0.1
Rectilinear	**0.109 ± 0.008**	11.9 ± 1.0

Table 4.4.6 – LOS

splines are again the fastest, but again their AUC/RMSE performance is poor. Linear/piecewise cubic performs best on the Mortality prediction tasks, whereas rectilinear is the best on LOS. Note that only the rectilinear model can actually be deployed in a continuous ICU monitoring scenario.

4.4.3 Benchmarking on MIMIC-IV

Finally, we benchmark the online (rectilinear) Neural CDE on the three tasks from the MIMIC-IV database.

We highlight that Neural CDEs have been tested before on such problems before (notably Kidger et al. (2020) on a sepsis detection tasks); however, this is the first such benchmarking of a Neural CDE model that could actually be deployed in an *online, real-time* in-hospital environment.

Our benchmarks include: vanilla GRU; GRU-dt, which is a GRU that additionally includes the time difference between observations; GRU-dt-intensity, the same as a GRU-dt but also includes the observational intensity; GRU-D (Che et al., 2018),

Model	MIMIC-IV		
	Mortality	LOS	Sepsis
GRU	0.846 ± 0.05	0.236 ± 0.028	0.791 ± 0.002
GRU-dt	0.912 ± 0.003	0.149 ± 0.008	0.8 ± 0.001
GRU-dt-intensity	0.924 ± 0.014	0.142 ± 0.013	**0.801 ± 0.002**
GRU-D	0.93 ± 0.002	0.148 ± 0.002	**0.801 ± 0.002**
ODE-RNN	0.524 ± 0.004	0.154 ± 0.004	0.793 ± 0.001
Online Neural CDE	0.908 ± 0.003	0.11 ± 0.009	0.77 ± 0.002
Online Neural CDE w/ observational freq	**0.943 ± 0.003**	**0.099 ± 0.001**	0.795 ± 0.004

Table 4.4.7 – Benchmarking online Neural CDEs (Neural CDEs with rectilinear interpolation) using the `'rk4'` solver with and without observational frequency against a range of algorithms on the MIMIC-IV tasks. The top performer for each problem is shown in bold.

incorporates both time differences and observational intensity but in a more sophisticated manner; and ODE-RNN (Rubanova et al., 2019). The ODE-RNN is chosen as it represents a continuously online ODE-benchmark, whereas all other models operate only discretely online.

The results over three runs are presented in Table 4.4.7. We see that upon inclusion of the measurement intensity matrix, the causal Neural CDE becomes extremely competitive with the GRU-D benchmark, achieves convincing improvements in performance in the LOS and mortality tasks, and is only narrowly beaten in the sepsis detection task. We also see across-the-board superior performance to the ODE-RNN.

It is already known that Neural CDEs are effective at modelling irregular functions on time series. However, these results, in particular those on rectilinear controls, are the first demonstration that Neural CDEs can be adapted for use in a real-world online scenario whilst retaining performance.

4.5 Discussion and limitations

Recommendations for the control signal Considering the results from Section 5.4 and the properties from Section 4.2, we make the following recommendations for Neural CDE control paths:

1. If the problem is online and requires the solution to be continuously online, then use rectilinear controls. This is the only scheme that is continuously online.

   Likewise if the problem is online, has missing data, and requires the solution to be discretely online, then use rectilinear controls. This is the only scheme that is discretely online in the presence of missing data.

2. If the problem is online, has no missing data, and requires the solution to be discretely online, then use cubic Hermite splines with backward differences. We

see in Tables 4.4.1 to 4.4.6 that the performance is in-line with both linear and rectilinear, but at faster speeds.

Likewise, these are also recommended if the problem is offline.

3. If speed is of greater importance than accuracy, and the task is offline, then use natural cubic splines. This is evidenced by the results in Tables 4.4.1 to 4.4.4 which show natural cubic splines to be significantly faster than other alternatives.

As such our findings recommend either rectilinear control or cubic Hermite splines with backward differences, both introduced in this paper, for most cases.

Limitations The main limitation of the techniques introduced here come from rectilinear controls. These are typically slow, and if trained with discretise-then-optimise techniques come with high memory usage. Despite this, in many online cases these are 'the only game in town'.

Implementation To help facilitate adoption, both rectilinear controls and cubic Hermite splines with backward differences have been implemented in the `torchcde` open-source library for CDEs.

4.6 Conclusion

We formalised the properties that ideal Neural CDE control schemes should have. In doing so, we identified two new control schemes that address issues with existing implementations, in particular with respect to online predictions and speed. Having performed both a theoretical and empirical study into the schemes' behaviour, we provide recommendations regarding which scheme to use when. This has included benchmarking the online Neural CDE on three continuous monitoring ICU tasks, in which improved and state-of-the-art performance is demonstrated against similar ODE or RNN based approaches.

Chapter 5

Neural Rough Differential Equations[1]

In this chapter, we use ideas from rough path theory to extend the Neural CDE formulation to a wider class of driving signals. Instead of directly embedding into path space, we instead represent the input signal over small time intervals through its log-signature, the statistics most relevant for solving a CDE. This extension produces real practical benefits for analysing long time series; experimentally we observe improvements in performance, speed, and memory, on problems up to length 17,000.

- Morrill, J., Salvi, C., Kidger, P., Foster, J., and Lyons, T. (2021b). Neural rough differential equations for long time series. *International Conference on Machine Learning*

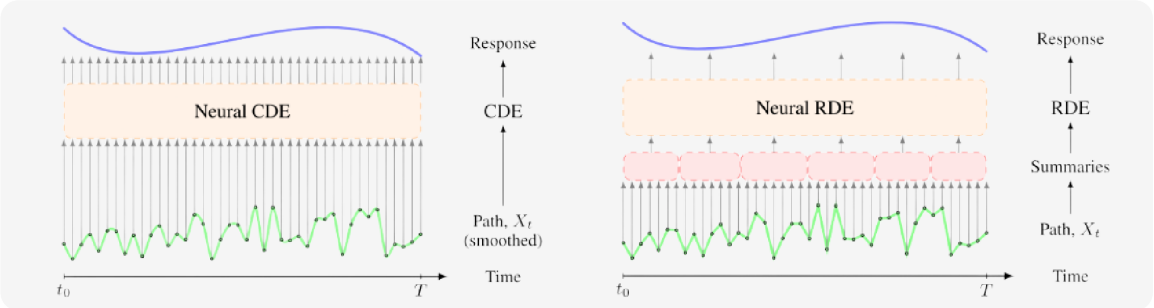


Figure 5.0.1 – Here we give a high level comparison of the CDE and the RDE formulations. **Left:** The original CDE formulation where the data is smoothly interpolated and pointwise derivative information is used to drive the CDE. **Right:** The corresponding rough approach. Local interval summarisations of the data are computed and used to drive the response over the interval.

[1]Joint work with Patrick Kidger, Cristopher Salvi, and James Foster

5.1 Introduction

The previous approach saw us embedding the discrete data $\mathbf{x}$ into continuous-time path space. However, there is an alternative approach. In this Chapter, we instead represent the input signal over small time intervals through its log-signature, which can be intuitively thought of as the features that best describe how an input signal drives a CDE. This is in contrast to Chapter 3 where we drive the response, $z(t)$, by pointwise evaluations of the derivative of the continuous-time embedding of the data, $X_{\mathbf{x}}$.

In contrast to the methodology introduced in Chapters 3 and 4, the key idea behind rough path theory Lyons (1998); Lyons et al. (2004); Friz and Victoir (2010) is that the solution of a CDE is best understood by its effect on systems, rather than by an analysis of the path itself. That is, CDEs should not be understood by pointwise evaluations of the control path (as in Chapters 3 and 4), but instead through a specific summarisation – *the log-signature* – of the control path over intervals. We term a CDE treated in this way a *rough differential equation*, for obvious reasons.

Reformulation in this manner comes with real practical benefits. 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 represent 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.

A high level overview of the differences between the two methods is given in Figure 5.0.1. We can see that the Neural CDE method would evaluate the pointwise derivative at each observation time to drive the signal, whereas the Neural RDE drives the signal by local (log-signature) summaries, resulting in far fewer calls to the integration solver.

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.

5.2 The Log-ODE method

Suppose, as usual, that we observe a time series $\mathbf{x}$ as in Definition 2.2.2. Let $X\colon [0, T] \to \mathbb{R}^d$ be some continuous time embedding of the time series.

Now pick some $0 \leq r < s \leq T$ and consider

$$z(s) = z(r) + \int_r^s f(z_t)\mathrm{d}X_t. \tag{5.1}$$

If we were solving this using the methods from Chapter 3, we would compute the derivative of X and evaluate the integral. However, if the interval contains a lot of data, either many evaluations of the derivative must be computed to get an accurate representation of the underlying data, or lots of information will be lost via this approach. Additionally, what do we do if the control is not differentiable?

It turns out, there is a technique from rough path theory – *the log-ODE method* – that can address both of these issues. The method states that Equation (5.1) can be approximated via

$$z(s) \approx z(r) + \int_r^s \widehat{f}(t) \frac{\text{LogSig}^N_{r,s}(X_t)}{s-r} \mathrm{d}t. \tag{5.2}$$

with N being some chosen truncation depth of the log-signature. We have that $\text{LogSig}^N_{r,s}(X) \in \mathbb{R}^{\beta(d,N)}$ where $\beta(d,N)$ is an integer describing the number of terms in the log-signature of a d-dimensional path truncated to depth N. For information on how this is derived, see Reizenstein (2019).

You may also notice that we have written $\widehat{f}$ rather than f here. $\widehat{f}$ is a modified version of f since it is now being applied to the log-signature which for starters is a different dimension to X. The relationship between them is given in Lyons et al. (2004).

That is, the solution of the CDE over the may be approximated by the ODE from Equation (5.2) over the interval. Crucially, this ODE yields significantly more accurate approximations to $z(s)$ with significantly fewer integration steps that would be required using the derivative.

Typically, this will be applied over the domain $[0, n]$ by picking some points r_i such that $0 = r_0 < r_1 < \cdots < r_m = 1$ (where $m \ll n$), splitting up the CDE over regions $[r_i, r_{i+1}]$ and applying 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*.

5.3 The Neural Rough Differential Equation

Now that we have the log-ODE method, as defined in Section 5.2, it is straightforward to apply this approach to a Neural CDE equation.

Let $\ell^1_\theta \colon \mathbb{R}^d \to \mathbb{R}^v$, $\ell^2_\theta \colon \mathbb{R}^v \to \mathbb{R}^w$ be linear maps, and $f_\theta \colon \mathbb{R}^v \mapsto \mathbb{R}^{v \times d}$ be a neural network. We call z the solution of a neural controlled differential equation if

$$z_0 = \ell^1_\theta(t_0, x_0) \quad \text{and} \quad z_t = \int_0^t f_\theta(z_t) \mathrm{d}X_t \quad \text{with} \quad y_i = \ell^2_\theta(z_t). \tag{5.3}$$

To make this a rough differential equation, we pick r_i such that $0 = r_0 < r_1 < \cdots < r_n = T$, define

$$g_{\theta,X}(z,s) = \widehat{f_\theta}(z) \frac{\text{LogSig}^N_{r_i, r_i+1}(X_t)}{r_{i+1} - r_i} \quad \text{for } s \in [r_i, r_{i+1}), \tag{5.4}$$

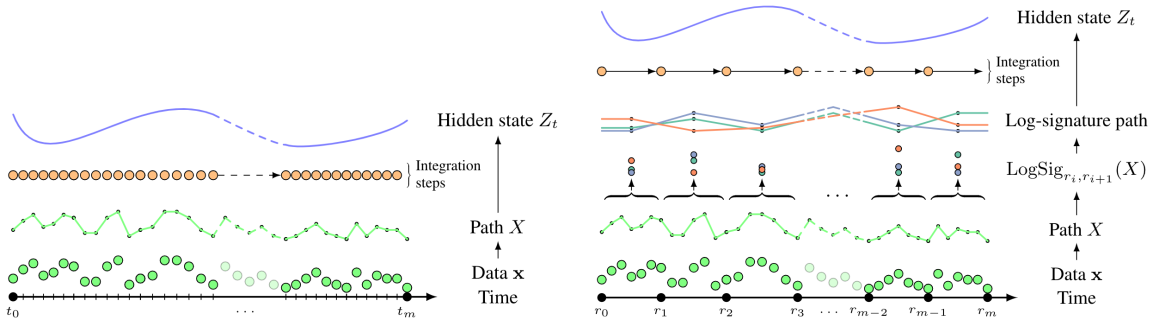


Figure 5.3.1 – An overview of the log-ODE method applied to Neural RDEs. **Left:** A single step (CDE or RDE) model. The path X is quickly varying, meaning a lot of integration steps are needed to resolve it. **Right:** The Neural RDE utilising the log-ODE method with integration steps larger than the 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. Here we have constructed a path in log-signature space by linearly interpolating the log-signatures over each interval.

where $\widehat{f_\theta}\colon \mathbb{R}^v \to \mathbb{R}^{v\times\beta(d,N)}$ is an arbitrary neural network, $\mathrm{LogSig}^N_{r_i,r_i+1}(X) \in \mathbb{R}^{\beta(d,N)}$, and the right hand side denotes a matrix-vector product between $\widehat{f_\theta}$ and the log-signature. We then say z is the solution of a *rough differential equation* if z is as Equation (5.3) where the integral is solved via

$$z(t) = z_0 + \int_0^t g_{\theta,X}(z,s)ds. \tag{5.5}$$

This, like the Neural CDE from Chapter 3, can now be solved as an ODE using standard ODE solvers.

We give an overview of this process in Figure 5.3.1. 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 logsignature.

5.3.1 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.

5.4 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 5.3.1. 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, 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 5.3.1). 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 B.2. For brevity, we provide results for only some of the step sizes here. The full results are described in Appendix B.3.

5.4.1 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 5.4.1. 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

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 5.4.1 – 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.

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.

Results across all step sizes may be found in Appendix B.3.

5.4.2 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 5.4.2.

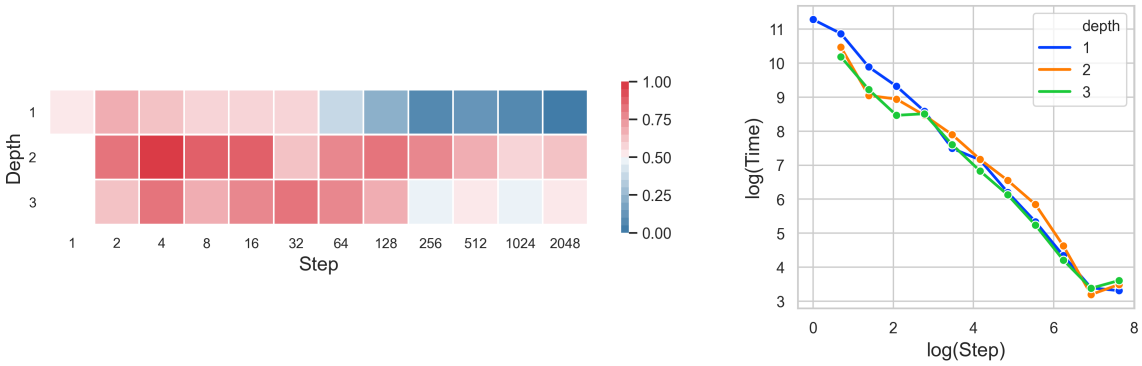


Figure 5.4.1 – **Left:** Heatmap of normalised accuracies on the EigenWorms dataset for differing step sizes and depths. **Right:** Log-log plot of the elapsed time of the algorithm against the step size.

Figure 5.4.2 – 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.

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 5.4.2.

The full results across the full range of step sizes may be found in Appendix B.3.

Model	Step	L^2			Time (Hrs)			Memory (Mb)
		RR	HR	SpO_2	RR	HR	SpO_2	
ODE-RNN	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	**2.42 ± 0.19**	**7.67 ± 0.40**	**2.55 ± 0.13**	2.9	3.2	3.1	43.3
	128	**1.51 ± 0.08**	**2.97 ± 0.45***	**1.37 ± 0.22**	0.5	1.7	1.7	17.3
	512	**1.49 ± 0.08***	**3.46 ± 0.13**	**1.29 ± 0.15***	0.3	0.4	0.4	15.4

Table 5.4.2 – The three experiments on BIDMC datasets: mean ± 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.

5.5 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 This point has been covered in detail in Section 3.2.2, but suffice it to say that the same applies here. The neural RDE method can be solved via a log-signature modified ODE and thus memory efficiency can be attained via utilising the optimise-then-discretise approach, or implementation or a reversible solver or sensible use of checkpointing.

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 (5.2). 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 (Hornik et al.

(1989); Cybenko (1989), or more recent variations Pinkus (1999); Kidger and Lyons (2020)) 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 and Lyons, 2021). As equation (5.5) is an ODE, it may be solved directly using tools such as `torchdiffeq` (Chen, 2018).

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.

5.6 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.

5.7 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 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 and 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.

5.8 Conclusions

We have introduced the neural controlled differential equation that can be considered as a continuous time equivalent for the RNN. The model is particularly well suited to modelling functions on irregular time time series due to it interacting with the data through a continuous time embedding of the discrete time series which place all time series on the same playing field. Finally, we demonstrated that the model achieves state-of-the-art performance against similar models in empirical studies across different datasets.

Part III

Signatures

Chapter 6

The Generalised Signature Method[1]

The signature method refers to a collection of feature extraction techniques for multivariate time series. A large body of literature exists on the subject with authors providing a wide variety of modifications with the aim to improve some aspect of it. However, no paper has sought to collate these modifications together, so as to categorise the different variations and compare them against each other. In this chapter we will do two things: firstly, we identify four groupings under which each proposed variation constitutes a special case of, we then show that these groups can be unified under a general framework that we call the generalised signature method; secondly, we will perform a first-of-its-kind extensive empirical study into these different variations which will allow us to derive a canonical collection of choices that provide us with a domain agnostic starting point for working with signatures.

- Morrill, J., Fermanian, A., Kidger, P., and Lyons, T. (2020a). A generalised signature method for time series. *arXiv:2006.00873*

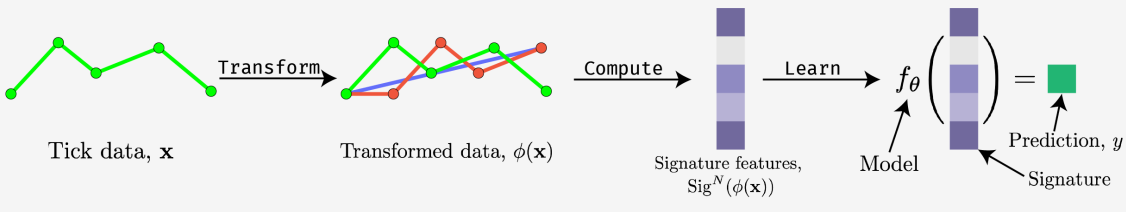


Figure 6.0.1 – High level overview of the signature method. Data is transformed, the signature is computed, and a machine learning model is trained on the resulting features to predict an output.

[1]Joint work with Adeline Fermanian and Patrick Kidger

6.1 What is "The Signature Method"?

Multidimensional stream data poses an assortment of additional challenges compared with standard static tabular feature data. The sequential nature of the data adds another dimension of complexity to the analysis, aside from understanding relationships across the feature dimensions, there are also relationships between observations. Successive observations are often highly correlated, with complex non-linear interactions occurring between features over time. Aside from all this, data is often irregularly sampled and partially observed and the best approaches for dealing with this are problem dependent.

The signature method refers to a collection of feature extraction techniques for multidimensional stream data that make use of the (log-)signature transform (as introduced in Sections 2.4 and 2.4.2. The stand alone signature transform applied to a sequential data stream is the simplest example of the signature method. However, the method has been adapted and evolved over time by users seeking to improve the quality of the extracted features than those that would be returned through a base application of the transform alone.

The stand alone signature transform, as introduced in Chapter 2, is the simplest example of the signature method. However, the method has evolved over time with many papers being published that differ from the base signature transform by including a multitude of different processing operations on the data before (and after) application of the (log-)signature transform, with the aim being to improve the quality of the features generated by the signature for the problem at hand.

A high level overview of the signature method, in the context of a machine learning classification pipeline, is given at the beginning of this chapter Figure 6.0.1. Raw tick data is first channelled through a series of transformations; these transformations represent the different modifications that can be applied to very the resulting features. The (log-)signature transform is then applied to the transformed signal, before application of a classifier to return a prediction.

Benefits of the signature method include: a high degree of flexibility, making it possible to customise the method to specific datasets; strong theoretical guarantees; an interpretable feature set; ease of handling irregularly sampled and/or partially observed data; and it being well-defined for some highly irregular processes such as ARMA, Gaussian processes or even Brownian motion. Also, signature features do not need to learned, which can make them particularly effective on (but not limited to) small datasets.

The flexibility of the signature method has made it possible to be tailored to specific applications and achieve state-of-the-art performance in wide range of problem domains, such handwriting recognition (Wilson-Nunn et al., 2018; Yang et al., 2016b), action recognition Yang et al. (2016a, 2017), and medical time series prediction tasks (Morrill et al., 2019, 2020b). However, this flexibility comes at the cost of additional complexity in the model search space.

To the best of our knowledge, no comprehensive studies exist that collate and combine the most common method variations found in the literature and assemble them under a common mathematical framework. Additionally, no baseline signature model has ever been tested against other time series classification baselines. Our goal will be to address both of these issues, alongside building an open source implementation of the signature method in code, so as to both improve our understanding of signature variations and to make the methods more accessible to a wider audience.

6.2 Previous Applications of the Signature Method

The signature transform has been used in a wide range of applications in machine learning predictive tasks. For example, the signature has been used as a feature extraction layer in classifiers that achieved state-of-the-art performance at the time of publication, for both Arabic (Wilson-Nunn et al., 2018) and Chinese (Yang et al., 2016b) handwriting recognition. Similarly, signatures features were successfully used in human action recognition by Li et al. (2017); Yang et al. (2017); Liao et al. (2019). In the medical domain, signature features were used as part of the top performing model at the Physionet 2019 challenge for prediction of sepsis (Reyna et al., 2019; Morrill et al., 2019, 2020b). Other applications involve finance (Lyons et al., 2014; Arribas, 2018), mental health (Kormilitzin et al., 2017; Arribas et al., 2018), and emotion recognition (Wang et al., 2019, 2020a).

These applications have prompted the creation of high performance software for computing the signature and logsignature transforms (Reizenstein and Graham, 2018; Kidger and Lyons, 2021).

In almost all these applications, the method has been utilised in different ways. Many authors consider transformations of the input time series before application of the signature (Levin et al., 2013; Flint et al., 2016; Lyons and Oberhauser, 2017; Yang et al., 2017; Liao et al., 2019; Kidger and Lyons, 2021; Wu et al., 2020). People have also explored different windows over which the signature transform should be taken, so as to extract information over different scales (Yang et al., 2017; Bonnier et al., 2019). Additionally a choice must be made between the signature and logsignature transforms, as must choices for the scaling of the terms in the signature (Kormilitzin et al., 2016; Lai et al., 2017).

The differences between some of these choice have been shown by Fermanian (2019) to significantly impact the performance of the methodology. However this study used a small collection of datasets and considered only some of the most common variations that exist in the literature. There is therefore a need for a comprehensive study and unification of all these different choices.

6.3 Components of the signature method

In this section we collate the modifications to the signature transform that have been proposed in signature literature to date. We will show that each can be categorised into one of the following groups:

- **Augmentations** These describe the transformation of a time series into one or more new series, in order to return different information in the signature features and deal with dimensionality issues.
- **Windows** Splitting the time series over different subsequences (or windows), so that signatures may be applied locally.
- **Transform** The choice between the signature or the log-signature transformation.
- **Rescaling** Ways of normalising the terms in the signature.

We then go on to show that these groupings can themselves be synergised into a single mathematical framework that we term *the generalised signature method*. For clarity, we will begin by discussing each of these individually, and then afterwards show how they may be combined.

As usual, we assume we observe some tick data $\mathbf{x} \in \mathcal{S}(\mathbb{R}^d)$ (as in Definition 2.2.1) with associated time stamps $\mathbf{t}$.

6.3.1 Augmentations

We define an augmentation to be a transformation of tick data $\mathbf{x}$ into a new but related set of tick data $\widehat{\mathbf{x}}$.

6.3.1.1 Add Time

Recall the definition from Section 1 that $\mathbf{t}$ denotes a sequence of increasing time stamps, we define the *add time* augmentation to be the map $\phi_t : \mathcal{S}(\mathbb{R}^d) \to \mathcal{S}(\mathbb{R}^{d+1})$ such

$$\phi_t(\mathbf{x}) = \big((t_0, x_0), \dots, (t_n, x_n)\big). \tag{6.1}$$

Application of this augmentation supplies information about the time parameterization of the series, it gives us information on the speed at which the sequence was traversed. Moreover, this is enough to guarantee uniqueness of the signature Hambly and Lyons (2010).

6.3.1.2 Lead-Lag

Here we follow a definition similar to that given in Gyurkó et al. (2013). Given $\mathbf{x} \in \mathcal{S}(\mathbb{R}^d)$ with length n, we define $\mathbf{x}^{\text{lead}} \in \mathcal{S}(\mathbb{R}^d)$ as,

$$x_j^{\text{lead}} = x_{\lceil j/2 \rceil}, \text{ for } j \in \{0, \dots, 2n-1\} \tag{6.2}$$

Table 6.3.1 – Summary of the different augmentations. In this table e denotes the dimension of the paths produced by the augmentation and p denotes the number of paths produced.

	e	p	**Property**
Fixed augmentations			
None	d	1	
Time	$d+1$	1	sensitivity to parametrization, uniqueness of the signature map
Invisibility-reset	$d+1$	1	sensitivity to translation
Basepoint	d	1	sensitivity to translation
Lead-lag	$2d$	1	information about quadratic variation, uniqueness of the signature map
Coordinates projection			dimensionality reduction
1	2	d	
2	3	$d(d-1)$	
3	4	$d(d^2-1)$	
Random projections	e	p	dimensionality reduction
Learnt augmentations			
Learnt projections	e	p	data-dependent and linear
Stream-preserving neural network	e	1	data-dependent

and $\mathbf{x}^{\text{lag}} \in \mathcal{S}(\mathbb{R}^d)$ as,

$$x_j^{\text{lag}} = x_{\lfloor j/2 \rfloor}, \text{ for } j \in \{0, \dots, 2n-1\} \tag{6.3}$$

We are now in a position to define the lead-lag transform $\phi_{\text{lead-lag}} : \mathcal{S}(\mathbb{R}^d) \to \mathcal{S}(\mathbb{R}^{2d})$ as follows

$$\phi_{\text{lead-lag}}(\mathbf{x}) = \big((x_0^{\text{lead}}, x_0^{\text{lag}}), \dots, (x_{2n-1}^{\text{lead}}, x_{2n-1}^{\text{lag}})\big) \tag{6.4}$$

6.3.1.3 Invisibility Reset

First introduced by Yang et al. (2017), the invisibility-reset augmentation consists in adding a coordinate to the sequence $\mathbf{x}$ that is constant equal to 1 but drops to 0 at the last time step, i.e.,

$$\phi_{\text{ir}}(\mathbf{x}) = \big((1, x_1), \dots, (1, x_n), (0, x_n), (0, 0)\big) \in \mathcal{S}(\mathbb{R}^{d+1}).$$

This augmentation adds sensitivity to the absolute value of the initial position of the path. The downside of this augmentation is that it results in an additional channel which typically means we will go less deep in the signature.

6.3.1.4 Basepoint

We define the basepoint augmentation as a map $\phi_b : \mathcal{S}(\mathbb{R}^d) \to \mathcal{S}(\mathbb{R}^d)$ that appends the zero-vector to the start of the path. We define this as

$$\phi_b = (0, x_0, \dots, x_n). \tag{6.5}$$

Similarly to invisibility reset, this provides sensitivity to the absolute value of the path. However, basepoint does not require the introduction of an additional channel. The price of this is that the signature of the path is not perfectly preserved due to the introduction of the zero-vector. In practice, for normalised data, this does not appear to matter much and is often preferable to the introduction of the additional channel that appears in the invisibility-reset augmentation.

6.3.1.5 Coordinate Projections

Often it may be desirable to compute the signature for a subset of the path coordinates. This reduces the dimension of the computation, but of course restricts the signature to only produce information on interactions between features in the chosen subset.

Let $\mathbf{x}^i$ denote the i^{th} coordinate of $\mathbf{x}$. We define the single coordinate projection as

$$\phi_{\text{cp1}}(\mathbf{x}) = \big((\mathbf{t}, \mathbf{x}^1), (\mathbf{t}, \mathbf{x}^2), \dots, (\mathbf{t}, \mathbf{x}^d)\big) \in \mathcal{S}(\mathbb{R}^2)^d,$$

whilst considering all possible pairs of coordinates yields the augmentation

$$\phi_{\text{cp2}} = \big((\mathbf{t}, \mathbf{x}^1, \mathbf{x}^2), (\mathbf{t}, \mathbf{x}^1, \mathbf{x}^3), \dots, (\mathbf{t}, \mathbf{x}^d, \mathbf{x}^{d-1})\big) \in \mathcal{S}(\mathbb{R}^3)^{d(d-1)},$$

and all possible triples yields the augmentation

$$\phi_{\text{cp3}}\big((\mathbf{t}, \mathbf{x}^1, \mathbf{x}^1, \mathbf{x}^2), (\mathbf{t}, \mathbf{x}^1, \mathbf{x}^1, \mathbf{x}^3), \dots, (\mathbf{t}, \mathbf{x}^{\mathbf{d}}, \mathbf{x}^d, \mathbf{x}^{d-1})\big) \in \mathcal{S}(\mathbb{R}^4)^{d(d^2-1)}.$$

Time may or may not be included, we have included it here so that the a meaning is ascribed to the singleton projections.

6.3.1.6 Random projections

Lyons and Oberhauser (2017) propose projecting the path onto a subspace by applying multiple random projections, this has particular utility when the dimension of the input stream is large. Let A denote p different affine transformations indexed by i such that $A_i : \mathbb{R}^d \to \mathbb{R}^e$ with $e < d$. We define the random projection augmentation as

$$\phi_{\text{rp}}(\mathbf{x}) = \big((A_1 x_1, \dots, A_1 x_n), \dots, (A_p x_1, \dots, A_p x_n)\big) \in \mathcal{S}(\mathbb{R}^e)^p.$$

6.3.1.7 Learnt projections

Liao et al. (2019) perform a similar process as in random projections, except they opt to learn the projection from the data. The form is that same as above but with A_i being learnt.

6.3.1.8 Learnt augmentation

In Bonnier et al. (2019) the authors introduce the idea of a learnt augmentation. This is simply a map

$$\phi_{\text{learnt}} \colon \mathcal{S}(\mathbb{R}^d) \to \mathcal{S}(\mathbb{R}^e),$$

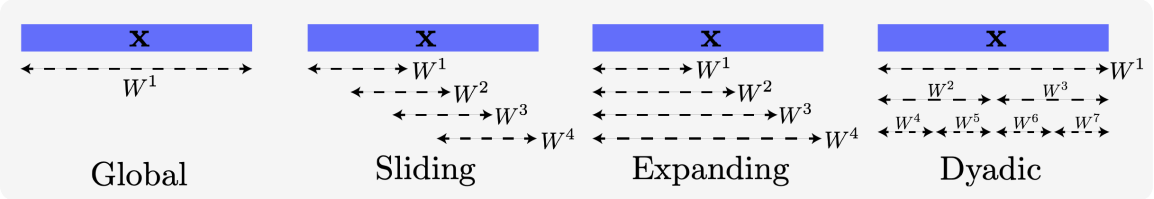


Figure 6.3.1 – Illustration of the four windowing operations.

where the mapping is learnt and parameterised via a neural network.

We summarise these augmentations in Table 6.3.1

6.3.2 Windows

Computing a single signature transform globally across the domain is very unlikely to produce optimal results. It is generally preferable to apply multiple signature transforms across the domain, so as to extract features of the time series over different data localities. To formalise this idea, we define a window to be a map

$$W : \mathcal{S}(\mathbb{R}^e) \to \mathcal{S}(\mathbb{R}^e)^w,$$

for some $w \in \mathbb{N}$. W maps a single time series in $\mathbb{R}^e$ into a collection of W time series also in $\mathbb{R}^e$. For example, splitting the time series at the halfway point producing two such series would be a perfectly valid operation.

The simplest possible windowing operation is to leave the time series as it is. We call this the *global* windowing operation, defined by

$$W(\mathbf{x}) = (\mathbf{x}). \tag{6.6}$$

Two common windowing operations to extract information over different scales are known as *sliding* and *expanding* windows. For a sliding window, given a fixed length L and a step size s, we slide a window of length l along the data in steps of length s, that is

$$W(\mathbf{x}) = (\mathbf{x}_{0:L}, \mathbf{x}_{L:L+s}, \mathbf{x}_{L+s:L+2s}, \ldots).$$

An expanding window instead of sliding with step s, expands over the data with step s, such that the window grows by length s at each iteration. This can be written as follows,

$$W(\mathbf{x}) = (\mathbf{x}_{0:L}, \mathbf{x}_{0:L+s}, \mathbf{x}_{1:L+2s}, \ldots).$$

The final operation we will consider is lesser known, we term it the *dyadic window*. Let $q \in \mathbb{N}$ and suppose 2^q divides Length$(\mathbf{x})$ (if it does not, in practice we will truncate the series at the largest subsequence with length divisible by 2^q). Letting L denote Length$(\mathbf{x})$, the dyadic window can be given as

$$W^q(\mathbf{x}) = \Big(\mathbf{x}, \big(\mathbf{x}_{0:\frac{L}{2}}, \mathbf{x}_{\frac{L}{2}:L}\big), \big(\mathbf{x}_{0:\frac{L}{2^2}}, \mathbf{x}_{\frac{L}{2^2}:\frac{2L}{2^2}}, \mathbf{x}_{\frac{2L}{2^2}:\frac{3L}{2^2}}, \mathbf{x}_{\frac{3L}{2^2}:L}\big), \\ \ldots, \big(\mathbf{x}_{\frac{iL}{2^q}:\frac{(i+1)L}{2^q}} \text{ for } i \in (0, 1, \ldots, q)\big)\Big). \tag{6.7}$$

In other words, it is the collection of sliding windows of length $L/2^n$ and step $L/2^n$ for n ranging from 1 to q.

One glaring omission from these windowing operations is an exponentially weighted window. That is where we first apply $X_t = X_t e^{-\lambda t}$ so that more recent observations are given more weight and thus have more impact on the signature output. This was a mistake on our point and something we should have included. Whilst it does not appear in the remainder of this chapter, we mention it so as to recommend it still be considered in signature applications.

An illustration of each of these windowing operations is presented in Figure 6.3.1.

6.3.3 Transforms

The core of the signature method is of course the signature transform itself. For this we have only two choices to make. The first choice is between the signature and the log-signature. As mentioned, the signature transform will result in an increased number of features for the same information, however it has the property of universal nonlinearity which may make these features easier to learn from.

Given a transform, we must also decide on the truncation depth. The higher the depth, the more information returned, but at a cost of an increased number of features which can add noise. The depth is highly problem dependent, we suggest that normally it should be treated as a hyperparameter to be optimised over.

6.3.4 Rescaling

In Chapter 2, Proposition 2.4.4, we gave the factorial decay property of the signature; that is, the depth k-term in the signature is of size $\mathcal{O}(1/k!)$. Typically, rescaling these terms to $\mathcal{O}(1)$ will aid in subsequent learning procedures. The most straightforward approach to normalise the terms is therefore to multiply the signature terms at depth k by $k!$. We term this *post-signature scaling*.

An alternative approach is to apply a pre-signature scaling. Here we multiply the input data $\mathbf{x}$ by α resulting in a k^{th} term of size $\mathcal{O}(\alpha^k/k!)$. Supposing we truncate the signature to depth-N, then taking $\alpha = (N!)^{1/N}$ will result in the depth N terms being $\mathcal{O}(1)$. Note that Stirling's approximation implies that the $N/2$-th term will be of size $\mathcal{O}(2^{N/2})$ meaning that terms between-depths are not of the same order. This may cause problems for some solvers, in particular for situations where we are considering a high depth.

6.4 The generalised signature method

Let $\phi : \mathcal{S}(\mathbb{R}^d) \to \mathcal{S}(\mathbb{R}^e)$ denote a concatenation of p augmentations such that

$$\phi \mapsto \big(\phi_1(\mathbf{x}), \dots, \phi_p(\mathbf{x})\big). \tag{6.8}$$

Let $W : \mathcal{S}(\mathbb{R}^e) \to (\mathcal{S}(\mathbb{R}^d))^w$ denote a concatenation of w windowing operations such that

$$W : \mathbf{x} \mapsto \big(W^1(\mathbf{x}), \ldots, W^w(\mathbf{x})\big),$$

Let S^N represent either the signature or logsignature transform of depth N. Let ρ_{pre} represent either the pre-signature scaling by α, or the identity. Let ρ_{post} represent either the post-signature scaling by $k!$ or the identity.

Now, we are in a position to define in abstraction the *generalised signature method* applied to a sequence $\mathbf{x}$ as

$$\mathbf{z}_{i,j} = (\rho_{\text{post}} \circ S^N \circ \rho_{\text{pre}} \circ W^j \circ \phi^i)(\mathbf{x}) \tag{6.9}$$

over all $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, w\}$ with $\mathbf{z}_{i,j}$ representing the resulting features.

6.5 Empirical Study

We perform a first-of-its-kind empirical study across 26 datasets to determine the most important aspects of this framework.

6.5.1 Methodology

Datasets The datasets used are the Human Activities and Postural Transitions dataset provided by Reyes-Ortiz et al. (2016), the Speech Commands dataset provided by Warden (2018), and 24 datasets from the UEA time series classification archive, provided by Bagnall et al. (2018). A few datasets from the UEA archive were excluded due to their high number of channels resulting in too large a computational burden.

Baseline We begin by defining a single baseline procedure, representing a simple and straightforward collection of choices for the generalised signature method. This baseline is to take the augmentation ϕ as appending time as defined by (6.1), W as the global window defined by (6.6), have the transform be a signature transform of depth 3, and to use pre-signature scaling of the path. This means that the input features are the collection

$$z = \text{Sig}^3 \circ \rho_{\text{pre}} \circ \phi_{\mathbf{t}}(\mathbf{x}).$$

Individual variations With respect to this baseline procedure, we then consider, in turn, the groups described in Section 6.4. These were *augmentations*, *windows*, *transform*, and *rescaling*. For each group we modify the baseline by implementing each option in the group one-by-one. Each such variation defines a particular form of the generalised signature method as in equation (6.9). Example variations are to switch to using a log-signature transform of depth 5, or to use a sliding window instead of a global window. We discuss the precise variations below.

Models On top of every variation, we then consider four different models: logistic regression, random forest, Gated Recurrent Unit (GRU) (Cho et al., 2014), and a residual Convolutional Neural Network (CNN) (He et al., 2016). We test nearly every combination of dataset, variation of the generalised signature method, and model. Different datasets and variations produce different numbers of features $\mathbf{z}_{i,j}$, so to reduce the computational burden we omit those cases for which the number of features is greater than 10^5. Of the 9984 total combinations of dataset, variation, and model, this leaves out 1415 combinations. See Appendix C.1.2 for a break down of the omitted combinations by different cases.

Analysis We define the performance of a variation on a dataset as the best performance across the four models considered, to reflect the fact that different models are better suited for different problems. We then follow the methodology of Demšar (2006); Benavoli et al. (2016); Ruiz et al. (2020) to compare the variations across the multiple datasets. We first perform a Friedman test to reject the null hypothesis that all methods are equivalent. Then, we perform pairwise Wilcoxon signed-rank tests to form cliques of not-significant methods, and use critical difference plots to visualize the performance of each signature method. A thick line indicates that the Wilcoxon test between methods inside the clique is not rejected at significance threshold of 5%, subject to Bonferroni's multiple testing correction.

We refer the reader to Appendix C.1 for further details on the methodology, such as precise architectural choices, learning rates, and so on.

6.5.2 Results

Due to the large number of variations and datasets considered, we present only the critical difference plots in the main paper. See Appendix C.2 for all the tables of the underlying numerical values.

Augmentations We split the augmentations into two categories. The first category consists of those augmentations which remove the signature's invariance to translation (basepoint augmentation, invisibility-reset augmentation) or reparameterisation (time augmentation). We see in Figure 6.5.1 that augmenting with time, and either basepoint or invisibility-reset, are both typically important. This is expected; in general a problem need not be invariant to either translation or reparametersiation.

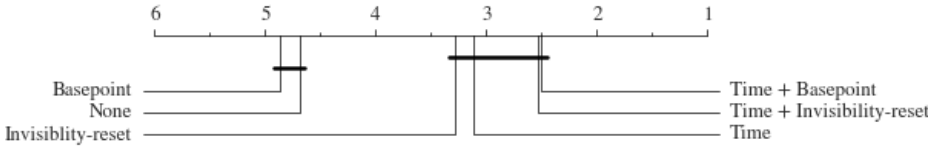


Figure 6.5.1 – Performance of invariance-removing augmentations.

The second category consists of those augmentations which either seek to reduce dimensionality or introduce additional information. We see in Figure 6.5.2 that most

augmentations actually do not help matters, except for lead-lag which usually represents a good choice. We posit that the best augmentation is likely to be dataset dependent, so we break this down by dataset characteristics in Table 6.5.1.

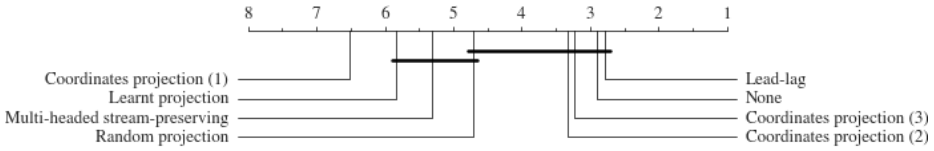


Figure 6.5.2 – Performance of other augmentations.

Table 6.5.1 – Average ranks for different augmentations by dataset characteristics. Lower is better.

	Augmentation							
Characteristic	None	Lead-lag	Coordinates projection (1)	(2)	(3)	Random Projection	Learnt Projection	MHSP
Data type								
EEG	4.88	4.83	6.50	3.13	5.67	4.38	**2.75**	**2.75**
HAR	2.25	**1.78**	7.20	3.50	2.90	4.75	6.50	6.50
MOTION	2.63	**1.75**	7.00	4.50	2.13	5.00	7.33	5.00
OTHER	2.88	3.92	5.44	**2.63**	3.29	4.69	6.00	5.21
Series length								
<50	3.20	**2.20**	7.40	3.20	2.70	5.10	7.00	5.20
50-100	2.20	**1.33**	6.00	4.10	2.75	6.20	4.80	5.10
100-500	**2.28**	2.57	7.28	3.50	2.63	4.17	6.63	5.33
>500	4.00	4.00	5.28	**2.64**	4.57	4.07	4.40	5.60
Dimension d								
2	4.67	3.5	6.33	4.33	4.0	**2.83**	6.67	3.67
3-5	2.5	**2.21**	6.36	3.43	3.14	4.64	6.67	6.83
6-8	3.25	**2.5**	6.94	3.0	3.56	5.0	5.29	6.0
>8	**2.25**	3.75	6.31	3.19	2.5	5.19	5.29	4.19

Here we indeed see that there is generally a better choice than doing nothing at all, but that this better choice is dependent on some characteristic of the dataset. For example on long or high-dimensional datasets, coordinate projections often perform well, whilst multi-headed stream preserving transformations do substantially better on EEG datasets. Lead-lag remains a strong choice in many cases.

We note that in these results we excluded datasets with dimension above 60 Appendix C.1. In such cases dimension reduction techniques would be essential as the computational cost of standard transforms would simply be too high.

Windows We consider the possibility of global, sliding, expanding, and dyadic windows. The results are shown in Figure 6.5.3. We see that the dyadic and expanding

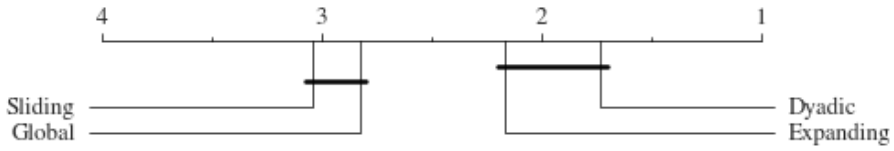


Figure 6.5.3 – Performance of different windows.

windows are significantly better than sliding and global windows. The poor performance of sliding windows is a little surprising, but tallies with the observations of Fermanian (2019). This is an important finding, as global and sliding windows tend to be commonly used with signature methods.

Signature versus log-signature transforms We consider the signature and log-signature transforms with depths ranging from 1 to 6. As higher depths always produce more information, we define the performance of the (log-)signature transform as the best performance across all depths. With this metric, the average rank of the signature transform was 1.23 whilst the average rank of the log-signature transform was 1.77, corresponding to a p-value of 0.01 with the Wilcoxon signed-rank test. Thus we find that the signature transform performs significantly better than the log-signature transform.

Rescaling We consider using no rescaling, pre-signature rescaling, or post-signature rescaling. We find that pre-rescaling performs worse than both post-rescaling and no rescaling within the significance level, however no significant difference between post-rescaling and no rescaling is found. This can be seen from the diagram in Figure 6.5.4.

We note though that this can largely be attributed to our using a random forest classifier, which being a tree based model is extremely robust to variations in scale. When using a model more sensitive to scaling, especially with large vales for signature depth (due to the factorial decay property), appropriate scaling is a must.

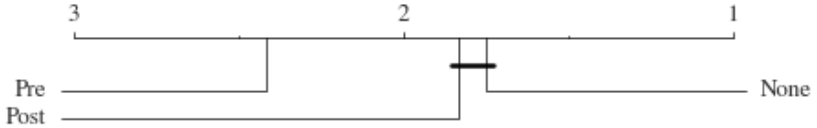


Figure 6.5.4 – Performance of different rescalings.

The key results To conclude, these results show that invariance-removing transformations such as time and basepoint augmentations should a priori be used, that the lead-lag performs well but not significantly better than no additional augmentation, and that the hierarchical dyadic window performs significantly better than

the sliding and global ones. These conclusions are the basis for the definition of the canonical signature pipeline, which we will present in the next section.

6.5.3 Further results

See Appendix C.2 for further results, in particular on the running times, sensitivity-inducing augmentations broken down by dataset type, an additional study on signature depth, and the precise numerical results for each individual test considered here.

6.6 A Canonical Pipeline

6.6.1 Definition

We are now in a position to define the canonical signature pipeline, illustrated in Figure 6.6.1. In a nutshell, the pipeline consists in applying the basepoint and time augmentations, a hierarchical dyadic window and a signature transform, which can be written as a particular case of (6.9) as follows. Let W be a hierarchical dyadic window of depth q, $\phi_{\mathbf{t}}$ and ϕ^b be the time and basepoint augmentations, then the canonical signature pipeline may be written as

$$y_j = S^N \circ W^j \circ \phi^b \circ \phi_{\mathbf{t}}(\mathbf{x}), \quad j \in \{1, \dots, 2^q - 1\}. \tag{6.10}$$

Both the signature and window depths, respectively N and q, must be optimised for the problem (typically via cross-validation). We note that this canonical method may be adapted to the problem at hand in two ways: if the problem is known to be parametrization invariant, as is the case for example for characters recognition, then the time augmentation should not be applied. Moreover, if the problem is translation-invariant, then the basepoint augmentation is not applied. We emphasise that this pipeline does not represent a best option for every application, but is meant to represent a compromise between broad applicability, ease of implementation, computational cost, and good performance.

6.6.2 Performance

To validate the performance of the canonical signature pipeline, we compare its performance to the results of Ruiz et al. (2020) who benchmark a collection of modern multivariate time series classifiers over 26 UEA datasets. Note that the multivariate UEA archive is not to be confused with the univariate UCR archive, which has received more attention. Moreover, the UEA archive has proved to be more challenging for producing performant benchmarks (Bagnall et al., 2018; Ruiz et al., 2020). For this comparison, we combine the feature set obtained with the canonical signature pipeline, summarized in (6.10) with a random forest—see Appendix C.1.3 for more details.

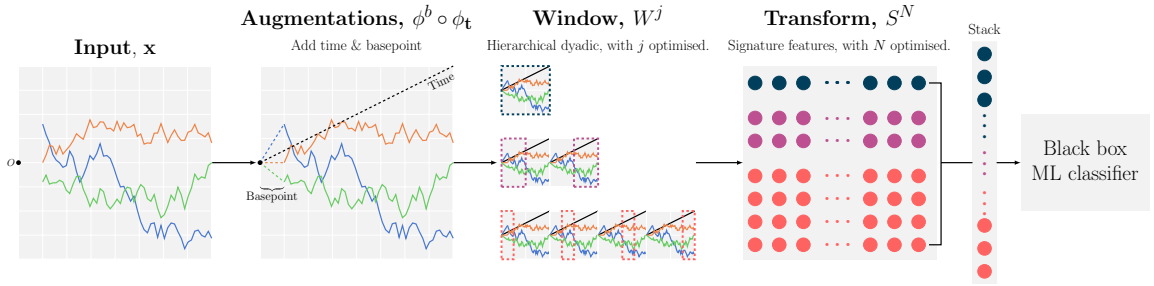


Figure 6.6.1 – Pictorial representation of the canonical signature pipeline. First, we apply the time and basepoint augmentations to the input paths, then we compute the signature features over dyadic windows, and finally compute the signature features over each dyadic window. These features can now be compiled together and fed into any standard machine learning classifier.

Our benchmarks are three variants of the classical Dynamic Time Wrapping transform combined with a one nearest neighbour algorithm (DTWI, DTWD and DTWA in Figure 6.6.2), the generalized Random Shapelet Forest, denoted by gRSF (Karlsson et al., 2016), two deep neural networks, MLCN (Karim et al., 2019) and TapNet (Wang et al., 2017), the ensemble HIVE COTE (Bagnall et al., 2020) and the WEASEL-MUSE algorithm (Schäfer and Leser, 2017). We refer the reader to Ruiz et al. (2020) for further details.

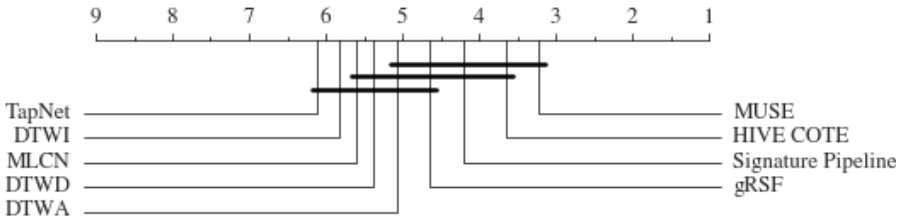


Figure 6.6.2 – Performance on UEA datasets.

Figure 6.6.2 shows the critical difference plot of this comparison. The signature pipeline is in the first clique, that is the group of classifiers that achieve the best accuracy while not being significantly different from one another. The two algorithms with a better rank than the signature pipeline are MUSE and HIVE-COTE. It is worth noting that MUSE is very memory intensive—Ruiz et al. (2020) report that it could not finish on 5 of the 26 UEA datasets on a computer with 500GB of memory—whilst HIVE-COTE is an ensemble of several sub-classifiers, and thus has very high training and inference costs. On the other hand, all experiments for the canonical signature pipeline were completed with no memory errors on a computer with less memory, and are significantly faster to run than HIVE-COTE—see Appendix C.2.

The canonical signature pipeline is meant to be a sensible starting point from which the user can propose additional variations following the structure defined in equation

(6.9), but as a standalone classifier this pipeline performs comparably to state-of-the-art classifiers,on the UEA data, whilst being less computationally demanding.

6.7 Open Source Implementation

This work has also resulted in the implementation of an open source implementation of "The Generalised Signature Method", made available and maintained in the open source `sktime` project. The implementation essentially packages the described pipeline into an sklearn style transformer allowing for easy application of the generalised signature approach followed by any standard machine learning classifier for tabular data.

We give an example use case in code 6.1. Here we initialise the `GeneralisedSignatureMethod` class and add input arguments that are ordered in the same manner as our components in Section 6.4. In the code example we implement the 'addtime' and 'basepoint' augmentations with a depth 3 dyadic window, pre-rescaling, and the signature transform to depth 3. A `RandomForestClassifier` is then fit to the resultant signature features and predictions are made on the test data.

We hope that this implementation will serve as an easy to use starting point for ML practitioners who are looking to use signatures, but are relatively new to the space. The sklearn style implementation enables the user to train over a variety of complex signature method pipelines with relatively little understanding about the underlying mathematics. Moreover, the initial values of the parameters are set to those found to provide a highly performant starting point from Section 6.10.

```python
# Load the GeneralisedSignatureMethod module in from sktime
from sktime.transformations.panel.signature_based import
↪  GeneralisedSignatureMethod
from sklearn.ensemble import RandomForestClassifier

# Load in some data
X_train, X_test, y_train, y_test = load_data()

# Setup the generalised method with chosen options
signature_method = GeneralisedSignatureMethod(
    augmentation_list=('addtime', 'basepoint'),
    window_name='dyadic',
    window_depth=3,
    window_length=None,      # Used only for expanding/sliding
    window_step=None,        # Used only for expanding/sliding
    rescaling='pre',
    sig_tfm='signature',
    depth=3
)

# The simply transform the stream data
X_train_signatures = signature_method.fit_transform(X_train)

# The it is easy to train a model on the new features
model = RandomForestClassifier()
model.fit(X_train_signatures, y_train)

# Make predictions on test data
X_test_signatures = signature_method.transform(X_test)
test_preds = model.predict(X_test_signatures)
```

Code 6.1 – Example code for using the open source sktime implementation of the generalised signature method.

6.8 Conclusions

In this section we have introduced the 'Generalised Signature Method' for multivariate time series data. This is a framework for working with signatures that combines the most common variations on the base signature transform under a common mathematical formalism. We hope that thinking about signatures in this way will both lower the barrier of entry to working with signatures, as currently the literature can seem quite disparate and the mathematics challenging to someone unfamiliar with signatures. We also hope that

We then went on to consider variations of the signature method to gain some insight as into the hyperparameters that produce the most between-dataset variation, and

the values of the hyperparamters that tend to perform best *on average*. These results were then used to define the 'Canonical Pipeline' which represents a best practice starting point for from which more complicated variations can be considered.

Finally we developed an open source implementation of the code that enables users to quickly build and optimise signature classification pipelines. The code is straightforward to use and requires minimal underlying knowledge of the components that make up the Generalised Signature Method, and as such, we hope it will encourage more people to use signatures in their machine learning models as well as engage with the signature community as a whole.

Chapter 7

Predicting Sepsis in the ICU with Signatures

This chapter is devoted to the development of our signature-based model for the PhysioNet 2019 Challenge titled The Early Prediction of Sepsis from Clinical Data. Our model achieved 1st place from models submitted by over 100 participating teams.

- *Morrill, J., Kormilitzin, A., Nevado-Holgado, A., Swaminathan, S., Howison, S., and Lyons, T. (2019). The signature-based model for early detection of sepsis from electronic health records in the intensive care unit.* International Conference in Computing in Cardiology

- *Morrill, J. H., Kormilitzin, A., Nevado-Holgado, A. J., Swaminathan, S., Howison, S. D., and Lyons, T. J. (2020b). Utilization of the signature method to identify the early onset of sepsis from multivariate physiological time series in critical care monitoring.* Critical Care Medicine, *48(10):e976–e981*

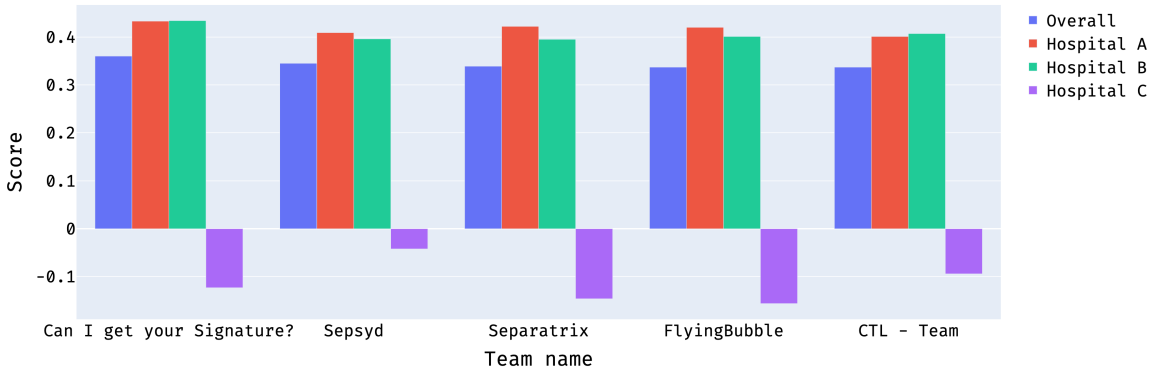


Figure 7.0.1 – Top 5 entries on all hospitals and their overall score. Our team 'Can I get your signature' is furthest left with the highest overall score.

7.1 Introduction

Sepsis is a life-threatening organ dysfunction caused by a dysregulated host response to infection. Sepsis management represents an extremely challenging problem throughout the world. Recent estimates suggest around 1.7 million adult Americans develop sepsis each year, of which nearly 270,000 dies, this accounting for one in three hospital deaths. A recent study Buchman et al. (2020a,b,c) evaluated the cost to Medicare of sepsis inpatient admission and subsequent patient care to be in excess of $41.5 billion, and found that both the numbers of sepsis-related claims and the aggregate dollar costs are rising. Additionally, it has been reported that mortality rates increase significantly with delays in receiving antibiotics. For example, Kumar et al. (2006) report that in cases of septic shock, the risk of death increases by $\sim$10% for every hour of delay in antibiotic treatment. Early detection of sepsis events is thus essential to improving sepsis management and mortality rates in the ICU.

Machine-learning algorithms have been investigated as a method for early prediction of sepsis Henry et al. (2015); Calvert et al. (2016); Fagerström et al. (2019). However, it is difficult to compare the different algorithms since each addresses a slightly different problem, utilises different data sets, maintains a different definition of sepsis onset, and uses different metrics. Furthermore, none use the new Sepsis-3 definition Singer et al. (2016) to denote the onset of sepsis.

Here, we describe the development of a machine-learning model for the early detection of sepsis based on readily attainable ICU data. The model was developed during the 2019 PhysioNet challenge entitled 'Early Prediction of Sepsis from Clinical Data' during which participants were invited to submit algorithms that were trained on the same set of data and validated under a common performance metric on unseen test data. Sepsis onset was defined using the Sepsis-3 criteria.

7.2 The PhysioNet 2019 Challenge Setup

Each year, beginning in 2000, PhysioNet host a challenge based on streams of complex physiological signal data and invite participants to tackle a clinically interesting problem related to the data that is either unsolved or not well-solved. 2019 marked the 20^{th} consecutive year of these challenges and was titled *Early Prediction of Sepsis from Clinical Data*. The official description of the challenge is given in Reyna et al. (2019).

The aim of the challenge was to develop a model that can early-detect sepsis 6 hours before its occurrence based on minimal data that is easy to obtain in an ICU. Further, the models were to be trained on data from two different hospital systems and tested on three to give a better indication of their generalisation performance than the models that have been developed previously which are trained and tested on data from a single system. In this section, we will provide an overview as to how the cases were labelled, the datasets that were used, and the challenge scoring metric.

In this section we describe the challenge set up and scoring metrics and our methodology for model building and training, before concluding with the final challenge results. In subsequent sections describe further ideas and analysis that arose during and after the challenge.

7.2.1 Sepsis Labelling

The challenge data is given pre-labelled with the value 1 for patients who develop sepsis where, with time being measured in hours, $t \geq t_{sepsis} - 6$ and 0 for $t < t_{sepsis} - 6$, with t_{sepsis} being the time of sepsis onset as defined by the Sepsis-3 definition Singer et al. (2016). For patients who never develop sepsis the data is labelled zero everywhere. To determine the location of t_{sepsis} the Sequential [Sepsis-related] Organ Failure Assessment (SOFA) score must be computed for each time-point Singer et al. (2016). This score to determines the extent of a person's organ function or rate of failure. Once this is computed, we note the following times:

$\underline{t_{\text{suspicion}}}$: Time of clinical suspicion of infection, identified as the earlier timestamp of IV antibiotic administration and the drawing of blood cultures within a specified duration. If antibiotics were given first, then the cultures must have been obtained within 24 hours. If cultures were obtained first, then antibiotic must have been subsequently ordered within 72 hours. In this case antibiotics must have been administered for at least 72 consecutive hours to be considered.

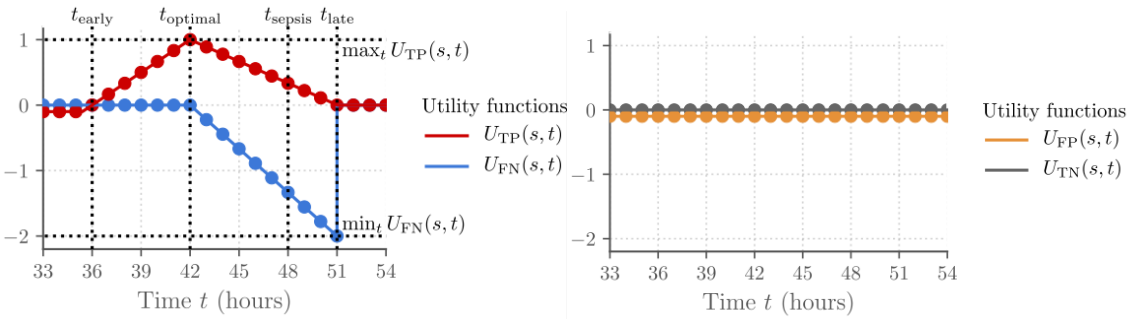
$\underline{t_{\text{SOFA}}}$: The occurrence of end organ damage (damage in major organs fed by the circulatory system) as identified by a two-point deterioration in SOFA score within a 24-hour period.

If both of these events occur and t_{SOFA} occurs no more than 24 hours before or 12 hours after $t_{\text{suspicion}}$ then time of sepsis is then defined to be the earlier of $t_{\text{suspicion}}$ and t_{SOFA}. If this does not happen, the patient is marked as non-septic. This is written more succinctly below:

$$t_{\text{sepsis}} = \min(t_{\text{suspicion}}, t_{\text{SOFA}}), \quad \text{if} \quad t_{\text{suspicion}} - 24 < t_{\text{SOFA}} \leq t_{\text{suspicion}} + 12. \tag{7.1}$$

7.2.2 Dataset

Data for this study was taken from the PhysioNet Challenge 2019. Full details are found in the official challenge paper Reyna et al. (2019). In brief, the data was sourced from ICU patients in three separate hospital systems. Data from two hospital systems was split into a publicly available training set and an undisclosed validation set, both to be used for model development and testing. The validation set and third hospital system data were kept private, with submitted models being scored against these unseen data from all three systems. A total of 40,336 patients were used in model training with 40 features per patient, consisting of demographic, vital sign, and laboratory data. The data for each patient was indexed with time at one-hour increments and predictions were to be made sequentially at each hour in a patient's



(a) Utility score for a septic patient.

(b) Utility score for a non-septic patient.

Figure 7.2.1 – Utility function scores in cases of sepsis and non-sepsis.

time-series with any future information being censored. The models were scored against the custom utility function defined in Reyna et al. (2019).

7.2.3 Scoring

Predictions are scored for their binary classification performance against a utility function defined specially for this challenge. Each patient is assigned a utility score which is the value generated on evaluation of the utility function. True positives are awarded a positive score, false negatives are penalised with a large negative score, false positives a small negative score and false negatives are awarded zero score.

This is done by defining a score $U(s,t)$ such that for each time t in the data:

$$U(s,t) = \begin{cases} U_{\text{TP}}(s,t), & \text{positive prediction at time } t \text{ for sepsis patient } s \\ U_{\text{FN}}(s,t), & \text{positive prediction at time } t \text{ for non-sepsis patient } s \\ U_{\text{FP}}(s,t), & \text{negative prediction at time } t \text{ for sepsis patient } s \\ U_{\text{TN}}(s,t), & \text{negative prediction at time } t \text{ for non-sepsis patient } s \end{cases}$$

In Figure 7.2.1 we plot the utility function values for a septic patient (left) and a non-septic patient (right). In this example, the patient has developed sepsis at $t_{\text{sepsis}} = 48$. We see that positive scores are achieved for predicting sepsis from 9 hours before up to 3 hours after the actual time of sepsis (red line). Scores are accumulated as we make more predictions in time. We also see that points are lost (blue line) for every missed case after t_{optimal}.

This information and the figures are taken from the official challenge paper Reyna et al. (2019), in which can also be found an extended description of the challenge scoring function.

Feature name	Description
ShockIndex	Heart Rate / Systolic Blood Pressure
BUN/CR	Bilirubin / Creatinine
PartialSOFA	Score of the SOFA components that are found in the challenge data
SOFA_Deterioration	Binary label given 1 if PartialSOFA has increased by 2 in the last 24 hours

Table 7.3.1 – Non-signature features extracted from the data.

7.3 Methodology

7.3.1 Data Imputation

Missing values during a patient's hospital stay were imputed by using a forward-fill method. When a value was missing, it was taken to be the most recent measurement of that particular variable. If no previous value existed the value was left as 'NaN', since decision tree based algorithms can handle such missing data effectively.

7.3.2 Non-Signature Feature Extraction

We augmented the data sets by including a number of additional features typically thought to be useful for discerning the onset of sepsis. These features, based on literature review and "expert knowledge", included a partial construction of the Sequential [Sepsis-related] Organ Failure Assessment (SOFA) score Singer et al. (2016), which we call PartialSOFA. The term "partial" was used because the data set did not include all variables that comprise the SOFA. Hence, the score was calculated based on the available required information which consisted of threshold conditions on each of the platelet count, bilirubin, mean arterial pressure, and creatinine variables. Given that one of the requirements of the Sepsis-3 definition is a deterioration of the SOFA score, any approximation of this quantity would be expected to aid in prediction of sepsis onset. Other added features included the ratio of heart rate to systolic blood pressure (the "shock index") and the ratio of bilirubin to creatinine as suggested in Henry et al. (2015). These features are summarised in Table 7.3.1.

Additionally to this, we extracted some temporal features that we believed would provide a useful measure of assessing patient health. Firstly, we hypothesised that the frequency of measurements would also provide an indication of patient health, given the anticipated increase in sampling when physicians are concerned about patients. For this reason we also included a counter for the temperature variable and the laboratory values, which denotes the number of times a given variable had been measured in some lookback window. We call the collection of these variables the 'measurement intensity' variables. Finally, we included the maximum and minimum of the vital signs over a lookback window. The size the lookback windows were treated

	Utility score		
Time	0	1	**Label**
1	0.00	-0.05	-0.05
2	0.00	-0.05	-0.05
3	-0.22	0.89	1.11
4	-0.44	0.78	1.22
5	-0.67	0.67	1.33
6	-0.89	0.56	1.44

Table 7.3.2 – Example of the case labelling given the utility score from predicting 0 and 1.

as hyper-parameters and optimised in model training and are given in Table 7.3.3.

7.3.3 Signature Feature Extraction

To extract additional longitudinal information from the time series, we turn to the signature transformation, as introduced in Section 2.4. A simple sliding window approach was used so that signature features were computed for each time-point over a window of some given look-back size. We computed the signatures of the 'PartialSOFA', 'MAP' and 'BUN/CR' variables using the 'addtime' augmentation (see Section ?). Additionally, we computed signatures of all non-stationary features with the 'leadlag' and 'cumsum' augmentations (see Section ?). The signature truncation levels and the lengths of the lookback windows were treated as hyperparameters to be optimised in model training.

7.3.4 Hyperparameters

In Table 7.3.3 we give the values of some of these hyperparameters that were used in the final submitted model. This table displays the lookback windows and signature depths for the (PartialSOFA, MAP and BUN/CR) variables and the ('cumsum', 'leadlag') signatures that were extracted for all non-stationary features, along with the lookback windows for the derived features.

7.3.5 Sepsis Labels

The data was pre-labelled with the value 1 at any location of sepsis occurrence or predefined window around sepsis onset, and zero otherwise. Given that the aim was to optimise the utility score (as defined in Reyna et al. (2019)), not simply the percentage of correct binary predictions, we created an alternative labelling that accounted for information about the utility score and enabled the classifier to place higher weights on points that lead to a larger score if predicted correctly. Let $U_y(s,t)$ denote the utility score of predicting $y \in \{0,1\}$ for patient s at time t. The "modified utility

Parameter	Final Value
Measurement intensity lookback	8 hours
Min/max lookback	8 hours
(PartialSOFA, BUN/CR, MAP) signature lookback	7 hours
(PartialSOFA, BUN/CR, MAP) signature order	3
Cumsum leadlag signature lookback	7 hours
Cumsum leadlag signature order	3

Table 7.3.3 – List of all hyperparameters with the values used for each in the final model.

		Hospital			
Rank	**Team Name**	A	B	C	**Final Utility Score**
1	Can I get your Signature?	**0.433**	**0.434**	-0.123	**0.360**
2	Sepsyd	0.409	0.396	**-0.042**	0.345
3	Separatrix	0.422	0.395	-0.146	0.339
4	FlyingBubble	0.420	0.401	-0.156	0.337
5	CTL-Team	0.401	0.407	-0.094	0.337
6	SBU	0.408	0.402	-0.154	0.332
7	Ping An Health Technology	0.414	0.400	-0.182	0.331
8	prna	0.411	0.389	-0.159	0.328
9	Antanas Kascenas	0.406	0.397	-0.195	0.323
10	NN-MIH	0.414	0.373	-0.174	0.323

Table 7.3.4 – Top 10 official scores from the final phase of the challenge scored on the hidden test set. Bold indicates best performance for the given hospital/overall.

score" is defined as $U_{MUS}(s,t) = U_1(s,t) - U_0(s,t)$ and constitutes the case labelling against which the regressor was trained. This allowed for increased weighting on cases that yield a larger utility difference if predicted correctly. An example of this relabelling process is given in Table 7.3.2.

7.3.6 Model Training and Validation

We used a stratified 5-fold cross validation with a uniform distribution of time-points and sepsis labels in each fold for hyper-parameter optimisation of the predictive algorithm. No patient had data in more than one fold. We used the LightGBM implementation Ke et al. (2017) of the gradient boosting machine algorithm with the mean square error loss function for the regression task. Hyper-parameters were chosen to maximise the mean utility score over the 5-folds.

	Hospital			
	A	B	C	**Average (std)**
Training	0.442	0.421	—	0.434 (0.016)
Validation	0.433	0.434	−0.123	0.360

Table 7.4.1 – Scores achieved on each hospital for both the public training set and hidden test set. The public training set scores are taken to be the average utility over the 5-folds used in cross-validation. We see that generalisation is good when tested on data from the same hospital system, but extremely poor on a different hospital system (hospital C).

7.4 Results

The submitted model was the 1st-place official entry to the challenge with an average utility score of 0.360. A detailed breakdown of the top 10 performing models is given in Section 7.3.6.

We additionally present the scores our model achieved for each hospital both for the public training data and the unseen validation data in Section 7.3.6. The scores for the public training data are taken to be the average utility over the 5-folds used in cross-validation. One can see that the algorithm scores on hospitals A and B in the training and validation sets are similar, which suggests that our model was not over-fitted when restricted to testing on the same hospital system as that on which the model was trained. Conversely, the scores on hospital C were significantly worse, which highlights the potential danger of using an algorithm that was trained on one hospital system to make predictions on a different one.

7.4.1 The Usefulness of Signatures

Table 7.4.2 shows the cross-validated and averaged utility score predictions on the training data for different subsets of features. Note that the score for each row is from the algorithm containing the features listed for that row, along with any features that are listed above it. The data indicates that the time since admission feature alone can be used to achieve a utility score of 0.282; this feature was, indeed, found to be the most important. Of particular further interest is the additional benefit to be gained from supplementing the dataset with the signature features. We see that the addition of the signature terms increases the utility score from 0.422 to 0.434 and this, while not a dramatic improvement, does show that the representation of the information fed into the algorithm after the signature transformation is beneficial to learning, and thus worthy of inclusion in comparable modelling approaches for similar problems.

Features	Averaged utility score
Time only	0.282
+ Original 40 features	0.389
+ Non-signature hand-crafted features	0.422
+ Signature features	0.434

Table 7.4.2 – Scores from models trained on different subsets of features. Each row also contains the features listed in any rows above it.

7.4.2 Predictions in an In-Hospital Environment

While the goal of the challenge was to optimise the score achieved on the pre-defined utility function, it is useful to consider how the model could be used in an in-hospital environment to provide clinically actionable information. At each time-point the model outputs a value with increasing values associated with higher risk of sepsis. When the model output exceeds a specified operating-point threshold, the subject is designated as a *sepsis-risk patient*, thus indicating to a doctor that closer monitoring or further tests may be warranted. This operating-point threshold can be chosen to achieve the most clinically meaningful sensitivity and specificity. The area under the receiver operating characteristic (ROC) curve of the optimised model was 0.868. A closer analysis is displayed in Figure 7.4.1. In the leftmost plot we have a confusion matrix, where a true negative represents that no call was made and the person didn't develop sepsis, and a true positive represents that the patient being flagged and developing sepsis *at some point* after this call. The operating-point threshold is set so that 33% specificity (the proportion of correct positives that were correctly identified) is achieved. On the right of the figure we give a performance analysis in increasing time-windows before onset; for example, the first bar gives the number of correct predictions made one or more hours in advance at 20% precision.

7.5 Discussion

The signature-based model that we have presented for the early prediction of sepsis constitutes a competitive approach to discerning early onset sepsis from health data streams. The model achieved the highest score in the official phase of the challenge Reyna et al. (2019). We believe the strength of our model lies primarily in the smart collection of hand-crafted features derived from the data, such as 'Partial-SOFA', along with effective utilisation of the signature method to extract additional features from the time-series. The non-signature features were not unique to our submission, however, the signature provided us with an extra piece of technology that allowed us to represent the longitudinal multidimensional clinical data as a collection of low-dimensional characteristic features that were not considered by other teams. In addition to this, training against the utility score as opposed to the provided binary labels for sepsis, and choice of classifier paired with an extensive hyperparameter

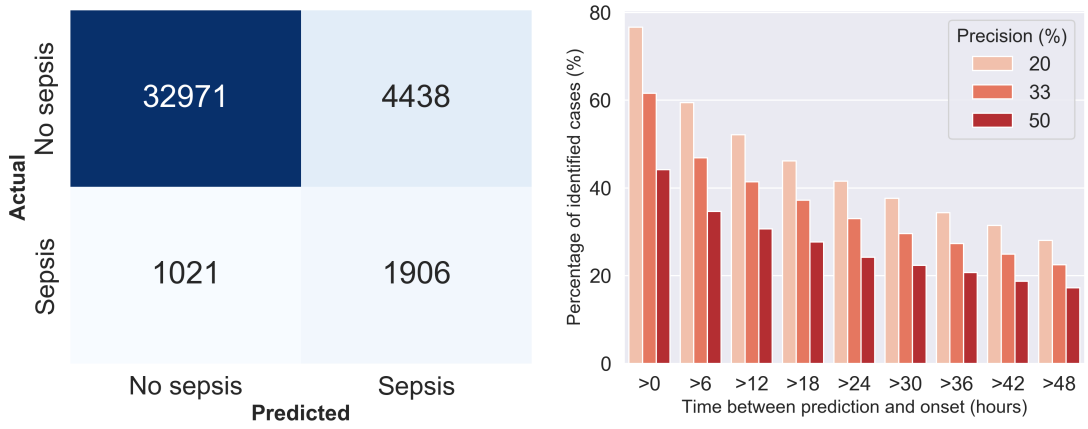


Figure 7.4.1 – Confusion matrix displaying the number of people predicted as likely to get sepsis compared with those who actually end up with sepsis with the threshold tuned to 33% specificity (left). Proportion of sepsis cases predicted correctly in different time windows tuned to different precision levels (right).

search were two other things that we found made significant improvements to the score. Despite the success of this method with comparison to other teams, there are a few limitations and caveats to be addressed.

7.5.1 Inability to Make Predictions in the Desired Window

First, one of the primary aims of the PhysioNet/CinC challenge was to develop a model to make accurate predictions in a window around 6 hours prior to the onset of sepsis, as this was determined to be the most clinically relevant region for plausible intervention. However, we see from Figure 7.4.1 (right) that the majority of predictions were not made in this desired time window. Similar long-time predictions have been found in other studies such as Henry et al. (2015), in which the governing model predicted sepsis shock with a median time between prediction and onset of 28.2 hours, and more recently Fagerström et al. (2019) demonstrated a median time-to-event prediction of 48.0 hours. Even when training against the utility function, which is biased towards making predictions in the desired 6-hour time-window, out model makes predictions with a median time of 25.0 hours in advance (at 33% specificity). In Buchman et al. (2020b) Buchman et al. provide evidence suggesting that sepsis may often occur as a result of a manifestation of some other chronic conditions, as opposed to a 'bad luck' event from a more ordinary infection. This hypothesis would make sense in the context of these model predictions, with the conclusion being that the model may be identifying characteristic features of chronic conditions from which sepsis is likely to manifest. A simple example being high-temperature correspondent to a fever from which sepsis can develop. This would explain why predictions are being made in advance of where the symptoms of sepsis are expected to have developed.

At different precision levels we find that, on average, around two-thirds of predictions

were made over 12 hours in advance, which suggests that the algorithm is performing a risk-stratification procedure rather than making a precise temporal prediction of sepsis onset. Similar long-time predictions have been found in other studies such as Henry et al. (2015), in which the governing model predicted sepsis shock with a median time between prediction and onset of 28.2 hours. A more recent paper Fagerström et al. (2019) demonstrated a window of 48 hours. Even when training against the utility function, which is biased towards predictions in a window close to onset, our model still struggled to predict in the desired window. Buchman et al. (2020b) find evidence that suggests sepsis may often occur as a manifestation of other chronic conditions. Assuming this to be true, this suggests that our model (and other models of sepsis) are likely identifying characteristic features of chronic conditions from which sepsis is more likely to manifest from, for example, a high-temperature may correspond to a fever, from which sepsis can develop from. When such features can be identified, the model associates a high risk of sepsis, even though sepsis has not yet occurred.

7.5.2 Uptake of Sepsis Appears to Follow a Poisson Process

In Figure 7.5.1 we show the number of people that have not yet developed sepsis as a fraction of the subset of people who do eventually develop it. This is plotted for both hospitals in the training set (blue solid lines) along with a best fit exponential (orange dashed lines). The figure shows that the fitted trend line matches the true data very closely. Given that the population level curve shows a clean exponential decay, it could be that the likelihood of any individual patient developing sepsis is dependent on a variable or variables that are governed by a Poisson process. Alternatively stated, there could be random events at play that are difficult to resolve with our modelling formalism. One could consider, for example, that sepsis arose out of the accidental use of a dirty needle. The point at which a dirty needle is used is almost impossible to predict (assuming it is a true Poisson process). If these events happened far enough in advance of onset, one could feasibly expect to build a model to make a positive prediction in a sensible time-window prior to the emergence of sepsis if the feature data could be collected; alternatively, the time-scale between such events and sepsis onset could be very short making predictive models difficult. An increase in the data sampling rate could be beneficial as 1-hour bins would likely not contain the required resolution to accurately resolve all of the relevant features.

7.5.3 The Model Achieves Poor Generalisation Performance to an Unseen Hospital

Finally, as previously noted, the models inability to maintain high prediction performance on hospital C indicates a limitation in generalising predictions to hospital data-sets on which the algorithm was not trained. There are a number of potential reasons as to why this might be the case: For example, the models were trained on

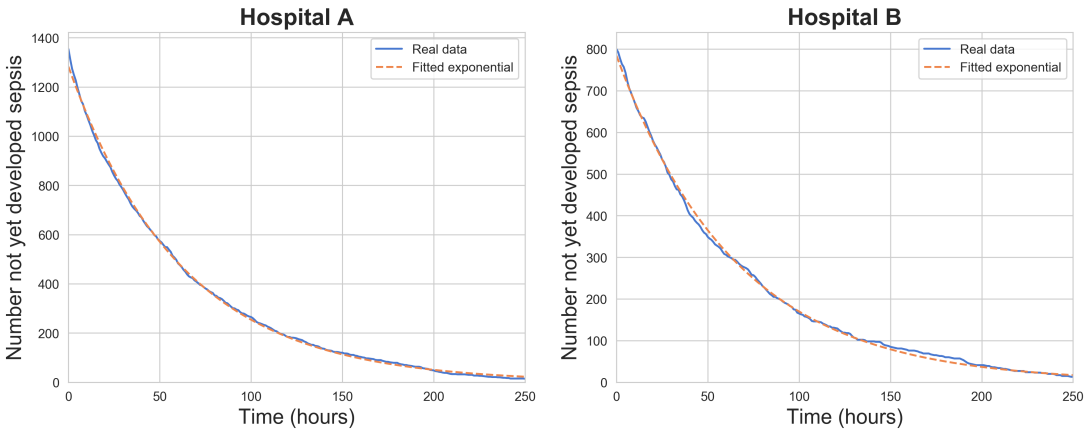


Figure 7.5.1 – The number of people who have not yet developed sepsis at each time, as a fraction of those who develop sepsis eventually, for each of the hospitals in the training set.

variables that were highly dependent on physician decision-making processes and thus local hospital policies. A measurement or assessment that is encouraged to be taken by a doctor in one hospital may not comport with the practices of another hospital. As such, any model that is trained of data-sets from healthcare systems in which a gold standard for assessment procedures and measurements does not exist, will inherently adopt some of the biases of the underlying training set. One remedy to this limitation would be to only train on variables that are sampled at pre-determined times of a patients stay and are independent of a doctor decision-making process. One could then expect to more safely transfer an algorithm from across multiple hospital systems. Any remaining uncontrolled variables between hospitals such as patient demographics, socioeconomic factors, etc would, of course, persist as sources of error.

7.5.4 The potential for semi-supervised learning

The poor performance on the out-of-sample hospital, hospital C, is likely caused due to differing practices between the hospitals, and thus the models reliance on features and relationships observed in hospitals A or B that do not pass across to hospital C. One idea for mitigating this effect is to first train a semi-supervised model to learn features that exist within all hospitals. One sensible approach would be to take data from each hospital and train a model to simultaneously maximise classification accuracy, and require that we cannot distinguish the hospital that the prediction features came from. This could be achieved by implementing a neural network with a low dimensional feature encoding layer, from which we build a discrimination network and a classification network. The models are then trained to maximise classification whilst reducing ability for discrimination.

The result of this would provide us with a model that is significantly more generalisable than the models that were trained in this section. If the performance of the

models remained high, it opens up the feasibility of safe deployment in hospitals from which the model has not received training data from – this is something that has certainly not been achieved by our current iteration.

7.6 Conclusions

We have presented a signature-based model for early prediction of sepsis. The signature representation produced a useful summary of the longitudinal physiological measurements that were used to effectively discriminate septic from non-septic cases. The addition of the signature terms improved the algorithms prediction performance significantly as demonstrated in Table 7.4.2. The method proposed has achieved the highest official score on the utility function on the full test data set from 104 submissions. We have further shown that model predictions can be turned into clinically-actionable information for use by doctors. Finally we explored: 1) why the model struggles to predict more cases in the desired time-window before onset, 2) the model's limited performance on new hospital systems and 3) some potential methodological enhancements for better generalising the model predictions.

Chapter 8

Improved Prediction of Stress Levels from Breathing Signals using Path Signature Features

Breathing is consistently ranked amongst the most predictive physiological markers for detecting stress levels. In previous work, simple features are extracted from the breathing signal over sliding windows (for example, mean inhalation time) and models are trained on the resulting features. These features are often highly interpretable but underutilise the raw signal since they throw away much of the information. Here, we describe a new approach utilising path signature features. Considering only low-order features we show that we can achieve a model on par with performance of existing approaches, whilst remaining interpretable. We then show that by increasing signature depth, where interpretability is sacrificed for additional features, the model achieves dramatic improvements over anything seen thus far in the literature.

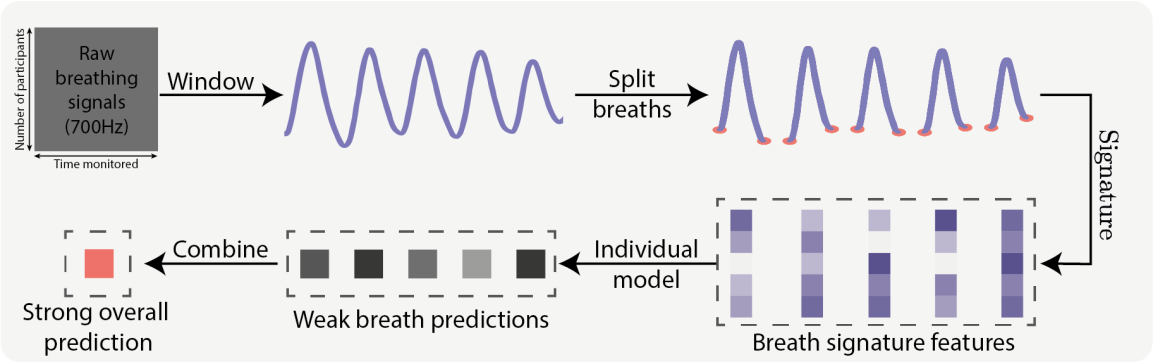


Figure 8.0.1 – A high level overview of the model we will introduce in this chapter. The raw breathing signal is first windowed, split into individual breaths, and then the signatures of each breath are computed. A model is trained on these signatures and the outputs are combined to produce an overall prediction.

8.1 Introduction

Emotion AI (or *affective computing*) refers to the study and development of systems that can classify human emotion; the end goal being to improve human-computer interactions. Empathetic machines would have the ability to detect emotions from sensor data and adapt their behaviour accordingly to suit the needs of the user(s). We use stress as a general term for a wide range of strong external stimuli which can cause a physiological response ScienceDaily (2021).

Stress is a particular type of affective state; it is interesting medically due to the influence it can have on mood, behaviour and health. The negative long-term effects of stress are significant and cause accelerated disease progression, severe impact on depression and mental health and are causes of gastrointestinal disorders to name just a few Leserman et al. (1999); Cohen et al. (2007); McEwen and Stellar (1993). The ability to effectively detect and monitor stressed states in individuals is thus important to mitigate such effects. This calls for automated stress detection methods that allow for real-time information to be sent back to a user experiencing a state of stress to help monitor and manage stress levels.

When experiencing a state of stress, the body releases 'stress hormones' which include cortisol and adrenaline. These hormones affect both the respiratory and cardiovascular systems, which tends to result in both increased breathing rate and increased heart rate, both in an effort to more quickly divert oxygen-rich blood to various areas of the body. This fact that such a 'fight or flight' response is induced by stress means that prediction of such a state can, in theory, be detected through analysis of the changes in a number of physiological signals.

In this work, we focus explicitly on stress prediction from breathing signals. Previous works on this topic Plarre et al. (2011); Jongyoon Choi et al. (2012); Schmidt et al. (2018) follow a standard sliding window prediction pipeline format. Here, a sliding window is run along the length of the data, and a host of window-aggregated features are computed and used as features to train a machine-learning classifier.

Our contributions in this area are threefold:

1. We propose a novel signature-based method for predicting stress from breathing data.
2. We propose three new interpretable features hitherto unconsidered in the literature that can be used by practitioners regardless of model choice. We explain the intuitions behind each feature and show that they show significant distributional differences in cases of stress vs non stress.
3. We verify experimentally:
   (a) Interpretable signature features alongside the proposed method adaptation results in performance **on par with existing methods**.

(b) Dramatic performance improvements can be achieved by considering further signature features, though at the cost of less interpretability.

The rest of this chapter is set out as follows: first, we overview the prior work in this space; second, we introduce the dataset and benchmarks we will be using in our analysis; third, we overview how to build the feature set and implement the methodology; fourth, we present the main results; fifth, we perform a study into the interpretability of the feature set and model; and finally, we give some directions of future work and conclusions.

8.2 Previous Work

The task of detecting stress from some set of physiological signals has been undertaken before; for example, Plarre et al. (2011); Jongyoon Choi et al. (2012); Schmidt et al. (2018) all describe studies in which participants were induced to experience states of stress and non-stress whilst having a collection of physiological signals continuously monitored. The most common signals include respiration (Resp), electrocardiograph (ECG), electromyograph (EDA), temperature (Temp), and accelerometer (ACC). Accuracies at detecting stress using multiple modalities typically reach around 90%, the aforementioned studies report 90%, 81% and 93% respectively.

In this work we choose to focus on the dataset introduced by (Schmidt et al., 2018), and explicitly on the respiration signal. This is because our proposed method is particularly well suited to this modality and is easy to demonstrate the signatures interpretability properties with. We also note that respiration has often been shown to achieve the top, if not close to, predictive performance against stress for a single modality, see for example Plarre et al. (2011); Schmidt et al. (2018).

8.3 The WESAD Study

In this paper we place particular focus on the methods used in the Wearable Stress and Affect Detection (WESAD) study, as the dataset used has been made freely available and thus is the one we will use throughout our analysis.

8.3.1 Study Overview

In this study, 15 healthy volunteers were equipped with sensors to measure five distinct signals. An abbreviation of each of the signals, along with what each measures is given below.

- **Resp**: Respiratory activity, measures the % expansion of a band located around the chest.
- **ECG**: Electrical activity of the heart (mV).
- **EDA**: Electrical conductivity of the skin (mV).

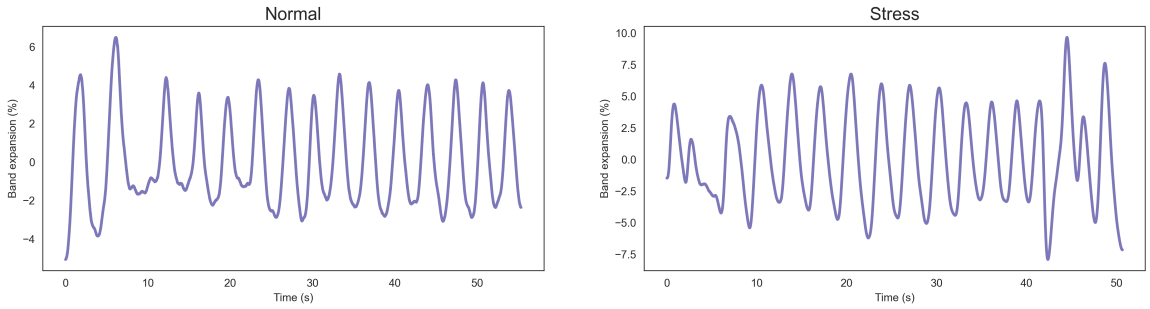


Figure 8.3.1 – An example of the breathing signal for a participant in the normal state (left), and a participant in a stressed state (right). The $y-$axis measures the percentage expansion of a band located around the wearers chest as they inhale and exhale.

- **EMG**: Electrical activity of the muscles (mV).
- **Temp**: Body temperature (°C)
- **ACC**: Motion in 3-dimensions, measuring the position of the person as they move around.

All signals were sampled at 700Hz. As already stated, in this paper we are specifically concerned with improving the extraction of information from the breathing signal, and as such, choose to reduce the data onto a 2D signal of Resp and time. Combination of information of the other modalities in sensible ways will likely lead to improved performance, however, we feel that it will only muddy the clarity of the analysis that would be done by examining the improvements on the breathing signal alone, and as such we ignore the other features from hereon.

The goal of the study was to elicit three states in the volunteers, and then attempt to discriminate between them using the data. The states under consideration were baseline, stressed and amused. The baseline state was captured by having the subjects in a relaxed state sitting/standing at a table reading; data was acquired for 20 minutes. During the amusement state, volunteers watched a set of eleven funny video clips for a total of 392 seconds. Finally for the stressed state the subjects were exposed to the Trier Social Stress Test which has been well studied and is thought to reliably induce a state of stress Kirschbaum et al. (1993). The test involves a public speaking task, and then to count down from 2023 to zero, in steps of 17. The total time in the stressed state was around 10 minutes for each person. The specifics as to how the procedure was conducted is given fully in Schmidt et al. (2018). Note that for this study we will merge the relaxed and amused states into a 'normal' or 'non-stressed' state so as to create a binary stress vs non-stress classification problem.

In Fig. 8.3.1 we plot two 60 second segments of the raw breathing signal, one for a person in a normal state (left), and one for a person experiencing a stressed state (right).

8.3.2 Study Benchmark

The authors of the original paper Schmidt et al. (2018) performed their own analysis to predict whether a person was experiencing a state of stress. The method they employed followed closely other examples found in the literature, for example Plarre et al. (2011); Jongyoon Choi et al. (2012). The approach is as follows:

1. Segment the signal using a sliding window of 60s with a window shift of 0.25s.
2. For each window, compute a selection of features over the window, resulting in a static feature vector for each window.
3. Train a variety of machine learning classifiers using a leave-one-person-out cross-validation procedure and evaluate the performance.

Considering the breathing signal only, the top performing classifier attained an accuracy of 88.1% and an F1-score of 85.6.

The features extracted over the window were:

- $\boldsymbol{\mu}_{\mathrm{I}}, \boldsymbol{\sigma}_{\mathrm{I}}$ - The mean and standard deviation of the inhalation duration.
- $\boldsymbol{\mu}_{\mathrm{E}}, \boldsymbol{\sigma}_{\mathrm{E}}$ - The mean and standard deviation of the exhalation duration.
- **I/E** - Total inhalation to exhalation duration ratio.
- **Stretch** - Maximum value of the resp signal minus the minimum value over the interval.
- $\mathbf{Vol}_{\mathrm{insp}}$ - Inspiration volume.
- $\sum_{\mathrm{Resp}}$ - Respiration duration.
- **RR** - Respiration rate

These features are typical for this kind of problem, as can be seen in Plarre et al. (2011); Jongyoon Choi et al. (2012). On the one hand, these features are well known and highly interpretable. On the other hand, they are not mathematically rigorous and likely to be highly correlated. Whats more, there is a significant loss of information between the raw signal and the features listed above, and whilst it is technically possible they contain all the relevant information for predicting stress, we will see later that this is not the case.

Another downside of the above feature set is that there is not obvious way in which to extend the set, so as to provide a more accurate depiction of any 60s window. We would need to search for other features, however, we have no guarantees that these would actually be adding anything or whether they were already contained in the features that have already been computed.

Our method seeks to address all of these issues, as will be discussed in the following Section 8.5.

8.4 Log-signature interpretability of a single breath

We begin by outlining the interpretability of the log-signature transformation in the context of a single breath. The log-signature is a more suitable representation of the information when it comes to interpretability, and as such, the following analysis is given in terms of the log-signature feature values.

8.4.1 Log-signature features

The log-signature of a 2-dimensional stream has (2, 1, 2) features for depths (1, 2, 3) respectively. Here, our signal features are time (T) and breathing (B), and as such, we label the features as follows:

Depth 1: **T** and **B**

Depth 2: [**T**, **B**]

Depth 3: [**T**, [**T**, **B**]] and [**T**, [**B**, **B**]]

The log-signature to depth 3 is then the collection of features {**T**, **B**, [**T**, **B**], [**T**, [**T**, **B**]], [**T**, [**B**, **B**]]}.

We write the features in this form as this aligns with how they are typically presented in the literature. The square brackets have a special mathematical meaning (they are known as Lie Brackets, see Reizenstein (2019)) however it is not important for what follows and will simply be enough to consider them as standalone labels.

We will now go on to describe the interpretation for each of the features in turn for depths 1, 2, and 3.

8.4.2 Feature interpretability

We now outline the interpretability of the features up to depth 3. Depth 2 interpretability has been shown already in Chapter 2, however the depth 3 analysis has hiterto not been seen.

Depth 1 signature terms simply describe the change in each feature over the interval. Here this is the respiration duration and the difference in band expansion from the beginning to the end of the breath which we term the *breath residual* (this will generally be close to 0).

$$\mathbf{T} = \Delta_{\text{time}} = \text{Respiration duration} \tag{8.1}$$

$$\mathbf{B} = \Delta_{\text{band\%}} \stackrel{\text{def}}{=} \text{Breath residual} \tag{8.2}$$

Depth 2 contains a single log-signature feature. It is well known that this is equivalent to the Lévy area of the path Kormilitzin et al. (2016); this is the signed area

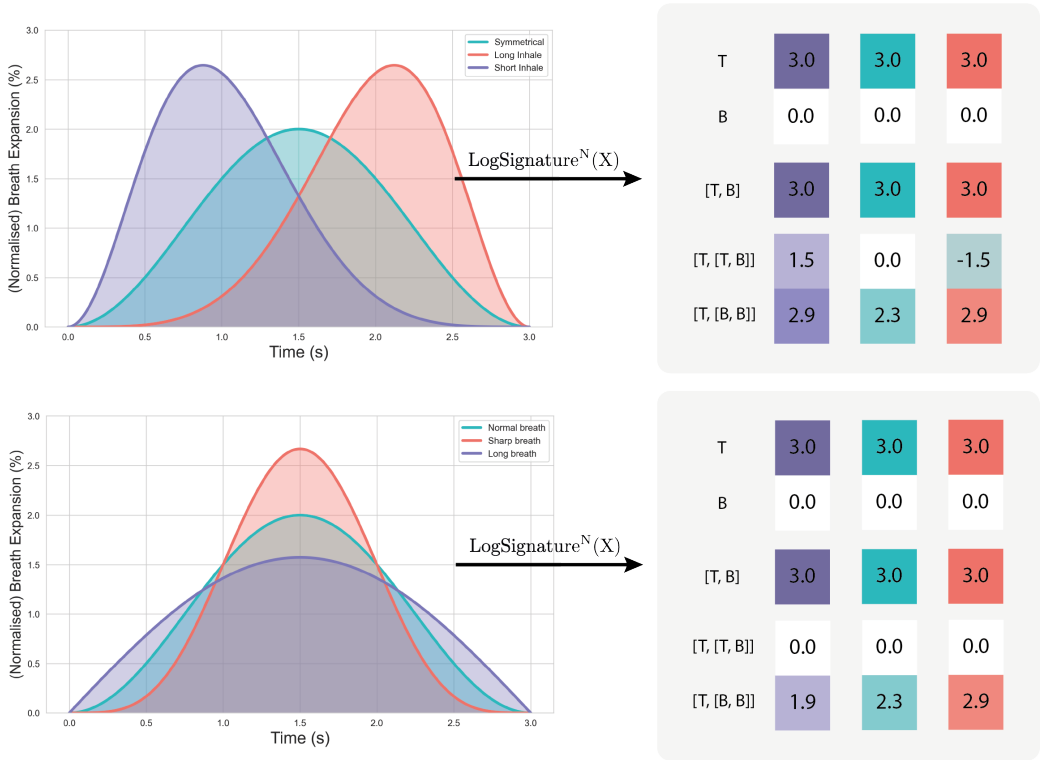


Figure 8.4.1 – Interpretation of the depth 3 log-signature features. **Top:** the **[T, [T, B]]** term varies with the amount of front or back *loading*. **Bottom:** the **[T, [B, B]]** term varies with the amount of vertical *pinch*.

between the path and the chord joining the endpoints (this was introduced in Section 2.4.3). Since this already has a name we will just refer to this as the breaths Lévy area.

$$[\mathbf{T}, \mathbf{B}] = \text{Lévy area} \tag{8.3}$$

Depth 3 results in two terms described by **[T, [T, B]]** and **[T, [B, B]]**. Depth 3 terms have seldom been visualised in the signature literature to date, however, we give a visualisation in the context of the breathing data in Figure 8.4.1.

We analyse these terms in turn by simulating three breaths that have equal values with each other at depths 1 and 2 (that is, the same deltas and Lévy area), but differ one of the depth 3 values.

For **[T, [T, B]]**, we can see from the top of Figure 8.4.1 that the term captures the extent to which the curve is front-loaded (purple) or back loaded (red) compared with the perfectly symmetrical curve (green). We can see the numerical change in the **[T, [T, B]]** feature by examining the 4^{th} row of the data on the right hand side that shows how the feature value shifts from positive, to zero, to negative depending on the weight of the loading. For this reason, we call this the *breath loading* feature.

From the perspective of breathing, a low-level description of this feature is that it captures the extent to which a given breath corresponds to a short inhale followed by

a long inhale, or vice-versa.

We now move to look at the **[T, [B, B]]** term in the bottom of Figure 8.4.1. We again present curves with the same areas, but now we vary the concentration of the areas along the vertical axis. We see that the higher the area displacement, the larger the **[T, [B, B]]** term. Given that this looks like we have pinched the curve in the vertical direction by increasing amounts, we call this feature the *breath pinch*.

Overall we have shown that the signature computation to depth 3 results in the following interpretable features

- **T** – Respiration duration
- **B** – Breath residual
- **[T, B]** – Lévy area
- **[T, [T, B]]** – Loading
- **[T, [B, B]]** – Pinch

8.4.3 Extension to higher depths is easy

As mentioned in Section 8.3.2, it is difficult to extend a set of hand crafted features, such as the ones used in the stress literature to data, in a rigorous manner. To extend such a set, new features must be introduced. However, once one reaches the terminus of existing expert knowledge, it is unclear how best to proceed.

This is not the case with signatures. Unlike hand-crafted feature sets we can extend the signature feature set in a mathematically rigorous manner to learn more accurate function approximations (Universal nonlinearity; Lyons (1998); Bonnier et al. (2019) (Theorem 2.4.3)).

This all comes with a large caveat since we do not have perfect learning algorithms or unlimited data. At some point performance will start to worsen as the feature space grows (either due to overfit or time restrictions). However, when using the signature we can easily extend our feature space to find this bottleneck with minimal additional effort.

8.4.4 Reconstruction of the Breath from the Signature Features

It is known that the signature truncated to infinite depth is (in principle) invertible and thus results in no information loss from the raw signal (Uniqueness of the Signature; Hambly and Lyons (2010) (Theorem 2.4.2)). However, in practice, the signature must be truncated to a finite depth resulting in a loss of information. Here we attempt to gauge how much information is lost by seeing how well the raw signal can be reconstructed from its signature features at different depths. The method we use to invert the signature is given in Appendix D.1.

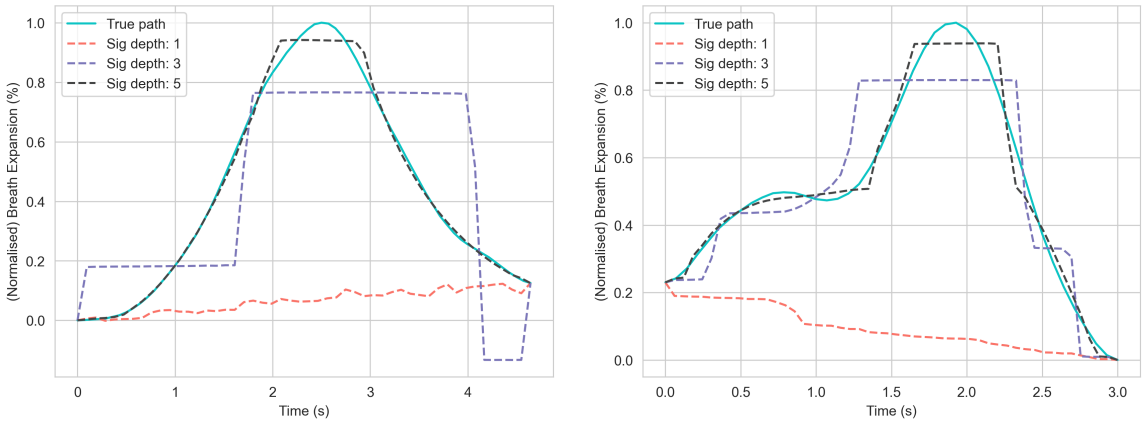


Figure 8.4.2 – Two example reconstructions of the respiratory signal from the signature features. At depth 1, little is understood except how to reconstruct the endpoint (which is to be expected), as depth increases an increased reconstruction quality is attained.

The reconstruction plots for depths 1, 3, 5 against the true signal are given in Figure 8.4.2. We see that the depth 1 reconstruction learns only how to reach the end point from the start point, significant improvements are made as we go up to depth 5, where the reconstruction becomes very close.

8.5 Method

In this section, we outline how we use signature features over different breaths to improve upon the quality of the information extracted from the breathing signal compared with that of the WESAD study as introduced in Section 8.3.

As outlined in Section 8.1, in previous work features were extracted from the breathing data and aggregated over an arbitrary 60s window before being used to train a machine-learning classifier.

Here we instead propose to train a model to predict the stressed state of an individual breath which we label the *weak predictor*. Then, we combine the outputs of these models over a window of breaths; this is the *strong predictor*.

We give an example of a window of breathing data split by individual breaths in Figure 8.5.1.

8.5.1 Creating a (weak) breath predictor

To create the weak predictor we utilise the signature transform as our mode of feature extraction. This is applied to the 2 dimensional signal of time (T) and respiration (R) to each individual breath segment. The result is a vector of features describing the breath (as in Section 8.4).

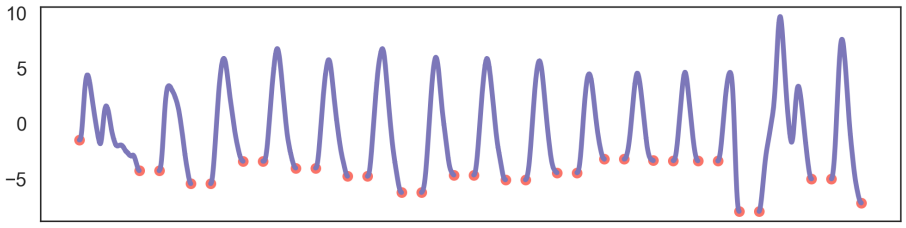


Figure 8.5.1 – The breathing signal after application of a peak finding algorithm designed to split the data into individual breath segments.

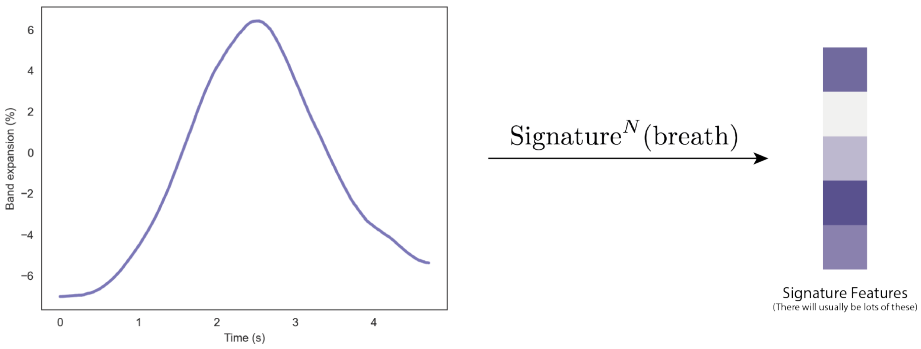


Figure 8.5.2 – Depth-N signature transform applied to a single breath returning a collection of signature features.

We give a pictorial representation of this feature extraction process in Fig. 8.5.2.

Now that each breath is represented by an equal length feature vector, we train a machine learning model to predict the state of stress from the signatures of a single breath. We found gradient boosted models to be particularly performant here, though linear models also work well.

8.5.2 Creating a (strong) window averaged predictor

The model from Section 8.5.1 outputs the probability that any individual breath is from a stressed individual. Since we are usually considering a window of breaths (often 60s to align with previous literature), the model defines a vector of probabilities corresponding the the probabilities that each breath was stressed.

To convert these probabilities over the window into a strong predictor, we perform a simple averaging operation. There are other things we could try, such as training a further model on the resultant probabilities, or utilising a voting classifier, however we found averaging perform no worse and with significantly less overhead.

8.5.3 Model overview

The above sections describe the model procedure in full. To overview we have,

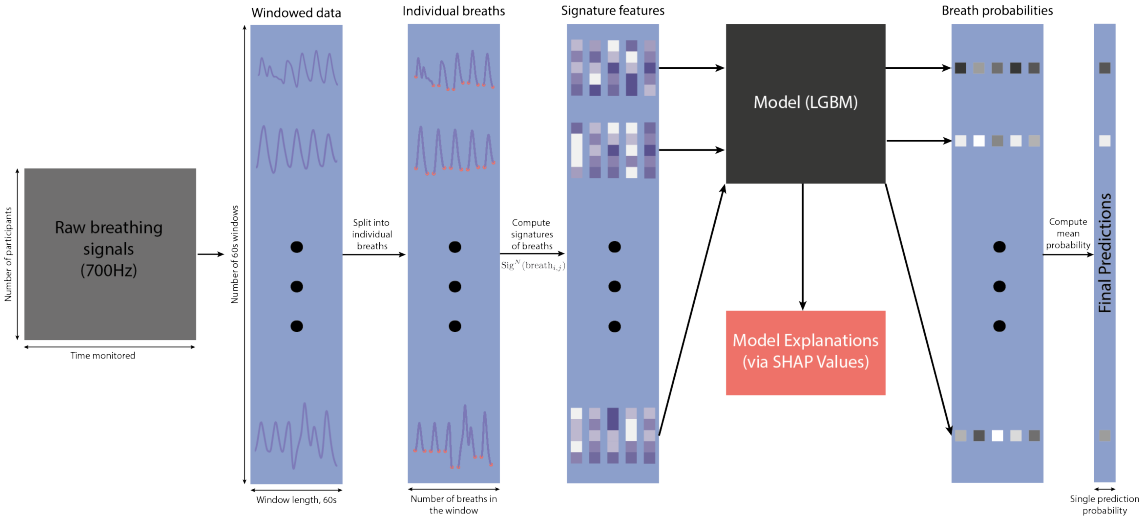


Figure 8.5.3 – An overview of the model training procedure. We start with the raw, high frequency breathing data. This data is then windowed and split into individual breath segments. The signature of each breath is then extracted before being fed through a machine learning classifier that determines the probability any given breath was from a stressed participant. These probabilities are then averaged to give an overall probability that the window of data came from a stressed participant. We have an additional arrow from the model that denotes that the model outputs can be analysed by any ML interpretation model (we use SHAP).

1. **Window the data** - Perform a standard windowing operation on the raw data (for example 60s to align with previous literature).
2. **Split out breaths** - Split the windowed data into individual breath components.
3. **Compute signatures** - Compute signatures to a chosen truncation depth (usually found via a hyperparameter search) for each breath in the data.
4. **Train a model** - Train a model to predict the stressed state of a given breath.
5. **Average the predictions** - Average the probabilities for each breath produce a single output probability for each distinct window.

A pictorial representation of this process is given in Fig. 8.5.3.

8.6 Results

In Table 8.6.1 we present the main performance comparison of the following methods:

1. **Original WESAD method** - The original method proposed in the WESAD paper where the features defined in Section 8.3.2 are extracted and aggregated along 60s windows before training a model on said features.

Method	Metrics		
	Accuracy	AUC	F1 Score
Original WESAD method	90.1	96.0	81.6
WESAD features, new method	91.6	96.4	83.9
Signatures (interpretable, depth 3)	91.8	95.8	84.6
Signatures (performance, depth 6)	**97.1**	**99.4**	**94.6**

Table 8.6.1 – Signatures vs benchmarks on the WESAD stress prediction task. The dashed line separates the original method (top) from the new methods. The WESAD (aggregated models) method uses the same features as in the original method, but applies the new method of aggregating single breath models. The results with signatures are separated into a low-depth method that retains interpretability, and a high depth method that sacrifices interpretability in place of a more powerful feature set.

2. **WESAD features, new method** - The proposed method of combining single breath models but using the features defined in the WESAD paper.
3. **Signatures (interpretable, depth 3)** - The proposed method using only the interpretable signatures given in Section 8.4.
4. **Signatures (performance, depth 6)** - The proposed method using a higher signature depth so as to maximise performance.

Firstly, we note that the WESAD benchmark is beaten (slightly) by using the proposed method than the original feature aggregation approach.

With regards to signatures, we see that the signature approach to depth 3 is sufficient to match the performance seen with the traditional features and note that here signatures remain interpretable. We also demonstrate that by increasing the signature depth to higher orders, performance can be significantly increased above traditional methods, seen most notably via a jump in the F1-score by 10 points.

In Table 8.6.2 and Fig. 8.6.1 we display the changes in performance as we increase the depth of the signature truncation level. The dashed lines on the figure represent the metric values from the WESAD benchmark. We find that by depth 3, the signature method improves upon the WESAD benchmark in all metrics, and continues to improve until depth 6.

We see that after signature depth 6 the performance of the model hits a plateau, and even decreases slightly. This is most likely because the number of features extracted by the signature is exponential in depth, the higher we increase depth, the more terms we attain. This is good up to a point because higher depth results in more information, however at some point too many features (a lot of which being uninformative) becomes detrimental to learning. For this reason depth should always be treated as a hyperparameter to be optimised.

	Metrics		
Depth	Accuracy (%)	AUC	F1
1	84.0	91.0	69.9
2	86.4	92.1	74.6
3	91.8	95.8	84.6
4	95.3	98.5	91.2
5	96.0	98.8	92.6
6	**97.1**	99.4	**94.6**
7	**97.1**	99.4	94.5
8	96.5	99.4	93.4
9	96.9	**99.5**	94.1
10	96.4	99.4	93.1

Table 8.6.2 – Performances of the signature model at different signature depths. Bold values indicate best performance for the given metric.

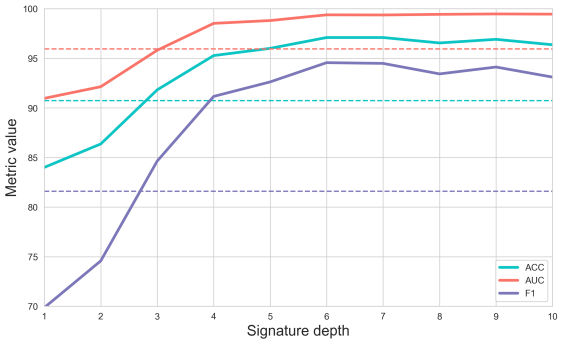


Figure 8.6.1 – Graphical representation of the information in the table on the left. Dashed lines represent the performance of the original WESAD model for each metric.

8.7 Model interpretation

In this section we perform a short analysis to interpret the decisions made by the model. We do this by performing an analysis of the most important features via a Shapley value analysis.

8.7.1 Shapley values

Shapley values, named due to their introduction by Llyod S Shapley (Shapley, 1953), are a result from cooperative game theory and describe a method for assigning payouts to individual players depending on their contribution to total the total payout of a game.

Whilst the field of cooperative game theory may appear at first quite disparate from that of machine learning interpretability, it turns out one can naturally think of 'the players' as the model features, and 'the game' as the prediction task at hand. The payout corresponds with the value of the model prediction, and the aim is to fairly distribute a payout to each of the features that explains their individual contributions.

Shapley values have strong theoretical groundings with axioms – efficiency, symmetry, dummy, and additivity – that give the method strong foundations. Full details of these axioms are given in the appendix, but note that non-satisfaction of these axioms is something that pervades other methods for model interpretability, such as LIME (Ribeiro et al., 2016). There have also been recent algorithmic developments have shown that they can be computed efficiently for tree-based models (Lundberg et al.,

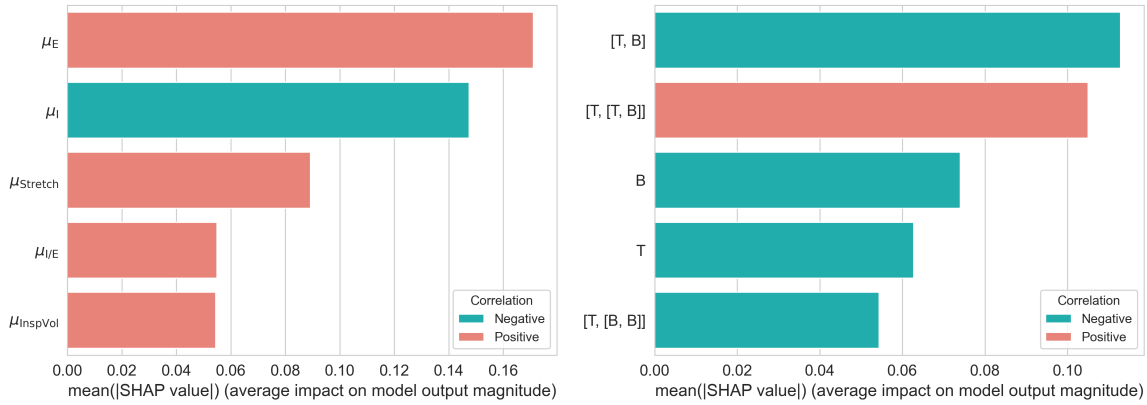


Figure 8.7.1 – Top 5 features as determined by the Shapley values for the WESAD features (left) and the signature features (right). An orange coloured bar indicates that the value is positively correlated with stress, and a green bar indicates the converse.

2018a), which are the top performing algorithms for our problem. For these reasons we choose to use Shapley values with which to form the basis of our model interpretability study.

8.7.2 Feature importance

We begin by examining the feature importance as returned by the mean of the absolute SHAP values for each feature. We plot the top 5 features by their average Shapley value in Fig. 8.7.1 for both the original WESAD method and the proposed signature approach. The features are ordered by importance with larger values corresponding to more important features. Additionally, an orange colour is used to denote that the feature has a positive correlation with stress with a green denoting negative.

For the WESAD case, we see that the mean inhalation and exhalation duration's are the most performant features, with almost twice as much impact on the target variable than the next most feature. Mean inhalation duration is negatively correlated with the output, and mean exhalation is positively correlated with the output suggesting that under stress, participants typically take sharper inhalations and longer exhalations.

This is echoed in the results from our signature features since the **[T, [T, B]]** feature sees a positive correlation with stress. Recall from Section 8.4.1 that we define **[T, [T, B]]** in terms of loading, and a positive value corresponds to a front loaded breath (i.e. a shorter inhale) suggesting we would see shorter inhalations and longer exhalations in a stressed participant.

8.7.3 Class conditional feature distributions

We plot normalised histograms for the depth 2 and 3 log-signature features in Fig. 8.7.2 conditional on the stressed state of the participant. We see that stress appears neg-

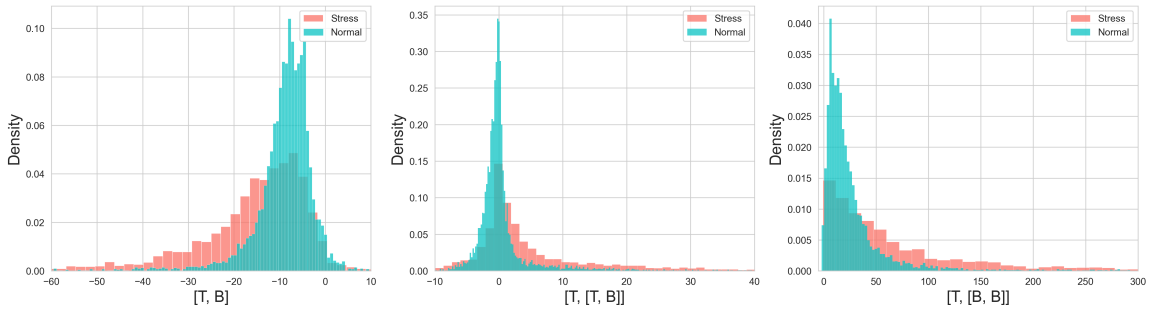


Figure 8.7.2 – Histogram that shows the distribution of values between normal and stressed breaths for the signature features of depth ≥ 3.

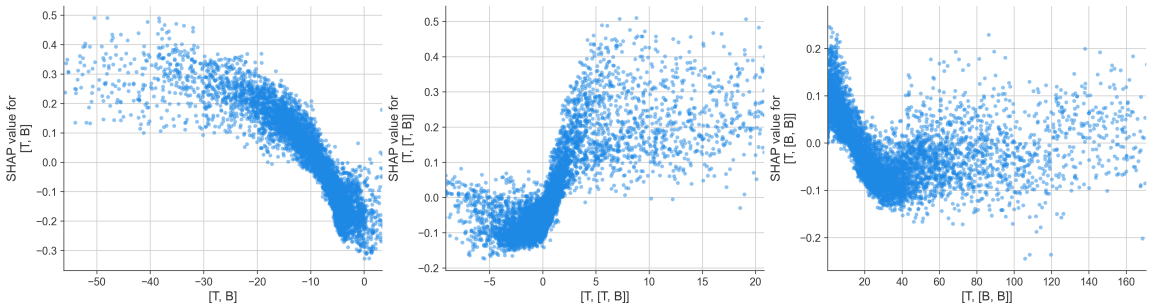


Figure 8.7.3 – SHAP partial dependence plot for the signature features of depth ≥ 3. Each dot represents a breath, the position along the x-axis gives the value of the corresponding feature, the y value gives the SHAP value at the feature value, and the points are coloured in accordance with the magnitude of the other feature.

atively correlated with **[T, B]** and positively correlated with **[T, [T, B]]**, which is inline with what we see in Fig. 8.7.1. However, the correlation with **[T, [T, B]]** appears positive, but is shown to be negative in the feature importance plot.

The **[T, B]** term represents the negative signed area of the signal, and as such, these observations show that in cases of stress, the total area swept out by the breathing signal is on average significantly higher than for cases of non stress.

Similarly reasoning with the depth 3 terms we have from the **[T, [T, B]]** term that the area under the inhalation curve is typically smaller than the area under the exhalation curve (short time for breath in, long for breath out) for stressed individuals. The positive correlation **[T, [B, B]]** with stress is suggestive of the fact that stressed individuals typically breath more 'pinched' breaths than non-stressed individuals.

8.7.4 Partial dependence plots

Finally, we examine the partial dependence of each feature with its Shapley value. We present this in Fig. 8.7.3 which gives the partial dependence plots for each feature.

For a given plot, each dot represents a single breath from the data with its x-

coordinate being the feature value, and y-coordinate its Shapley value. A positive Shapley value means that the feature value pushed the prediction towards a stress prediction from the state of stress, the larger the Shapley value, the larger the push. Similarly, a negative value means the feature pushed the prediction towards the non-stressed classification.

Take for example the leftmost plot that describes the **[T, B]** feature. We know from Fig. 8.7.2 that this exhibits a strong negative correlation with the state of stress, and this is also seen here. There is a strong negative correlation with the Shapley value in that a smaller value of **[T, B]** results in a larger SHAP value, which is what we expect.

The converse is seen for the **[T, [B, B]]** feature where Fig. 8.7.2 implies a positive correlation, but Fig. 8.7.3 shows a much less clear correlation. There is a nonlinear relationship between the value and its corresponding SHAP value. Large values often result in larger SHAP values, but smaller values also result in larger values.

To the best of our knowledge, these features, and the interpretablility thereof, have been hitherto unconsidered in the literature to date. As such, we make the suggestion to any practitioner looking to build a predictive model utilising the breathing signal, that she should examine the log-signature terms at least up to depth 3 (higher if interpretability is not a priority) and evaluate whether such a clear relationship exists and thus whether to incorporate them into any final modelling pipeline that is being constructed.

8.8 Conclusions

Here we have presented a new method for extracting information from breathing data using path signatures. The signature is used to extract features for each breath individually before a model is run and predictions are aggregated over windows of desired lengths. This approach yielded significant improvements in performance over existing baselines. Additionally to this, we examined the interpretability of the signature features and proposed three new features Lévy area, loading, and pinch, that we hope will be of use to practitioners analysing breathing signals in future.

Chapter 9

Conclusions

To the reader who has made it this far, we would first and foremost like to thank you very much for you time.

Three and a half years prior to the completion of this document, we set out with the goal to understand and develop some of the areas where applied rough path theory could be leveraged to make real world impact. The obvious route was through the fusion of rough path theory with machine learning, and ideas towards this end have formed the bulk of this document.

In Chapters 3 to 5 we introduced the neural controlled differential equation model - a continuous time model for irregular time series. We hope that this will find most impact in the modelling of 'messy' time series, problems that require continuous time inference, and through the modelling of long time series both via the ability to use memory efficient solvers and through the neural RDE approach. This chapters came with a host of applications that we hope have convinced the reader of their usefulness in each of these areas.

In Chapter 6 we overviewed and summarised the existing literature surrounding 'the signature method', and formalised and grouped a number of ideas that had been independently. This chapter aims to provide a springboard for an interested practitioner to develop performant models with signatures.

Chapters 7 and 8 were application focused chapters. These chapters are to serve as a demonstration of the usefulness of signatures, and more broadly ideas from rough paths, in real-world tasks involving streamed data.

To sum up, we hope that the work in this document has convinced the reader that machine learning and rough path theory are natural suitors, and that in combination they result in novel methodologies that are both interesting, and useful.

Bibliography

Arribas, I. P. (2018). Derivatives pricing using signature payoffs. *arXiv:1809.09466*.

Arribas, I. P., Goodwin, G. M., Geddes, J. R., Lyons, T., and Saunders, K. E. (2018). A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder. *Translational psychiatry*, 8:1–7.

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). The uea multivariate time series classification archive, 2018. *arXiv:1811.00075*.

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

Bagnall, A. J., Flynn, M., Large, J., Lines, J., and Middlehurst, M. (2020). A tale of two toolkits, report the third: on the usage and performance of hive-cote v1. 0. *CoRR*.

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

Bellot, A. and van der Schaar, M. (2021). Policy analysis using synthetic controls in continuous-time. *arXiv:2102.01577*.

Benavoli, A., Corani, G., and Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17:152–161.

Bonnier, P., Kidger, P., Perez Arribas, I., Salvi, C., and Lyons, T. (2019). Deep Signature Transforms. In *Advances in Neural Information Processing Systems*, pages 3099–3109.

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

Buchman, T. G., Simpson, S. Q., Sciarretta, K. L., Finne, K. P., Sowers, N., Collier, M., Chavan, S., Oke, I., Pennini, M. E., Santhosh, A., Wax, M., Woodbury, R., Chu, S., Merkeley, T. G., Disbrow, G. L., Bright, R. A., MaCurdy, T. E., and Kelman, J. A. (2020a). Sepsis Among Medicare Beneficiaries: 1. The Burdens of Sepsis, 2012–2018*. *Critical Care Medicine*, 48(3):276–288.

Buchman, T. G., Simpson, S. Q., Sciarretta, K. L., Finne, K. P., Sowers, N., Collier, M., Chavan, S., Oke, I., Pennini, M. E., Santhosh, A., Wax, M., Woodbury, R., Chu, S., Merkeley, T. G., Disbrow, G. L., Bright, R. A., MaCurdy, T. E., and Kelman, J. A. (2020b). Sepsis Among Medicare Beneficiaries: 2. The Trajectories of Sepsis, 2012–2018*. *Critical Care Medicine*, 48(3):289–301.

Buchman, T. G., Simpson, S. Q., Sciarretta, K. L., Finne, K. P., Sowers, N., Collier, M., Chavan, S., Oke, I., Pennini, M. E., Santhosh, A., Wax, M., Woodbury, R., Chu, S., Merkeley, T. G., Disbrow, G. L., Bright, R. A., MaCurdy, T. E., and Kelman, J. A. (2020c). Sepsis Among Medicare Beneficiaries: 3. The Methods, Models, and Forecasts of Sepsis, 2012–2018*. *Critical Care Medicine*, 48(3):302–318.

Calvert, J. S., Price, D. A., Chettipally, U. K., Barton, C. W., Feldman, M. D., Hoffman, J. L., Jay, M., and Das, R. (2016). A computational approach to early sepsis detection. *Computers in biology and medicine*, 74:69–73.

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

Chang, B., Chen, M., Haber, E., and Chi, E. H. (2019). AntisymmetricRNN: A dynamical system view on recurrent neural networks. *International Conference on Learning Representations*.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. (2018). Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

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

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12.

Chen, K.-T. (1954). Iterated integrals and exponential homomorphisms. *Proceedings of the London Mathematical Society*, 3(1):502–512.

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

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

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *EMNLP 2014*.

Choi, E., Bahadori, M. T., Kulas, J. A., Schuetz, A., Stewart, W. F., and Sun, J. (2016). Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. *arXiv:1608.05745*.

Chu, S. R. and Shoureshi, R. (1991). A neural network approach for identification of continuous-time nonlinear dynamic systems. In *1991 American Control Conference*, pages 1–5. IEEE.

Cohen, S., Janicki-Deverts, D., and Miller, G. E. (2007). Psychological Stress and Disease. *JAMA*, 298(14):1685.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. (2019). GRU-ODE-Bayes:

Continuous modeling of sporadically-observed time series. *Advances in Neural Information Processing Systems*.

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

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7:1–30.

Desautels, T., Calvert, J., Hoffman, J., Jay, M., Kerem, Y., Shieh, L., Shimabukuro, D., Chettipally, U., Feldman, M. D., Barton, C., Wales, D. J., , and Das, R. (2016). Prediction of sepsis in the intensive care unit with minimal electronic health record data: A machine learning approach. *JMIR medical informatics*, 4(3).

Facebook (2021). Ax · adaptive experimentation platform. https://ax.dev/.

Fagerström, J., Bång, M., Wilhelms, D., and Chew, M. S. (2019). LiSep LSTM: A machine learning algorithm for early detection of septic shock. *Scientific reports*, 9(1):1–8.

Fermanian, A. (2019). Embedding and learning with signatures. *arXiv:1911.13211*.

Flint, G., Hambly, B., and Lyons, T. (2016). Discretely sampled signals and the rough Hoff process. *Stochastic Processes and their Applications*, 126:2593–2614.

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

Friz, P. K. and Hairer, M. (2020). *A course on rough paths*. Springer.

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

Gholami, A., Keutzer, K., and Biros, G. (2019). Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv:1902.10298*.

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. (2017). The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30.

González-García, R., Rico-Martìnez, R., and Kevrekidis, I. G. (1998). Identification of distributed parameter systems: A neural net based approach. *Computers & chemical engineering*, 22:S965–S968.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2018). Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv:1810.01367*.

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

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772. PMLR.

Greff, K., Klein, A., Chovanec, M., Hutter, F., and Schmidhuber, J. (2017). The sacred infrastructure for computational research. In *Proceedings of the Python in Science Conferences-SciPy Conferences*.

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

Gyurkó, L. G., Lyons, T., Kontkowski, M., and Field, J. (2013). Extracting information from the signature of a financial data stream. *arXiv:1307.7244*.

Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse problems*, 34(1):014004.

Haber, E., Ruthotto, L., Holtham, E., and Jun, S.-H. (2018). Learning across scales—multiscale methods for convolution neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Henry, K. E., Hager, D. N., Pronovost, P. J., and Saria, S. (2015). A targeted real-time early warning score (TREWScore) for septic shock. *Science Translational Medicine*, 7(299):299ra122–299ra122.

Herrera, C., Krach, F., and Teichmann, J. (2021). Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering. In *International Conference on Learning Representations*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Jia, J. and Benson, A. R. (2019). Neural jump stochastic differential equations. *Advances in Neural Information Processing Systems*.

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

Johnson, A., Bulgarelli, L., Pollard, T., Horng, S., Celi, L. A., and Mark, R. (2021). The MIMIC-IV clinical database. `https://doi.org/10.13026/s6n6-xd98`.

Jongyoon Choi, Ahmed, B., and Gutierrez-Osuna, R. (2012). Development and Evaluation of an Ambulatory Stress Monitor Based on Wearable Sensors. *IEEE Transactions on Information Technology in Biomedicine*, 16(2):279–286.

Karim, F., Majumdar, S., Darabi, H., and Harford, S. (2019). Multivariate LSTM-FCNs for time series classification. *Neural Networks*, 116:237–245.

Karlsson, I., Papapetrou, P., and Boström, H. (2016). Generalized random shapelet forests. *Data mining and knowledge discovery*, 30(5):1053–1085.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *NeurIPS*, page 9.

Kidger, P. (2022). On neural differential equations. *arXiv:2202.02435*.

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

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

Kidger, P. and Lyons, T. (2021). Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*. `https://github.com/patrick-kidger/signatory`.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural Controlled Differential Equations for Irregular Time Series. *Advances in Neural Information Processing Systems*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*.

Kirschbaum, C., Pirke, K.-M., and Hellhammer, D. H. (1993). The 'Trier Social Stress Test' – A Tool for Investigating Psychobiological Stress Responses in a Laboratory Setting. *Neuropsychobiology*, 28(1-2):76–81.

Kormilitzin, A., Saunders, K., Harrison, P., Geddes, J., and Lyons, T. (2016). Application of the signature method to pattern recognition in the cequel clinical trial. *arXiv:1606.02074*.

Kormilitzin, A., Saunders, K. E., Harrison, P. J., Geddes, J. R., and Lyons, T. (2017). Detecting early signs of depressive and manic episodes in patients with bipolar disorder using the signature-based model. *arXiv:1708.01206*.

Kumar, A., Roberts, D., Wood, K. E., Light, B., Parrillo, J. E., Sharma, S., Suppes, R., Feinstein, D., Zanotti, S., Taiberg, L., Gurka, D., Kumar, A., and Cheang, M. (2006). Duration of hypotension before initiation of effective antimicrobial therapy is the critical determinant of survival in human septic shock:. *Critical Care Medicine*, 34(6):1589–1596.

Lai, S., Jin, L., and Yang, W. (2017). Online signature verification using recurrent neural network and length-normalized path signature descriptor. In *Proceedings of the 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 400–405. IEEE.

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

Leserman, J., Jackson, E. D., Petitto, J. M., Golden, R. N., Silva, S. G., Perkins, D. O., Cai, J., Folds, J. D., and Evans, D. L. (1999). Progression to AIDS: The Effects of Stress, Depressive Symptoms, and Social Support. *Psychosomatic Medicine*, 61(3):397–406.

Levin, D., Lyons, T., and Ni, H. (2013). Learning from the past, predicting the statistics for the future, learning an evolving system. *arXiv:1309.0260*.

Li, C., Zhang, X., and Jin, L. (2017). LPSNet: a novel log path signature feature based hand gesture recognition framework. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 631–639.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, pages 5243–5253.

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

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

Lundberg, S. M., Erion, G. G., and Lee, S.-I. (2018a). Consistent individualized feature attribution for tree ensembles. *arXiv:1802.03888*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777.

Lundberg, S. M., Nair, B., Vavilala, M. S., Horibe, M., Eisses, M. J., Adams, T., Liston, D. E., Low, D. K.-W., Newman, S.-F., Kim, J., et al. (2018b). Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering*, 2(10):749–760.

Lyons, T. (2014a). Rough paths, Signatures and the modelling of functions on streams. *arXiv:1405.4537 [math, q-fin, stat]*.

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

Lyons, T., Michael, C., and Thierry, L. (2007a). *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.

Lyons, T., Ni, H., and Oberhauser, H. (2014). A feature set for streams and an application to high-frequency financial tick data. In *Proceedings of the 2014 International Conference on Big Data Science and Computing*, page 5. ACM.

Lyons, T. and Oberhauser, H. (2017). Sketching the order of events. *arXiv:1708.09708*.

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

Lyons, T. J., Caruana, M., and Lévy, T. (2007b). *Differential equations driven by rough paths*. Springer.

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

McEwen, B. S. and Stellar, E. (1993). Stress and the individual: Mechanisms leading to disease. *Archives of internal medicine*, 153(18):2093–2101.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Morrill, J., Fermanian, A., Kidger, P., and Lyons, T. (2020a). A generalised signature method for time series. *arXiv:2006.00873*.

Morrill, J., Kidger, P., Yang, L., and Lyons, T. (2021a). Neural controlled differential equations for online prediction tasks. *arXiv:2106.11028*.

Morrill, J., Kormilitzin, A., Nevado-Holgado, A., Swaminathan, S., Howison, S., and Lyons, T. (2019). The signature-based model for early detection of sepsis from electronic health records in the intensive care unit. *International Conference in Computing in Cardiology*.

Morrill, J., Salvi, C., Kidger, P., Foster, J., and Lyons, T. (2021b). Neural rough differential equations for long time series. *International Conference on Machine Learning*.

Morrill, J. H., Kormilitzin, A., Nevado-Holgado, A. J., Swaminathan, S., Howison, S. D., and Lyons, T. J. (2020b). Utilization of the signature method to identify the early onset of sepsis from multivariate physiological time series in critical care monitoring. *Critical Care Medicine*, 48(10):e976–e981.

Nemati, S., Holder, A., Razmi, F., Stanley, M. D., Clifford, G. D., and Buchman, T. G. (2018). An interpretable machine learning model for accurate prediction of sepsis in the icu. *Critical Care Medicine.*, 46(4):547–53.

Onken, D. and Ruthotto, L. (2020). Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv:2005.13420*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. *GitHub*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

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

Plarre, K., Raij, A., Hossain, S. M., Ali, A. A., Nakajima, M., Al'Absi, M., Ertin, E., Kamarck, T., Kumar, S., Scott, M., et al. (2011). Continuous inference of psychological stress from sensory measurements collected in the natural environment. In *Proceedings of the 10th ACM/IEEE international conference on information processing in sensor networks*, pages 97–108. IEEE.

Pontryagin, L. S. (1987). *Mathematical theory of optimal processes*. CRC press.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. D. (2019). jl-a julia library for neural differential equations. *arXiv:1902.02376*.

Rackauckas, C. and Nie, Q. (2017). Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1).

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

Reizenstein, J. and Graham, B. (2018). The iisignature library: efficient calculation of iterated-integral signatures and log signatures. *arXiv:1802.08252*.

Reizenstein, J. F. (2019). *Iterated-integral signatures in machine learning*. PhD thesis, University of Warwick.

Reyes-Ortiz, J.-L., Oneto, L., Samà, A., Parra, X., and Anguita, D. (2016). Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767.

Reyna, M. A., Josef, C., Seyedi, S., Jeter, R., Shashikumar, S. P., Westover, M. B., Sharma, A., Nemati, S., and Clifford, G. D. (2019). Early prediction of sepsis from clinical data: the physionet/computing in cardiology challenge 2019. In *2019 Computing in Cardiology (CinC)*, pages Page–1. IEEE.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Model-agnostic interpretability of machine learning. *arXiv:1606.05386*.

Rico-Martinez, R., Anderson, J., and Kevrekidis, I. (1994). Continuous-time nonlinear signal processing: a neural network based approach for gray box identification. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 596–605. IEEE.

Rojas, E., Kahira, A. N., Meneses, E., Gomez, L. B., and Badia, R. M. (2020). A study of checkpointing in large scale training of deep neural networks. *arXiv:2012.00825*.

Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. (2019). Latent ordinary differential equations for irregularly-sampled time series. *Advances in Neural Information Processing Systems*.

Ruiz, A. P., Flynn, M., and Bagnall, A. (2020). Benchmarking multivariate time series classification algorithms. *arXiv:2007.13156*.

Schäfer, P. and Leser, U. (2017). Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 637–646.

Schmidt, P., Reiss, A., Duerichen, R., Marberger, C., and Van Laerhoven, K. (2018). Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection. In *Proceedings of the 2018 on International Conference on Multimodal Interaction - ICMI '18*, pages 400–408, Boulder, CO, USA. ACM Press.

ScienceDaily (2021). Stress medicine. https://www.sciencedaily.com/terms/stress_(medicine).htm.

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317.

Singer, M., Deutschman, C. S., Seymour, C. W., Shankar-Hari, M., Annane, D., Bauer, M., Bellomo, R., Bernard, G. R., Chiche, J.-D., Coopersmith, C. M., Hotchkiss, R. S., Levy, M. M., Marshall, J. C., Martin, G. S., Opal, S. M., Rubenfeld, G. D., van der Poll, T., Vincent, J.-L., and Angus, D. C. (2016). The Third International Consensus Definitions for Sepsis and Septic Shock (Sepsis-3). *JAMA*, 315(8):801.

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

Štrumbelj, E. and Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *ICML*.

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

Tange, O. (2011). Gnu parallel - the command-line power tool. *The USENIX Magazine*, 36:42–47.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv:1706.03762*.

Vincent, J. L., Moreno, R., Takala, J., Willatts, S., De Mendonça, A., Bruining, H., Reinhart, C. K., Suter, P. M., and Thijs, L. G. (1996). The sofa (sepsis-related organ failure assessment) score to describe organ dysfunction/failure. on behalf of the working group on sepsis-related problems of the european society of intensive care medicine. *Intensive Care Medicine*, 22:707–710.

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

Wang, B., Liakata, M., Ni, H., Lyons, T., Nevado-Holgado, A. J., and Saunders, K. (2019). A path signature approach for speech emotion recognition. In *Interspeech 2019*, pages 1661–1665. ISCA.

Wang, B., Wu, Y., Taylor, N., Lyons, T., Liakata, M., Nevado-Holgado, A. J., and Saunders, K. E. (2020a). Learning to detect bipolar disorder and borderline personality disorder with language and speech in non-clinical interviews. *arXiv:2008.03408*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020b). Linformer: Self-attention with linear complexity. *arXiv:2006.04768*.

Wang, Z., Yan, W., and Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE.

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv:1804.03209*.

Weinan, E. (2017). A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11.

Williams, D. (1991). *Probability with Martingales*. Cambridge University Press.

Wilson-Nunn, D., Lyons, T., Papavasiliou, A., and Ni, H. (2018). A path signature approach to online arabic handwriting recognition. In *2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*, pages 135–139. IEEE.

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

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

Wu, Y., Ni, H., Lyons, T., and Hudson, R. (2020). Signature features with the visibility transformation. *arXiv:2004.04006*.

Yang, W., Jin, L., and Liu, M. (2016a). Deepwriterid: An end-to-end online text-independent writer identification system. *IEEE Intelligent Systems*, 31:45–53.

Yang, W., Jin, L., Tao, D., Xie, Z., and Feng, Z. (2016b). Dropsample: A new training method to enhance deep convolutional neural networks for large-scale unconstrained handwritten chinese character recognition. *Pattern Recognition*, 58:190–203.

Yang, W., Lyons, T., Ni, H., Schmid, C., Jin, L., and Chang, J. (2017). Developing the path signature methodology and its application to landmark-based human action recognition. *arXiv:1707.03993*.

Appendices

Appendix A

Understanding Control Signals for Neural CDEs

A.1 Theoretical conditions on the control

A.1.1 Boundedness

This section extends ideas introduced in Eq. (4.2).

A.1.1.1 Boundedness is required for universal approximation

Let $\mathcal{X}$ be some set of time series such that

$$\sup_{\mathbf{x}\in\mathcal{X}} \omega(\max_i \tau_i, \min_i \tau_i, \max_i |x_i|) < \infty.$$

Let $\mathfrak{X} = \{X_{\mathbf{x}}\}\, \mathbf{x} \in \mathcal{X}$. By equation (4.2), then

$$\sup_{X\in\mathfrak{X}} \|X\|_\infty + \left\|\frac{\mathrm{d}X}{\mathrm{d}t}\right\|_\infty + \left|\frac{\mathrm{d}X}{\mathrm{d}t}\right|_{BV} < \infty.$$

Let $\mathfrak{X}' = \left\{{}^{\mathrm{d}X}/_{\mathrm{d}t}\right\} X \in \mathfrak{X}$. Then $\mathfrak{X}$ is bounded in $W^{1,\infty}[t_0, t_n]$ and so relatively compact in $L^\infty[t_0, t_n]$, and $\mathfrak{X}'$ is bounded in $BV[t_0, t_n]$ and so relatively compact in $L^1[t_0, t_n]$. Therefore $\mathfrak{X} \times \mathfrak{X}'$ is relatively compact in $L^\infty[t_0, t_n] \times L^1[t_0, t_n]$.

Let $\mathbb{X} = \left\{(X, {}^{\mathrm{d}X}/_{\mathrm{d}t})\right\} X \in \mathfrak{X}$. Then $\mathbb{X} \subseteq \mathfrak{X} \times \mathfrak{X}'$ so $\mathbb{X}$ is also relatively compact in $L^\infty[t_0, t_n] \times L^1[t_0, t_n]$. This implies that $\mathfrak{X}$ is relatively compact with respect to the topology generated by $\|X\|_\infty + \left\|{}^{\mathrm{d}X}/_{\mathrm{d}t}\right\|_1$, and hence also with respect to the topology generated by $\|X\|_\infty + |X|_{BV}$.

This now satisfies the compactness condition of Kidger et al. (2020, Theorem B.7), implying that Neural CDEs driven by $X \in \mathfrak{X}$ are universal approximators on $\mathbf{x} \in \mathcal{X}$.

A.1.1.2 Boundedness of cubic Hermite splines with backward differences

The boundedness result for natural cubic splines is given in Kidger et al. (2020, Lemma B.9). We now give a corresponding proof for cubic Hermite splines with backward differences.

Let $X_{\mathbf{x}}\colon [t_0, t_n] \mapsto \mathbb{R}$ be the cubic Hermite splines with backward differences such that $X_{\mathbf{x}}(t_i) = x_i$. Note that it is sufficient to consider $X_{\mathbf{x}} \in \mathbb{R}$ here since each dimension is interpolated separately. Let the i^{th} piece of $X_{\mathbf{x}}$, on the interval $[t_i, t_{i+1}]$ be denoted by X_i. Without loss of generality, translate each piece onto the interval $[0, \tau_i]$ where $\tau_i = t_{i+1} - t_i$, so that $X_i\colon [0, \tau_i] \mapsto \mathbb{R}$. Let $X_i(t) = a_i + b_i t + c_i t^2 + d_i t^3$ for some coefficients a_i, b_i, c_i, d_i and $i \in \{0, \ldots, n-1\}$.

For cubic Hermite splines with backward differences we enforce $X_i(0) = x_i$, $X_i(\tau_i) = x_{i+1}$, along with the derivative conditions

$$X_i'(0) = \frac{x_i - x_{i-1}}{\tau_{i-1}}, \tag{A.1}$$

$$X_{i+1}'(\tau_i) = \frac{x_{i+1} - x_i}{\tau_i}. \tag{A.2}$$

Letting $\Delta x_i = x_i - x_{i-1}$, we find

$$a_i = x_i, \tag{A.3}$$

$$b_i = \tau_{i-1}^{-1} \Delta x_i, \tag{A.4}$$

$$c_i = 2\tau_i^{-2}\tau_{i-1}^{-1}(\tau_{i-1}\Delta x_{i+1} - \tau_i \Delta x_i), \tag{A.5}$$

$$d_i = \tau_i^{-3}\tau_{i-1}^{-1}(\tau_i \Delta x_i - \tau_{i-1}\Delta x_{i+1}). \tag{A.6}$$

Letting $x_{\max} = \max_i |x_i|$, $\tau_{\max} = \max_i \tau_i$, $\tau_{\min} = \min_i \tau_i$, it can be shown that

$$\|X_{\mathbf{x}}\|_\infty \leq C_1\big(x_{\max} + \frac{x_{\max}\tau_{\max}}{\tau_{\min}}\big), \tag{A.7}$$

$$\left\|\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t}\right\|_\infty \leq C_2 \frac{x_{\max}}{\tau_{\min}}, \tag{A.8}$$

$$\left|\frac{\mathrm{d}X_{\mathbf{x}}}{\mathrm{d}t}\right|_{BV} \leq C_3 \frac{x_{\max}}{\tau_{\min}^2}, \tag{A.9}$$

$$\tag{A.10}$$

for fixed constants $C_1, C_2, C_3 \in \mathbb{R}$. This proves the boundedness property from Equation (4.2) for cubic Hermite splines with backward differences.

A.1.2 Uniqueness

A.1.2.1 Interpolation schemes are non-unique in general

Here we give a simple counterexample to prove non-uniqueness of the interpolation schemes if only (t_i, x_i), and not c_i, are included as information.

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two time series such that

$$\mathbf{x}_1 = \big((t_0, x_0), (t_1, x_0)\big), \tag{A.11}$$
$$\mathbf{x}_2 = \big((t_0, x_0), (t_*, x_0), (t_1, x_0)\big), \tag{A.12}$$

where $t_0 < t_* < t_1$. Clearly, as one has an additional observations, the two time series contain different information. However, all interpolation schemes given in Section 4.3 result in the interpolation $X_{\mathbf{x}}(t) = (t, x_0)$ for $t \in [t_0, t_1]$. As such, the map $\mathbf{x} \to X_{\mathbf{x}}$ is not injective.

A.1.2.2 Observational frequencies guarantee uniqueness

Suppose that we have two time series, $\mathbf{x} = \{(t_0, x_0, c_0^x), \dots (t_n, x_n, c_n^x)\}$ and $\mathbf{y} = \{(t_0, y_0, c_0^y), \dots (t_n, y_n, c_n^y)\}$, where c_i^* denotes the corresponding measurement intensity vector. Assuming that $\mathbf{x} \neq \mathbf{y}$, then we must have that either

1. There exists some t_i for which $x_i \neq y_i$.
2. There exists at least one t_i for which $\mathbf{x}$ is measured and $\mathbf{y}$ is not (we take $\mathbf{x}$ over $\mathbf{y}$ here wlog).

This proof proceeds in two parts. Firstly, we show injectivity of the map $\mathbf{x} \mapsto X_{\mathbf{x}}$ by showing the interpolations from two different sets of data must be different. Secondly, we prove uniqueness of the map $X_{\mathbf{x}} \mapsto \text{Signature}(X_{\mathbf{x}})$ by showing that the schemes from Section 4.3 cannot contain tree-like pieces (see Hambly and Lyons (2010)).

Let t_i denote the first time for which one of the two conditions above is violated. If 1 holds, then $x_i \neq y_i$ and so $X_{\mathbf{x}}(t_i) \neq X_{\mathbf{y}}(t_i)$ there. Similarly, if 2 holds, then we will have that $c_i^x \neq c_i^y$ there and so again $X_{\mathbf{x}}(t_i) \neq X_{\mathbf{y}}(t_i)$. This then proves that

$$\mathbf{x} \mapsto X_{\mathbf{x}} \text{ is injective with respect to } \mathcal{X}, \tag{A.13}$$

provided we include observational intensities. Here $\mathcal{X}$ is as defined in Section 4.2.

To show additionally that we have injectivity of the signature, we need to argue that the path cannot contain tree-like pieces. For a path to contain a tree-like piece, the path must exactly trace itself backwards which is a strong condition (Hambly and Lyons, 2010). This can only be true if there is a point of turnaround for all dimensions simultaneously, we will now argue why this cannot be the case for any of our proposed control paths if the observational intensities c_i are included.

Recall that $c_i \in \mathbb{R}^v$ increments by one according to the channels of x_i that are measured at t_i. Let $c_{i,j} \in \mathbb{R}$ denote the j^{th} channel of c_i. By definition, at every time t_i at least one feature of c_i is updated. Therefore we can find $j_1, \dots, j_n$ with each $j_k \in \{1, \dots, v\}$ such that c_{i,j_i} is incremented.

For linear and cubic Hermite controls, there is always a dimension in c_i that is increasing. This is a sufficient condition for there to be no tree-like pieces. Similarly, for rectilinear, always at least one of the time dimension or a c_i dimension is increasing,

and so rectilinear controls too have no tree-like pieces. This condition is unknown however for natural cubic splines.

Together, this shows that provided we include c_i then all of our considered control schemes result in

$$\mathbf{x} \to (x_0, \text{Signature}(X_{\mathbf{x}})) \text{ is injective with respect to } \mathcal{X}, \tag{A.14}$$

holding for linear, cubic Hermite, and rectilinear controls.

Note that inclusion of c_i is by no means the only way to guarantee this property, however, it represents a sensible approach from which uniqueness can be achieved.

A.2 Experimental details

A.2.1 General notes

We being with details common to all experiments.

Code All code is available at `https://github.com/jambo6/online-neural-cdes`.

Normalisation For all datasets, each channel was normalised to have zero mean and unit variance.

Data splits For all problems we split the dataset into 3 (stratified by label for classification problems) with 75%/15%/15% in training/validation/testing respectively.

ODE Solvers When comparing between interpolation schemes the integral in Eq. (3.10) was solved using the Dormand-Prince 5(4) ("`dopri5`") scheme with an absolute tolerance of 10^{-5} and a relative tolerance of 10^{-3}. For the MIMIC-IV benchmarking results (Table 4.4.7) we used the Runge-Kutta-4 ("`rk4`") scheme.

Optimiser The optimiser used for every problem was Adam (Kingma and Ba (2015)).

Training details All models were run with batch size 1024 for up to 1000 epochs. If training loss stagnated for 15 epochs the learning rate was decreased by a factor of 10. If training loss stagnated for 60 epochs then model training was terminated and the model was rolled back to the point of lowest validation loss.

Computing infrastructure All experiments were run on two computers. One was equipped with two Quadro GP100's, the other with one NVIDIA A100-PCIE-40GB.

A.2.2 Hyperparameter selection

Hyperparameters for all datasets and models were found using a Bayesian optimisation with 20 trials using the Adaptive Experimentation Platform (Facebook, 2021) framework.

For Neural CDEs, hyperparameters were optimised separately for the the natural cubic, linear, and rectilinear schemes, with non-natural cubic inheriting the hyperparameters from linear.

We admit a slight error in missing off the learning rate optimisation for the Neural CDE models (it was fixed at 0.005), however if anything, this biases against the Neural CDE results.

The search space for all regular datasets was `hidden_dim in range [32, 256]`, `hidden_hidden_dim in range [32, 192]`, `num_layers in range [1, 4]`, and `learning_rate = 0.005`.

The search space was modified slightly for the MIMIC-IV tasks so as to work within the memory requirements of the GPUs. We used `hidden_dim in range [32, 128]`, `hidden_hidden_dim in range [32, 128]`, `num_layers in range [1, 4]`, and `learning_rate = 0.005`. The learning rate was dropped by a factor of 10 for training the actual to account for observed overfit.

For the GRU variants (GRU, GRU-dt, GRU-dt-intensity, GRU-D) we used `hidden_dim in range [32, 512]`, `learning_rate in range [0.0001, 0.1]`.

For the ODE-RNN we used `hidden_dim in range [32, 256]`, `hidden_hidden_dim in range [32, 196]`, `num_layers in range [1, 4]`, `learning_rate in range [0.0001, 0.01]`.

Final values for all hyperparameters are given in Tables A.2.1 and A.2.2

A.2.3 MIMIC-IV

The Medical Information Mart for Intensive Care (MIMIC) IV dataset comprises of de-identified patient-level electronic health record (EHR) data from over 50,000 patients that stayed in ICUs at the Beth Israel Deaconess Medical Center, Boston, Massachusetts between 2008 and 2019. MIMIC-IV builds on its predecessor MIMIC-III. Improvements include approximate year of admission has been made available (previously obfuscated as pat of de-identification), and there are more granular information on intake of medications through use of new tables.

Specifically, MIMIC-IV contains information on 53,150 patients, across 69,211 hospital admissions and 76,540 ICU stays. These are stored in relational tables in SQL format. Our first stage of data processing involves merging the information and pivoting the data such that we have time series data with each channel/dimension a separate variable of interest. We extracted data on vital signs, laboratory reading,

Dataset	Model	Interpolation	Hidden	Hidden hidden	Num layers	LR
BeijingPM10	NCDE	Natural cubic	256	168	1	0.0005
	NCDE	Linear	252	120	1	0.0005
	NCDE	Rectilinear	249	170	1	0.0005
BeijingPM2pt5	NCDE	Natural cubic	180	32	3	0.0005
	NCDE	Linear	256	196	1	0.0005
	NCDE	Rectilinear	256	171	1	0.0005
CharTraj	NCDE	Natural cubic	238	194	1	0.0005
	NCDE	Linear	249	170	1	0.0005
	NCDE	Rectilinear	148	89	2	0.0005
LOS	NCDE	Natural cubic	108	40	4	0.0005
	NCDE	Linear	65	119	2	0.0005
	NCDE	Rectilinear	32	126	4	0.0005
	ODE-RNN	–	32	168	1	0.002268
Mortality	NCDE	Natural cubic	41	55	4	0.0005
	NCDE	Linear	33	91	2	0.0005
	NCDE	Rectilinear	70	55	2	0.0005
	ODE-RNN	–	32	123	4	0.1
Sepsis	NCDE	Natural cubic	82	65	2	0.0005
	NCDE	Linear	82	65	2	0.0005
	NCDE	Rectilinear	82	65	2	0.0005
	ODE-RNN	–	215	98	2	0.001083
SpeechCommands	NCDE	Natural cubic	56	125	3	0.0005
	NCDE	Linear	77	118	3	0.0005
	NCDE	Rectilinear	113	124	3	0.0005

Table A.2.1 – Final hyperparameter values for the ODE models (Neural CDE and ODE-RNN).

Dataset	Model	Hidden dim	Learning rate
LOS	GRU	435	0.017858
LOS	GRU-dt	299	0.001359
LOS	GRU-dt-intensity	294	0.000753
LOS	GRU-D	408	0.000143
Mortality	GRU	432	0.025304
Mortality	GRU-dt	508	0.000158
Mortality	GRU-dt-intensity	196	0.017743
Mortality	GRU-D	398	0.000161
Sepsis	GRU	362	0.000697
Sepsis	GRU-dt	362	0.000697
Sepsis	GRU-dt-intensity	362	0.000697
Sepsis	GRU-D	362	0.000697

Table A.2.2 – Final hyperparameter values for the GRU variants.

ventilation devices. The full list of extracted features can be seen at [redacted for anonymity].

We use the MIMIC-IV dataset on three problems of interest, namely mortality prediction, length of stay (LOS) prediction and sepsis prediction.

For all tasks we removed patients whose stay length was greater than 72 hours (as the data has a very long tail in this regard), and additionally required a stay length greater than 4 hours and measurements at at least 4 different times. As the problems are different, the exclusion criteria are slightly different in each case. The exclusion concept diagram can be seen in Figure A.2.1

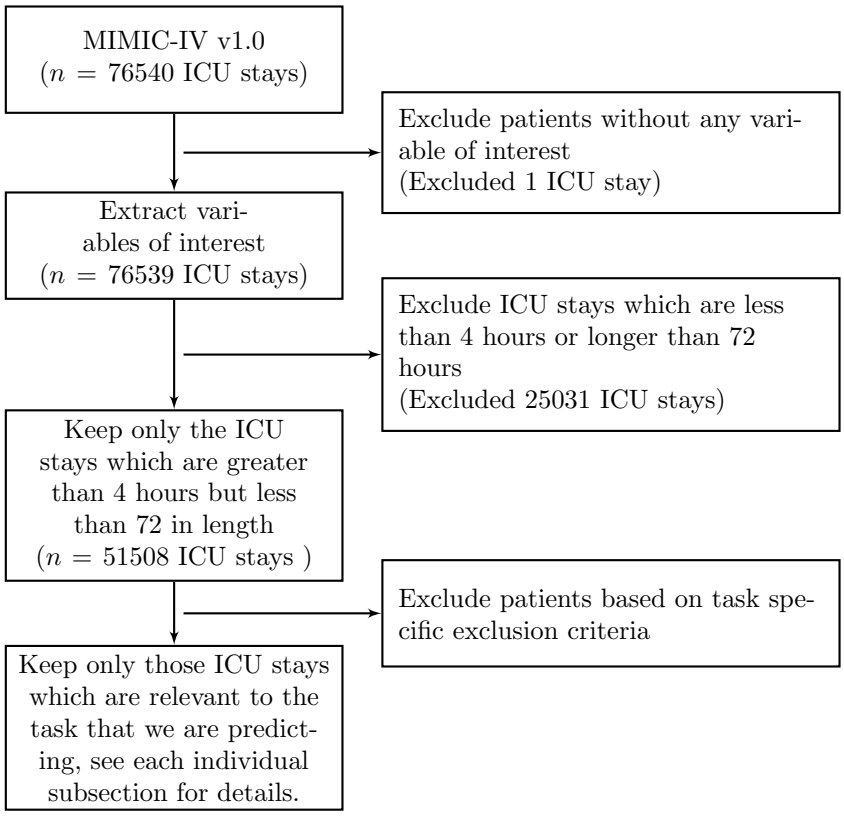


Figure A.2.1 – Study flow diagram.

Mortality Here we attempted to predict in hospital mortality for each patient. We required measurements at at least 4 unique times, and at least 4 hours worth of data. This resulted in 50,538 ICU stays.

LOS We predicted length of stay given the first 24 hours of data for patients who had a length of stay between 24 and 72 hours. This resulted in 16,054 ICU stays.

Sepsis Historically, there have been many standards used to define sepsis from the data as a way to operationalize research. One of the latest and commonly used defini-

tion is Sepsis-3 (Singer et al., 2016) which describes sepsis as "life-threatening organ dysfunction caused by a dysregulated host response to infection." For the purpose of operationalization, organ dysfunction is identified by an increase in Sequential Organ Failure Assessment (SOFA) score (Vincent et al., 1996) by 2 points or more consequent to the infection.

As the focus in (Singer et al., 2016) was not the detection of onset, the onset time was not made explicit. As a result, there have been various interpretations, for example at the point of SOFA score increase, or the earlier of time of suspected infection and time SOFA score increase in (Desautels et al., 2016) and (Nemati et al., 2018) respectively. From a clinical perspective, it makes sense to define the onset of sepsis as the time when we see organ dysfunction (as indicated by SOFA score) provided that we see evidence of suspicions of infection around the same time (thorough the use of antibiotics and cultures). Therefore this is the way we have chosen to define the time of onset of sepsis, in line with Desautels et al. (2016).

To determine the time of sepsis onset, we looked for a suspicion of infection and a deterioration in the SOFA score. To identify a suspicion of infection, we first find the times that lab cultures were taken. We also find the start of new antibiotic treatments, and require that there are two doses of antibiotics given to the patient within a 96 hour window, otherwise that time is not used in suspicion of infection. If an antibiotic treatment has commenced within 24 hours before or 72 hours after taking a culture sample, then the earlier of the two times defines a time of suspected infection. We then look at a patient's SOFA scores and see if there is an increase within 24 hours before or 48 hours after the time of suspected infection. We note that the choice of these time windows are aligned with Singer et al. (2016) but other choices are used in the literature. The time of sepsis onset is then given by the earlier time, i.e. min(time of suspected infection, time of SOFA deterioration).

The older, more obsolete EHR data from the CareVue system (2003-2008) has been removed in MIMIC-IV. This has meant that more patients can be included in the sepsis analysis as previously the CareVue data lacked the necessary antibiotic information required for the Sepsis-3 definition.

As well as using a cut off at 72 hours, we also removed patients who were prescribed antibiotics before their ICU admission time and those developed sepsis within 4 hours of their ICU stay. This resulted in 45,218 ICU stays.

A.2.4 Additional results

We give the interpolation results for the sepsis task in Appendix A.2.3. This is left out of the main paper only for reasons of space, but we see that it tells much the same story as the Mortality and LOS datasets. One note is that the rectilinear interpolation batch size had to be reduced from 1024 to 512 as it would not fit within GPU memory.

Control	**AUC**	**NFEs**$\times 10^3$
Natural cubic	0.779 ± 0.003	**16.2 ± 0.1**
Linear	**0.784 ± 0.002**	27.8 ± 0.3
Cubic Hermite	0.782 ± 0.002	20.2 ± 0.4
Rectilinear	0.772 ± 0.004	136.7 ± 4.5

Table A.2.3 – Comparison of the control signals on the sepsis dataset. This complements Tables 4.4.5 and 4.4.6 and is included in the appendix for completeness since it does not alter the narrative of the paper.

Appendix B

Neural Rough Differential Equations

B.1 An introduction to the log-ODE method for controlled differential equations

This section is technical and left in to show the reader how one actually derives an expression for the log-signature of a path, as well as how one derives the log-ODE method. Note that it is not as simple as simply taking the log! None of this is necessary for using the neural RDE method, but it can be useful to have a high level overview for how these things are derived.

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

$$\begin{aligned} \mathrm{d}Y_t &= f(Y_t)\,\mathrm{d}X_t, \\ Y_0 &= \xi, \end{aligned} \tag{B.1}$$

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 (Eq. (B.1)) 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 (Eq. (B.1)) and usually assumed to have $\mathrm{Lip}(\boldsymbol{\gamma})$ regularity (see definition 10.2 in Friz and Victoir (2010)). 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 B.1.1. *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*

$$\boldsymbol{a} + \boldsymbol{b} = \big(a_0 + b_0, a_1 + b_1, \cdots\big), \tag{B.2}$$

$$\boldsymbol{a} \otimes \boldsymbol{b} = \big(c_0, c_1, c_2, \cdots\big), \tag{B.3}$$

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

$$c_n := \sum_{k=0}^{n} a_k \otimes b_{n-k}. \tag{B.4}$$

The use of $\otimes$ *in equation (Eq.* (B.4)*) denotes the usual tensor product. The use of* $\otimes$ *in equation (Eq.* (B.3)*) is also referred to as the "tensor product": when precisely one* a_i *and precisely one* b_i *are nonzero then it reduces to the usual tensor product; equation (Eq.* (B.3)*) is a generalisation.*

Definition B.1.2. *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:*

$$S_{s,t}\big(X\big) := \Big(1, X_{s,t}^{(1)}, X_{s,t}^{(2)}, X_{s,t}^{(3)}, \dots\Big) \in T\big(\big(\mathbb{R}^d\big)\big), \tag{B.5}$$

where for $n \geq 1$,

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

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

$$S_{s,t}^{N}\big(X\big) := \left(1, X_{s,t}^{(1)}, X_{s,t}^{(2)}, \dots, X_{s,t}^{(N)}\right) \in T^N\big(\mathbb{R}^d\big), \tag{B.6}$$

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 (Eq. (B.1)). 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

$$\int_0^t \int_0^s \mathrm{d}X_u^i \mathrm{d}X_s^j + \int_0^t \int_0^s \mathrm{d}X_u^j \mathrm{d}X_s^i = X_t^i X_t^j,$$

for $i, j \in \{1, \cdots, d\}$, which follows by the integration-by-parts formula. To this end, we will define the logarithm map on the depth-N 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 B.1.3 (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:*

$$\log(\boldsymbol{a}) := \log(a_0) + \sum_{n=1}^{\infty} \frac{(-1)^n}{n} \left(\mathbf{1} - \frac{\boldsymbol{a}}{a_0}\right)^{\otimes n}, \tag{B.7}$$

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 B.1.4 (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 (Eq.* (B.7)*) as*

$$\log^N(\boldsymbol{a}) := P_N\big(\log(\widetilde{\boldsymbol{a}})\big), \tag{B.8}$$

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 B.1.5. *The* log-signature *of a finite length path $X : [0, T] \to \mathbb{R}^d$ over the interval $[s, t]$ is defined as $\mathrm{LogSig}_{s,t}(X) := \log(S_{s,t}(X))$, where $S_{s,t}(X)$ denotes the path signature of X given by Definition definition B.1.2. Likewise, the depth-N (or truncated) log-signature of X is defined for each $N \geq 1$ as $\mathrm{LogSig}^N_{s,t}(X) := \log^N(S^N_{s,t}(X))$.*

In this section, we view each $\mathrm{LogSig}^N_{s,t}(X)$ as an element of $T^N\big(\mathbb{R}^d\big)$ to simplify the definition of the log-ODE method. That said, this is equivalent to the definition used in the main body of the paper, which defines the log-signature as a map from $X\colon [0, T] \to \mathbb{R}^d$ to $\mathbb{R}^{\beta(d,N)}$. This corresponds to the interpretation of a log-signature as an element of a certain free Lie algebra (see, for example, Lyons et al. (2007a); Reizenstein (2017) for details). The exact form of $\beta(d, N)$ is given by

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

with μ the Möbius function. The precise order of this remains an open question.

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 (Eq. (B.1)).

Definition B.1.6 (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 (Eq. (B.1)).

Definition B.1.7 (**The Taylor method**). *Given the CDE (Eq.* (B.1)*), 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*

$$\mathrm{Taylor}(Y_s, f, S^N_{s,t}(X)) := \sum_{k=0}^{N} f^{\circ k}(Y_s)\pi_k\big(S^N_{s,t}(X)\big), \tag{B.9}$$

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 B.1.8 (**The Log-ODE method**). *Using the Taylor method (Eq.* (B.9)*), we can define the function $\widehat{f} : \mathbb{R}^n \to L(T^N(\mathbb{R}^d), \mathbb{R}^n)$ by $\widehat{f}(z) \coloneqq \mathrm{Taylor}(z, f, \cdot)$. By applying $\widehat{f}$ to the truncated log-signature of the path X over an interval $[s,t]$, we can define the following ODE on $[0,1]$*

$$\begin{aligned} \frac{\mathrm{d}z}{\mathrm{d}u} &= \widehat{f}(z)\mathrm{LogSig}^N_{s,t}(X), \\ z(0) &= Y_s. \end{aligned} \tag{B.10}$$

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

$$\mathrm{LogODE}(Y_s, f, \mathrm{LogSig}^N_{s,t}(X)) := z(1). \tag{B.11}$$

Remark B.1.1. *Our assumptions of f ensure that $z \mapsto \widehat{f}(z)\mathrm{LogSig}^N_{s,t}(X)$ is either globally bounded and Lipschitz continuous or linear. Hence both the Taylor and log-ODE methods are well defined.*

Remark B.1.2. *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 Lyons et al. (2007a)). 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 $\mathrm{LogSig}^N_{s,t}(X)$ 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 $\widehat{f}(z)\mathrm{LogSig}^N_{s,t}(X)$. In particular, this leads to our construction of the log-ODE (Eq.* (5.2)*) using the Lyndon basis of the free Lie algebra (see Reizenstein (2017) for a precise description of the Lyndon basis). We direct the reader to Lyons (2014b) and Boutaib et al. (2014) for further details on this Lie theory.*

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

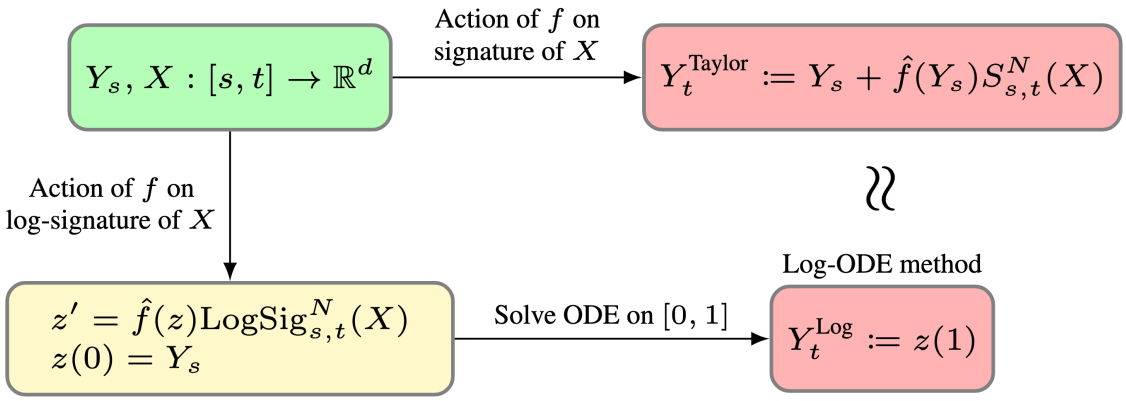


Figure B.1.1 – Illustration of the log-ODE and Taylor methods for controlled differential equations.

Example B.1.1 (The "increment-only" log-ODE method)**.** *When $N = 1$, the ODE (Eq.* (B.10)*) becomes*

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

Therefore we see that this "increment-only" log-ODE method is equivalent to driving the original CDE (Eq. (B.1)*) 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 Wong and Zakai (1965) for further details).*

Example B.1.2 (An application for SDE simulation)**.** *Consider the following affine SDE,*

$$\begin{aligned} \mathrm{d}Y_t &= a(b - y_t)\,\mathrm{d}t + \sigma y_t \circ \mathrm{d}W_t, \\ y(0) &= y_0 \in \mathbb{R}_{\geq 0}, \end{aligned} \tag{B.12}$$

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 (B.12) 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 (B.10) becomes

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

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

$$A_{s,t} := \int_s^t W_{s,r} \, \mathrm{d}r - \frac{1}{2} h W_{s,t},$$
$$L_{s,t}^{(1)} := \int_s^t \int_s^r W_{s,v} \circ \mathrm{d}W_v \, \mathrm{d}r - \frac{1}{2} W_{s,t} A_{s,t} - \frac{1}{6} h W_{s,t}^2,$$
$$L_{s,t}^{(2)} := \int_s^t \int_s^r W_{s,v} \, \mathrm{d}v \, \mathrm{d}r - \frac{1}{2} h A_{s,t} - \frac{1}{6} h^2 W_{s,t}.$$

In Foster et al. (2020), the depth-3 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.

For more information on convergence properties refer to Morrill et al. (2021b).

B.2 Experimental details

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

Data splits Each dataset was split into a training, validation, and testing dataset with relative sizes 70%/15%/15%.

Hyperparameter selection Hyperparameters were selected for the Neural CDE model by performing a grid search, with a step size 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. We additionally performed a separate hyperparameter selection for the ODE-RNN model. The Neural RDE models then use the same hyperparameters as the Neural CDE model.

Normalisation The training splits of each dataset were normalised to zero mean and unit variance. The statistics from the training set were then used to normalise the validation and testing datasets.

Architecture We give a graphical description of the architecture used for updating the Neural CDE hidden state in figure B.2.1. The input is first run through a multilayer perceptron with n layers of size h, with with n, h being hyperparameters. ReLU nonlinearities are used at each layer except the final one, where we instead use a tanh nonlinearity. The goal of this is to help prevent term blow-up over the long sequences.

Note that this is a small inconsistency between this work and the original model proposed in Kidger et al. (2020). Here, we applied the tanh function as the final

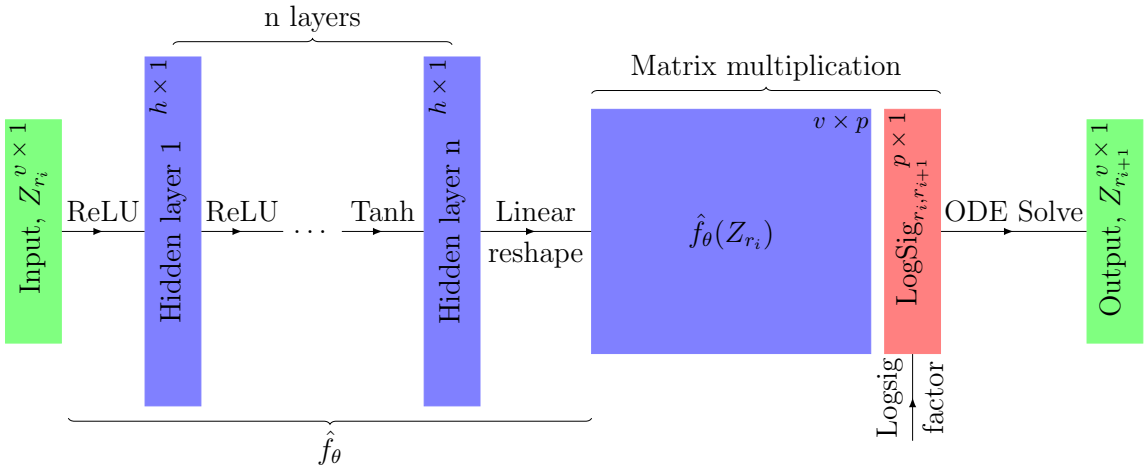


Figure B.2.1 – Overview of the hidden state update network structure. We give the dimensions at each layer in the top right hand corner of each box.

hidden layer nonlinearity, whilst in the original paper the tanh nonlinearity is applied after the final linear map. Both methods are used to constrain the rate of change of the hidden state; we do not know of a reason to prefer one over the other.

Note that the final linear layer in the multilayer perceptron is reshaped to produce a matrix-valued output, of shape $v \times p$. (As $\widehat{f_\theta}$ is matrix-valued.) A matrix-vector multiplication with the log-signature then produces the vector field for the ODE solver.

ODE Solver All problems used the 'rk4' solver as implemented by `torchdiffeq` (Chen, 2018) version 0.0.1.

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 0.032 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.

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

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 B.2.1 – Hyperparamter selection results for the EigenWorms dataset. The blue values denote the selected hyperparameters.

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 B.2.1. The dimension of each of these 'hidden hidden' layers with be this value multiplied by 'Hidden dimension'.

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 (B.2.1, B.2.3) with bolded values to show which values were eventually selected.

B.3 Experimental Results

Here we include the full breakdown of all experimental results. Tables B.3.1 and B.3.2 include all results from the EigenWorms and BIDMC datasets respectively.

Validation loss			Hidden dim	Total params
RR	HR	SpO2		
3.00	**12.82**	**3.37**	32	3871
3.00	12.82	3.37	64	9759
2.82	12.82	3.37	128	27679
2.49	12.82	3.37	192	53791
2.52	12.82	3.37	256	88095
2.50	12.82	3.37	320	130591
2.83	12.82	3.37	388	184719

Table B.2.2 – Hyperparameter selection results for the folded ODE-RNN model on the BIDMC problem. Bold values indicate selected hyperparamter values. The ODE-RNN model failed to train effectively for the HR and SpO2 problems which is why the validation losses are the same (to 2dp).

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 B.2.3 – 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.

Validation loss			Hidden dim	Total params
RR	HR	SpO2		
3.00	**12.82**	**3.37**	32	3871
3.00	12.82	3.37	64	9759
2.82	12.82	3.37	128	27679
2.49	12.82	3.37	192	53791
2.52	12.82	3.37	256	88095
2.50	12.82	3.37	320	130591
2.83	12.82	3.37	388	184719

Table B.2.4 – Hyperparameter selection results for the folded ODE-RNN model on the BIDMC problem. Bold values indicate selected hyperparamter values. The ODE-RNN model failed to train effectively for the HR and SpO2 problems which is why the validation losses are the same (to 2dp).

Model	Step	Test Accuracy	Time (Hrs)	Memory (Mb)
ODE-RNN (folded)	1	Memory Error	Memory Error	Memory Error
	2	36.8 ± 1.5	1.6	7170.1
	4	35.0 ± 1.5	0.8	3629.3
	6	36.8 ± 1.5	0.5	2448.6
	8	36.8 ± 1.5	0.4	1858.8
	16	32.5 ± 3.0	0.2	973.5
	32	32.5 ± 1.5	0.1	532.2
	64	41.0 ± 4.4	0.1	311.2
	128	47.9 ± 5.3	0.0	200.8
	256	46.2 ± 0.0	0.0	147.0
	512	47.9 ± 10.4	0.0	124.5
	1024	44.4 ± 7.4	0.0	122.4
	2048	48.7 ± 6.8	0.0	137.2
NCDE	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
	6	65.8 ± 12.9	2.6	31.5
	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
	1024	41.9 ± 14.6	0.0	2.7
	2048	38.5 ± 5.1	0.0	2.6
$NRDE_2$	2	**76.1 ± 13.2**	9.8	354.3
	4	**83.8 ± 3.0**	2.4	180.0
	6	**76.9 ± 6.8**	2.0	82.2
	8	**77.8 ± 5.9**	2.1	94.2
	16	**78.6 ± 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	**76.1 ± 5.9**	0.2	7.8
	256	**72.6 ± 12.1**	0.1	8.9
	512	**69.2 ± 11.8**	0.0	7.6
	1024	**65.0 ± 7.4**	0.0	6.9
	2048	**67.5 ± 3.9**	0.0	6.5
$NRDE_3$	2	66.7 ± 4.4	7.4	1766.2
	4	76.9 ± 9.2	2.8	856.8
	6	70.9 ± 1.5	1.4	606.1
	8	70.1 ± 6.5	1.3	460.7
	16	73.5 ± 3.0	1.4	243.7
	32	**75.2 ± 3.0**	0.6	134.7
	64	**74.4 ± 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
	1024	59.8 ± 3.9	0.0	29.6
	2048	61.5 ± 4.4	0.0	27.7

Table B.3.1 – Mean and standard deviation of test set accuracy (in %) over three repeats, as well as memory usage and training time, on the EigenWorms dataset for depths 1–3 and a small selection of step sizes. The bold values denote that the model was the top performer for that step size.

Depth	Step	L^2			Time (H)			Memory (Mb)
		RR	HR	SpO_2	RR	HR	SpO_2	
ODE-RNN (folded)	1	Error	13.06 ± 0.0	Error	Error	10.4	Error	3654.0
	2	Error	13.06 ± 0.0	Error	Error	5.5	Error	1840.4
	4	2.76 ± 0.14	13.06 ± 0.0	3.3 ± 0.0	3.0	2.7	2.1	1809.0
	8	2.47 ± 0.35	13.06 ± 0.0	3.3 ± 0.0	1.5	1.2	0.9	917.2
	16	2.21 ± 0.75	13.06 ± 0.0	3.3 ± 0.0	2.2	0.7	0.4	471.9
	32	1.82 ± 0.64	13.06 ± 0.0	3.3 ± 0.0	0.7	0.3	0.2	249.4
	64	1.6 ± 0.22	13.06 ± 0.0	3.3 ± 0.0	0.5	0.1	0.1	137.0
	128	1.62 ± 0.07	13.06 ± 0.0	3.3 ± 0.0	0.2	0.1	0.1	81.9
	256	1.57 ± 0.04	7.04 ± 1.04	**1.43 ± 0.11**	0.1	0.1	0.1	53.8
	512	1.66 ± 0.06	6.75 ± 0.9	1.98 ± 0.31	0.0	0.1	0.1	40.4
	1024	**1.69 ± 0.02**	8.4 ± 0.28	2.05 ± 0.14	0.0	0.0	0.0	36.2
	2048	1.75 ± 0.03	9.2 ± 0.27	**2.24 ± 0.11**	0.0	0.0	0.0	39.6
NCDE	1	2.79 ± 0.04	9.82 ± 0.34	2.83 ± 0.27	23.8	22.1	28.1	56.5
	2	2.87 ± 0.03	11.69 ± 0.38	**3.36 ± 0.2**	19.3	9.6	8.8	32.6
	4	**2.92 ± 0.08**	11.15 ± 0.49	3.69 ± 0.06	5.3	5.7	3.2	20.2
	8	2.8 ± 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	2.9 ± 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 ± 0.1	3.08 ± 0.1	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
	1024	2.67 ± 0.12	11.55 ± 0.03	2.91 ± 0.12	0.1	0.1	0.1	8.4
	2048	2.48 ± 0.03	12.03 ± 0.2	3.25 ± 0.01	0.0	0.1	0.0	8.2
$NRDE_2$	2	2.91 ± 0.1	11.11 ± 0.23	3.89 ± 0.44	12.7	9.3	8.2	58.3
	4	**2.92 ± 0.04**	11.14 ± 0.2	4.23 ± 0.57	18.1	5.0	3.4	34.0
	8	2.63 ± 0.12	8.63 ± 0.24	2.88 ± 0.15	2.1	3.4	3.3	21.8
	16	1.8 ± 0.07	5.73 ± 0.45	1.98 ± 0.21	2.2	1.4	2.5	16.0
	32	1.9 ± 0.02	7.9 ± 1.0	1.69 ± 0.2	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	2.1 ± 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
	1024	1.93 ± 0.11	6.0 ± 0.19	2.41 ± 0.07	0.1	0.1	0.2	10.2
	2048	**2.03 ± 0.03**	**7.7 ± 1.46**	2.55 ± 0.03	0.1	0.1	0.1	10.2
$NRDE_3$	2	**2.82 ± 0.08**	**11.01 ± 0.28**	4.1 ± 0.72	8.8	9.4	6.9	125.2
	4	2.97 ± 0.23	**10.13 ± 0.62**	**3.56 ± 0.44**	3.2	4.1	2.6	71.6
	8	**2.42 ± 0.19**	**7.67 ± 0.4**	**2.55 ± 0.13**	2.9	3.2	3.1	43.3
	16	**1.74 ± 0.05**	**4.11 ± 0.61**	**1.4 ± 0.06**	1.4	1.4	6.5	29.1
	32	**1.67 ± 0.01**	**4.5 ± 0.7**	**1.61 ± 0.05**	1.3	1.8	7.3	20.5
	64	**1.53 ± 0.08**	**3.05 ± 0.36**	**1.48 ± 0.14**	0.4	1.9	3.3	17.9
	128	**1.51 ± 0.08**	**2.97 ± 0.45***	**1.37 ± 0.22**	0.5	1.7	1.7	17.3
	256	**1.51 ± 0.06**	**3.4 ± 0.74**	1.47 ± 0.07	0.3	0.7	0.6	16.6
	512	**1.49 ± 0.08***	**3.46 ± 0.13**	**1.29 ± 0.15***	0.3	0.4	0.4	15.4
	1024	1.83 ± 0.33	**5.58 ± 2.5**	**1.72 ± 0.31**	0.2	0.1	0.1	15.7
	2048	2.31 ± 0.27	9.77 ± 1.53	2.45 ± 0.18	0.1	0.1	0.1	15.6

Table B.3.2 – Mean and standard deviation of the L^2 losses on the test set for each of the vitals signs prediction tasks (RR, HR, SpO_2) on the BIDMC dataset, across three repeats. Only mean times are shown for space. The memory usage is given as the mean over all three of the tasks as it was approximately the same for any task for a given depth and step. Error denotes that the model could not be run within GPU memory. The bold values denote the algorithm with the lowest test set loss for a fixed step size for each task.

Appendix C

The Generalised Signature Method

C.1 Implementation details

C.1.1 General notes

Code All the code for this project is available at:
`https://github.com/jambo6/generalised-signature-method`.

Libraries The machine learning framework used was PyTorch (Paszke et al., 2017) version 1.3.1. Signatures and logsignatures were computed using the Signatory library (Kidger and Lyons, 2021) version 1.1.6. Scikit-learn (Pedregosa et al., 2011) version 0.22.1 was used for the logistic regression and random forest models. The experiments were tracked using the Sacred framework (Greff et al., 2017) version 0.8.1.

Normalisation Every dataset was normalised so that each channel has mean zero and unit variance.

Architectures Two different GRU models were used on every dataset; a 'small' one with 32 hidden channels and 2 layers, and a 'large' one with 256 hidden channels and 3 layers.

Likewise, two different Residual CNN models were considered. The 'small' one used 6 blocks, each composed of batch normalisation, ReLU activation, convolution with 32 filters and kernel size 4, batch normalisation, ReLU activation, and a final convolution with 32 filters and kernel size 4, so that there are also 32 channels along the 'residual path'. A final two-hidden-layer neural network with 256 neurons was placed on the output. The 'large' is similar, except that it used 128 filters in both the blocks and the residual path, had 8 blocks, used a kernel size of 8, and the final neural network had 1024 neurons.

The logistic regression was performed three times with different amounts of L^2 regularisation, with scaling hyperparameters of 0.01, 0.2 and 1; for every experiment the regularization hyperparameter achieving the best accuracy on the test set was used.

The random forest used the default Scikit-learn implementation with a maximum depth of 6 and 100 trees.

Optimiser The GRU and CNN were optimised using Adam (Kingma and Ba, 2015). The learning rate was 0.01 for the GRU, and 0.001 for the residual CNN. The small models were trained for a maximum of 500 epochs; the large models were trained for a maximum of 1000 epochs. The learning rate was decreased by a factor of 10 if validation loss did not improve over a plateau of 10 epochs. Early stopping was used if the validation loss did not improve for 30 epochs. After training the parameters were always rolled back to those that demonstrated the best validation loss over the course of training. The batch size used varied by dataset; in each it was taken to be the power of two that meant that the number of batches per epoch was closest to 40.

Computing infrastructure Experiments were run on an Amazon AWS G3 Instance (g3.16xlarge) equipped with 4 Tesla M60s, parallelized using GNUParallel (Tange, 2011).

C.1.2 Analysis of variations of the signature method

Splits The UEA archive comes with a pre-defined train-test split, which we respect. We take an 80%/20% train/validation split in the training data, stratified by class label. For the Human Activities and Postural Transitions dataset, we take a 60%/15%/25% train/validation/test split from the whole dataset. For the Speech Commands dataset, we take a 68%/17%/15% train/validation/test split from the whole dataset. (These somewhat odd choices corresponding to taking either 25% or 15% of the dataset as test, and then splitting the remaining 80%/20% between train and validation.) These train/validation splits are only used for the training of the GRU and CNN classifiers.

Combinations In total we tested 8569 different combinations.

The variations tested are divided into groups. The first group consists of the sensitivity-adding augmentations, namely time, basepoint and invisibility-reset. Relative to the baseline model, we test every possible combination of these. (Including using none of them.)

The second group consists of those other augmentations, namely the lead-lag, singleton coordinate projection, pair coordinate projection, triplet coordinate projection, random projections, learnt projections, and multi-headed stream preserving neural networks, and finally also the case of no additional augmentation.

For the random projections, we consider four possibilities, with $e \in \{3, 6\}$ and $p \in \{2, 5\}$, all relative to the baseline model.

For no additional augmentation, lead-lag, coordinate projections, learnt projections, and the multi-headed stream preserving neural networks, we compose them with the time, time+basepoint and time+invisibility-reset augmentations (the clear best three from the first group), all relative to the baseline model.

For the learnt projections, we consider four different possibilities corresponding to $e \in \{3, 6\}$ and $p \in \{2, 5\}$; together with the time/time+basepoint/time+invisibility-reset cases this yields a total of twelve possibilities.

For the multi-headed stream-preserving neural networks, we again consider four different possibilities corresponding to $e \in \{3, 6\}$ and $p \in \{2, 5\}$, for a total of twelve possible augmentation strategies. In each the neural network operates elementwise, so as to map one sequence to another, and is given by a feedforward neural network of three hidden layers separated by ReLU activation functions. When $e = 3$ the hidden layers have 16 neurons each, and when $e = 6$ they have 32 neurons each.

For both the learnt projections and multi-headed stream-preserving neural networks, training these requires backpropagating through the model, so these were only considered for the GRU and residual CNN model. (The logistic regression model would in principle be possible as well, except that we ended up implementing this through Scikit-learn rather than PyTorch.)

We note that there are a great many possible ways of doing stream preserving neural networks, of which these are a small fraction. Their relatively weak performance here may likely be improved upon with greater tuning on an individual task, or the selection of better final models than were considered here.

The third group consisted of the different windows. Recall that the baseline model used a global window; we then consider varying this to two possible sliding windows, two possible expanding windows, and three possible dyadic windows. The two possible sliding/expanding windows are chosen so that either 5 or 20 windows are applied across the full length of the dataset. The three possible dyadic windows are depths 2, 3, 4. Thus in total there are 8 possible window combinations we consider.

The fourth group consists of rescaling options, namely no rescaling, pre-signature rescaling, and post-signature rescaling.

Omissions For the empirical study on the variations on the signature method, we excluded those UEA datasets with a dimension d over 60, so as to reduce the computational cost. This results removes 6 of the 30 datasets from the study, namely DuckDuckGeese, FaceDetection, Heartbeat, InsectWingbeat, MotorImagery, and PEMS-SF. These were nonetheless used in the demonstration of performance of the canonical signature method in Figure 6.6.2. Furthermore those combinations of dataset/variation/model which produced more than 10^5 signature features were omitted, to keep the computation managable. See Table C.1.1.

Table C.1.1 – Summary of the number of combinations considered and omitted.

Variations	# Variations	# Classifiers	# Omitted Combinations	# Total Combinations
Basic augmentations (Figure 6.5.1)	6	6	54	936
Other augmentations (Figure 6.5.2)				
Lead-lag/ None	3	6	100/27	468
Coordinates projection (1)/(2)/(3)	3	6	12/12/54	468
Random projections	4	6	32	624
Learnt projections / MHSP	12	4	348/176	1248
Windows (Figure 6.5.3)	8	6	227	1248
Signature/Logsignature transform	12	6	361	1872
Rescalings (Figure 6.5.4)	3	6	12	468
Total			1415	9984

C.1.3 The canonical signature pipeline

For each dataset, we implement the following steps. First, the sequences are augmented with time and basepoint augmentations. Then, we consider every combination of signature depth in $\{1, 2, 3, 4, 5, 6\}$ and hierarchical dyadic window depth in $\{2, 3, 4\}$. For each of these choices, we perform a randomized grid search on a random forest classifier to optimize its number of trees and maximal depth parameters. We test 20 combinations randomly sampled from the following grids:

$$\begin{aligned} \text{n\_trees} &= [50, 100, 500, 1000], \\ \text{max\_depth} &= [2, 4, 6, 8, 12, 16, 24, 32, 45, 60, 80, \text{None}]. \end{aligned}$$

Note that a maximal depth set to 'None' means that the trees are expanded until all leaves contain exactly one sample. Finally, we choose the combination of signature and hierarchical dyadic window depths which maximise the out-of-bag score.

C.2 Additional results

C.2.1 Analysis of variations of the signature method

Running time To get a sense of the cost of each augmentation or window, we present the run times of each augmentation/model combination, and each window/-model combination. (The times for varying between signature and logsignature, and between different rescalings, are largely insignificant.) See Table C.2.1.

The run times are averaged over every UEA dataset. As the datasets are of very different sizes this thus represents quite a crude statistic, and in particular produces very large variances, so these are most meaningful simply with respect to each other.

Sensitivity-inducing augmentations broken down by dataset type Table C.2.2 shows the average rank of each of the first group of augmentations (that add sensitiv-

Table C.2.1 – Average run time (in seconds) for various experiments given as mean (std), averaged over all UEA datasets.

	Classifier			
	CNN	GRU	Logistic regression	Random forest
Time augment & Global window (Baseline)	69.8 (98.0)	22.2 (31.8)	2.67 (7.09)	2.23 (4.84)
Augmentation				
None	48.1 (63.5)	16.8 (33.6)	3.55 (9.91)	66.3 (321)
Lead-lag	48.58 (69.99)	15.2 (18.1)	5.76 (11.7)	3.35 (6.04)
Coordinates projection (1)	32.8 (31.49)	13.4 (17.8)	1.37 (4.2)	12.2 (59.3)
Coordinates projection (2)	41.5 (51.4)	22.6 (62.3)	3.01 (8.54)	42.3 (203)
Coordinates projection (3)	41.3 (39.9)	19.1 (24.5)	5.41 (9.76)	6.3 (14.1)
Random projection	62.2 (70.1)	21.1 (31.2)	0.86 (1.25)	1.4 (2.47)
Learnt projection	917 (1288)	752 (972)	–	–
Multi-headed stream-preserving	1051 (1677)	1758 (4442)	–	–
Window				
Sliding	90.6 (120)	79.4 (175)	10.1 (27.4)	6.4 (16.0)
Expanding	102 (133)	68.7 (115)	9.98 (27.2)	7.17 (19.0)
Dyadic	725 (868)	56.9 (65.1)	12.5 (33.2)	7.59 (18.3)

ity to certain kinds of perturbation) by dataset type, where the types are taken from Bagnall et al. (2018). (This may be regarded as a companion to Table 6.3.1.)

It is interesting to note that for EEG data, it seems better not to consider the time augmentation, whereas it is the case for other applications. In particular the combination of time and basepoint augmentations achieve the best ranks for human action and motion recognition (HAR and MOTION in Table C.2.2). Recognizing an action may not be translation-invariant nor invariant by time reparametrization.

Table C.2.2 – Average ranks for different augmentations by type of data. Lower is better.

	Augmentation					
Data type	None	Time	Basepoint	Invisibility-reset	Time + Basepoint	Time + Invisibility-reset
EEG	3.88	3.50	4.00	**2.00**	4.00	3.63
HAR	5.00	2.95	4.85	3.65	**2.00**	2.55
MOTION	5.25	2.75	5.75	3.88	**1.50**	1.88
OTHER	4.43	3.31	4.88	3.19	2.87	**2.31**

Depth study on the signature transform In the main text we focused on the difference between the signature and logsignature transforms, and stated that larger depths must be chosen by a bias-variance tradeoff. Here we consider varying the depth together with the choice of signature or logsignature, and taking the best transform

for each depth. See Figure C.2.1. We see that larger depths do indeed generally correspond to increased performance, up to a point. The optimal depth will depend on the complexity of the task, as the number of features increases exponentially with the depth.

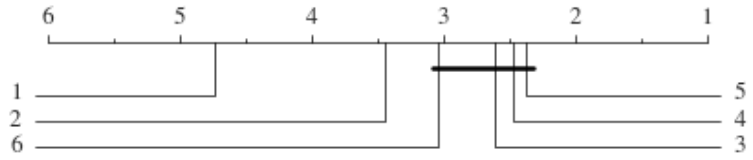


Figure C.2.1 – Critical differences plot for the depth study on the UEA datasets.

C.2.2 Complete results

We present in Tables C.2.3, C.2.4, C.2.5, C.2.6, C.2.7 and C.2.8 the performance of the different signature variations on each dataset. The tables were obtained by maximizing the test accuracy of the signature method over the different classifiers considered. Recall that some values are omitted due to the large number of signature features that would be obtained.

C.2.3 Canonical signature method

In Table C.2.9 we give the full results for our canonical signature method on all UEA datasets, together with the results of Ruiz et al. (2020) used in Figure 6.6.2.

Finally, we give in Table C.2.10 the hyperparameters that were selected for each dataset in the signature pipeline model.

Table C.2.3 – Accuracy of sensitivity-inducing augmentations for each dataset.

	Augmentation					
Dataset	None	Time	Basepoint	Invisibility-reset	Time + Basepoint	Time + Invisibility-reset
ArticularyWordRecognition	96.0	96.3	95.7	96.3	97.7	97.0
AtrialFibrillation	46.7	46.7	40.0	33.3	40.0	40.0
BasicMotions	100.0	100.0	100.0	100.0	100.0	100.0
CharacterTrajectories	88.3	93.2	86.4	88.7	93.8	93.7
Cricket	91.7	94.4	94.4	97.2	97.2	95.8
ERing	80.0	92.6	77.4	89.6	91.9	92.2
EigenWorms	72.5	79.4	74.8	76.3	87.0	81.7
Epilepsy	84.8	89.9	91.3	91.3	97.1	94.9
EthanolConcentration	27.8	29.3	33.5	41.8	34.6	41.4
FingerMovements	55.0	52.0	57.0	58.0	55.0	56.0
HandMovementDirection	29.7	33.8	32.4	33.8	36.5	32.4
Handwriting	21.9	30.8	23.6	24.6	30.6	28.7
JapaneseVowels	85.4	85.1	97.3	98.1	97.3	98.1
LSST	42.0	47.4	44.0	44.4	50.9	48.7
Libras	72.8	84.4	65.0	75.0	80.0	77.2
NATOPS	81.7	88.3	79.4	79.4	91.1	92.2
PenDigits	91.1	97.1	88.3	93.1	96.8	97.1
PhonemeSpectra	4.7	8.2	4.3	5.7	10.0	8.1
RacketSports	78.9	80.3	78.9	82.9	82.9	81.6
SelfRegulationSCP1	81.6	83.3	76.8	84.0	75.4	85.0
SelfRegulationSCP2	57.2	56.7	56.1	56.7	56.1	55.0
SpokenArabicDigits	82.5	85.5	80.5	88.0	85.1	90.1
StandWalkJump	60.0	46.7	40.0	46.7	40.0	46.7
UWaveGestureLibrary	84.1	87.5	79.7	82.8	87.5	83.4
Human Activity	73.0	76.6	92.3	92.2	93.0	93.8
Speech Commands	71.4	75.9	74.7	74.9	79.7	79.5
Average rank	4.69	3.12	4.87	3.29	**2.5**	2.54

Table C.2.4 – Accuracy of other augmentations for each dataset.

	Augmentation							
Dataset	None	Lead-lag	Coordinates projection (1)	(2)	(3)	Random projection	Learnt projection	MHSP
ArticularyWordRecognition	97.7	96.3	83.3	95.7	97.0	95.3	73.7	80.3
AtrialFibrillation	46.7	40.0	53.3	53.3	46.7	66.7	46.7	53.3
BasicMotions	100.0	100.0	80.0	100.0	100.0	100.0	97.5	87.5
CharacterTrajectories	93.8	95.3	43.9	93.2	93.8	93.3	89.6	91.1
Cricket	97.2	98.6	90.3	97.2	95.8	88.9	69.4	56.9
ERing	92.6	94.8	79.3	89.3	91.9	74.4	62.2	61.1
EigenWorms	87.0	87.8	50.4	84.0	89.3	78.6	–	–
Epilepsy	97.1	97.1	55.8	95.7	95.7	81.9	67.4	65.9
EthanolConcentration	41.4	39.9	42.2	43.3	42.2	30.4	32.3	30.0
FingerMovements	56.0	–	59.0	58.0	–	55.0	60.0	65.0
HandMovementDirection	36.5	31.1	31.1	40.5	37.8	37.8	33.8	44.6
Handwriting	30.8	33.5	11.3	27.8	30.0	21.6	12.6	13.2
JapaneseVowels	98.1	97.6	94.1	97.8	97.6	84.1	95.4	95.9
LSST	50.9	55.6	43.5	51.7	52.8	43.7	34.4	39.8
Libras	84.4	86.7	47.8	83.9	85.0	86.7	73.3	81.1
NATOPS	92.2	–	33.3	90.6	91.1	85.6	83.9	81.7
PenDigits	97.1	98.3	60.2	96.8	97.2	96.7	96.5	97.4
PhonemeSpectra	10.0	–	4.5	9.4	10.6	8.9	7.2	7.7
RacketSports	82.9	82.2	53.3	85.5	84.2	75.7	73.7	75.0
SelfRegulationSCP1	85.0	86.0	61.8	85.0	84.0	81.6	86.7	84.6
SelfRegulationSCP2	56.7	57.2	55.6	58.9	55.6	60.6	59.4	57.8
SpokenArabicDigits	90.1	96.6	58.5	86.0	90.0	83.0	88.0	86.0
StandWalkJump	46.7	40.0	40.0	53.3	40.0	53.3	–	–
UWaveGestureLibrary	87.5	88.8	50.6	85.6	87.5	86.2	74.1	75.6
Human Activity	93.8	93.6	75.8	93.2	93.6	69.2	91.3	91.5
Speech Commands	79.7	–	14.9	77.1	–	70.2	–	76.1
Average ranks	2.9	**2.77**	6.52	3.33	3.23	4.71	5.83	5.31

Table C.2.5 – Accuracy of windows for each dataset.

	Window			
Dataset	Global	Sliding	Expanding	Dyadic
ArticularyWordRecognition	96.3	89.3	99.0	99.0
AtrialFibrillation	46.7	46.7	46.7	60.0
BasicMotions	100.0	100.0	100.0	100.0
CharacterTrajectories	93.2	94.6	96.9	97.1
Cricket	97.2	93.1	97.2	95.8
ERing	90.7	88.5	91.9	94.8
EigenWorms	80.2	74.8	78.6	76.3
Epilepsy	89.9	92.8	92.0	94.2
EthanolConcentration	30.4	38.8	30.0	35.7
FingerMovements	50.0	–	–	–
HandMovementDirection	33.8	33.8	36.5	33.8
Handwriting	30.1	21.5	30.2	27.2
JapaneseVowels	85.1	76.5	88.1	89.2
LSST	47.8	43.1	48.3	46.6
Libras	83.3	85.6	91.1	90.0
NATOPS	93.3	84.4	90.6	–
PenDigits	95.8	–	–	97.6
PhonemeSpectra	8.6	9.2	9.6	10.4
RacketSports	82.2	80.3	84.2	88.2
SelfRegulationSCP1	82.6	87.4	84.0	86.7
SelfRegulationSCP2	56.7	60.6	54.4	56.1
SpokenArabicDigits	85.5	91.5	93.7	96.6
StandWalkJump	46.7	53.3	46.7	53.3
UWaveGestureLibrary	86.6	79.4	89.1	89.7
Human Activity	76.1	73.0	80.4	81.7
Speech Commands	75.9	76.5	82.3	83.0
Average ranks	2.83	3.04	2.17	**1.73**

Table C.2.6 – Accuracy of signature and logsignature transforms for each dataset.

	Transform	
Dataset	Signature	Logsignature
ArticularyWordRecognition	97.7	97.3
AtrialFibrillation	60.0	53.3
BasicMotions	100.0	100.0
CharacterTrajectories	93.8	93.8
Cricket	100.0	100.0
ERing	90.0	89.3
EigenWorms	79.4	81.7
Epilepsy	93.5	91.3
EthanolConcentration	31.9	30.0
FingerMovements	59.0	56.0
HandMovementDirection	40.5	40.5
Handwriting	35.3	24.5
JapaneseVowels	85.9	86.8
LSST	52.0	46.4
Libras	90.6	87.8
NATOPS	89.4	91.7
PenDigits	97.8	97.5
PhonemeSpectra	8.9	7.6
RacketSports	85.5	84.9
SelfRegulationSCP1	84.0	83.3
SelfRegulationSCP2	57.2	56.1
SpokenArabicDigits	87.5	85.8
StandWalkJump	53.3	53.3
UWaveGestureLibrary	90.0	86.9
Human Activity	78.7	78.3
Speech Commands	75.9	76.3
Average ranks	**1.25**	1.75

Table C.2.7 – Accuracy of rescaling choices for each dataset.

	Rescaling		
Dataset	None	Post	Pre
ArticularyWordRecognition	97.3	97.0	97.7
AtrialFibrillation	53.3	53.3	46.7
BasicMotions	100.0	100.0	100.0
CharacterTrajectories	94.6	94.6	94.6
Cricket	98.6	97.2	97.2
ERing	93.7	93.7	93.0
EigenWorms	80.9	80.9	79.4
Epilepsy	92.0	92.0	91.3
EthanolConcentration	31.2	31.6	30.0
FingerMovements	54.0	54.0	50.0
HandMovementDirection	35.1	32.4	29.7
Handwriting	36.6	36.4	37.1
JapaneseVowels	87.3	85.9	85.7
LSST	55.8	55.6	55.4
Libras	85.0	86.1	84.4
NATOPS	92.8	92.8	91.7
PenDigits	96.6	96.7	96.7
PhonemeSpectra	8.0	8.1	8.2
RacketSports	84.2	84.2	83.6
SelfRegulationSCP1	79.5	83.3	84.6
SelfRegulationSCP2	56.1	57.2	56.7
SpokenArabicDigits	90.5	90.5	90.2
StandWalkJump	46.7	53.3	46.7
UWaveGestureLibrary	87.5	87.2	87.2
Human Activity	85.0	84.6	85.1
Speech Commands	77.0	75.7	75.9
Average ranks	**1.73**	1.92	2.35

Table C.2.8 – Accuracy of (log)-signature depth for each dataset.

	Depth					
Dataset	1	2	3	4	5	6
ArticularyWordRecognition	83.3	96.0	97.3	97.7	95.3	–
AtrialFibrillation	40.0	40.0	60.0	33.3	40.0	53.3
BasicMotions	70.0	100.0	100.0	100.0	100.0	92.5
CharacterTrajectories	42.3	88.0	93.2	93.8	92.9	93.8
Cricket	30.6	93.1	97.2	98.6	100.0	–
ERing	77.0	89.6	90.0	89.3	88.9	84.8
EigenWorms	46.6	81.7	79.4	79.4	–	–
Epilepsy	50.7	78.3	89.9	93.5	93.5	93.5
EthanolConcentration	25.5	30.8	30.0	31.2	31.9	27.4
FingerMovements	57.0	58.0	59.0	–	–	–
HandMovementDirection	40.5	36.5	37.8	39.2	32.4	–
Handwriting	7.3	22.4	32.4	33.3	35.3	32.7
JapaneseVowels	78.9	85.9	86.8	84.3	81.4	–
LSST	40.9	45.6	47.6	50.6	52.0	44.7
Libras	51.7	77.2	85.0	87.8	88.9	90.6
NATOPS	35.0	86.7	91.7	–	–	–
PenDigits	60.0	90.4	96.9	97.7	97.4	97.8
PhonemeSpectra	4.1	7.6	8.9	–	–	–
RacketSports	44.1	77.0	78.9	84.9	85.5	82.2
SelfRegulationSCP1	53.6	80.2	84.0	83.3	81.9	–
SelfRegulationSCP2	56.1	55.0	56.7	54.4	57.2	–
SpokenArabicDigits	52.1	85.8	85.5	87.5	–	–
StandWalkJump	46.7	46.7	46.7	46.7	53.3	46.7
UWaveGestureLibrary	49.4	83.1	86.6	87.8	90.0	88.1
Human Activity	47.7	78.3	76.0	78.7	78.6	–
Speech Commands	14.8	69.6	76.3	–	–	–
Average ranks	4.73	3.44	2.62	2.48	**2.38**	3.04

	Classification method								
Dataset	DTWD	DTWA	DTWI	HIVE COTE	MLCN	MUSE	TapNet	gRSF	Signature Pipeline
ArticularyWordRecognition	98.7	98.7	98.0	99.0	95.7	**99.3**	95.7	98.3	97.7
AtrialFibrillation	20.0	26.7	26.7	13.3	33.3	40.0	20.0	26.7	**46.7**
BasicMotions	97.5	**100.0**	**100.0**	**100.0**	87.5	**100.0**	**100.0**	**100.0**	**100.0**
Cricket	**100.0**	**100.0**	98.6	98.6	91.7	98.6	**100.0**	98.6	95.8
Epilepsy	96.4	97.8	97.8	**100.0**	73.2	99.3	95.7	97.8	95.7
EthanolConcentration	32.3	31.6	30.4	**79.1**	37.3	47.5	30.8	34.6	43.3
ERing	91.5	92.6	91.9	**97.0**	94.1	97.4	90.4	95.2	94.8
FaceDetection	52.9	52.8	51.3	**65.6**	55.5	63.1	60.3	54.8	61.4
FingerMovements	53.0	51.0	52.0	55.0	**58.0**	55.0	47.0	**58.0**	52.0
HandMovementDirection	18.9	20.3	29.7	44.6	**52.7**	36.5	33.8	41.9	20.3
Handwriting	**60.7**	**60.7**	50.9	48.2	30.9	52.2	28.1	37.5	37.9
Heartbeat	71.7	69.3	65.9	72.2	38.0	71.2	**79.0**	76.1	69.8
Libras	87.2	88.3	89.4	90.0	85.0	89.4	87.8	69.4	**93.9**
LSST	55.1	56.7	57.5	57.5	52.8	64.0	51.3	**58.8**	56.9
NATOPS	88.3	88.3	85.0	88.9	90.0	90.6	81.1	84.4	**92.2**
PenDigits	97.7	97.7	93.9	93.4	**97.9**	96.7	85.6	93.5	97.4
Racketsports	80.3	84.2	84.2	88.8	84.2	**92.8**	87.5	88.2	90.8
SelfRegulationSCP1	77.5	78.5	76.5	85.3	90.8	69.6	**93.5**	82.3	78.8
SelfRegulationSCP2	**53.9**	52.2	53.3	46.1	50.6	52.8	48.3	51.7	50.6
StandWalkJump	20.0	33.3	33.3	33.3	40.0	26.7	13.3	33.3	**46.7**
UWaveGestureLibrary	90.3	90.0	86.9	89.1	85.9	**93.1**	90.0	89.7	90.9
Average Ranks	5.6	5.2	5.9	4.0	5.6	**3.2**	6.4	4.8	4.3

Table C.2.9 – Results of the signature canonical pipeline along with a selection of classifiers from Ruiz et al. (2020) (including the top performing MUSE algorithm) with a Random Forest for the UEA archive.

	Signature hyperparmeters		RF hyperparameters		Other
Dataset	Depth	Dyadic depth	Max depth	Num estimators	Training time (s)
ArticularyWordRecognition	2	2	45	500	60.3
AtrialFibrillation	1	2	None	50	35.9
BasicMotions	2	2	24	100	19.3
CharacterTrajectories	4	2	80	500	181.4
Cricket	2	4	6	500	249.0
DuckDuckGeese	1	2	16	100	140.9
ERing	2	3	8	1000	16.7
EigenWorms	3	3	12	100	250.1
Epilepsy	2	3	8	1000	42.8
EthanolConcentration	2	4	24	1000	454.2
FaceDetection	1	4	8	1000	1816.2
FingerMovements	1	2	4	100	30.8
HandMovementDirection	2	2	None	50	66.3
Handwriting	6	2	32	1000	280.3
Heartbeat	1	4	None	50	45.1
InsectWingbeat	1	3	45	1000	5367.5
JapaneseVowels	2	3	6	1000	95.4
LSST	4	2	60	1000	1590.5
Libras	6	2	None	100	28.4
MotorImagery	1	3	24	50	347.1
NATOPS	2	3	32	1000	37.8
PEMS-SF	1	3	80	1000	252.3
PenDigits	3	2	80	1000	302.3
PhonemeSpectra	2	4	45	1000	2188.7
RacketSports	3	2	None	500	13.9
SelfRegulationSCP1	3	2	None	100	186.6
SelfRegulationSCP2	3	2	6	50	138.1
SpokenArabicDigits	2	3	45	1000	1204.0
StandWalkJump	1	3	2	50	101.5
UWaveGestureLibrary	2	2	60	500	21.8

Table C.2.10 – Hyperparameters used for each dataset in the signature pipeline model.

Appendix D

Improved Prediction of Stress Levels from Breathing Signals using Path Signature Features

D.1 Signature inversion

We suppose that we have access to the depth-N (log-)signature $S^N(X_{\mathbf{x}})$ of a continuous time embedding of some tick data $\mathbf{x}$ with some interpolation scheme; we are assuming we cannot observe the original tick data. Our aim is to reconstruct the tick data $\mathbf{x}$ as close as possible from the truncated (log-)signature.

We begin by setting $\widehat{\mathbf{x}} = (\widehat{x}_0, \dots, \widehat{x}_n)$ where each $\widehat{x}_i \in \mathbb{R}^d$ is initialised to the zero vector. Note that since the signature is unique only up to translation, we cannot specify its initial point without additional information.

We then attempt to optimise $\mathbf{x}$ to minimise the difference between its signature and the known signature.

$$\text{Loss}(\widehat{\mathbf{x}}; S^N(X_{\mathbf{x}})) = \left\| S^N(X_{\widehat{\mathbf{x}}}) - S^N(X_{\mathbf{x}}) \right\|_1 . \tag{D.1}$$

In the case of our breathing example, we really want the construction that also has minimal length. As such, we minimise

$$\text{Loss}(\widehat{\mathbf{x}}; S^N(X_{\mathbf{x}})) = \left\| S^N(X_{\widehat{\mathbf{x}}}) - S^N(X_{\mathbf{x}}) \right\|_1 \left(1 + \lambda \sum_{i=1}^{L-1} \|\widehat{x}_{i+1} - \widehat{x}_i\|_1 \right). \tag{D.2}$$

D.2 Shapley Values

Shapley values are a concept from cooperative game theory that measure how much an individual player has contributed to the outcome of a game. The payout for a

given player is evaluated by considering all possible coalitions of players with and without the given player and examining the outcome.

Suppose we have a collection of players $\{x_1, ..., x_p\}$ playing a cooperative game that results in some payout. The Shapley value for a player x_j is defined as the weighted average of player x_j's contribution to the payout, summed over all possible coalitions.

Let C be the evaluation function that produces a payout value for any subset of the players and $C(\varnothing) = 0$. The Shapley value of x_j is defined

$$\phi_j(v) = \sum_{\mathcal{S} \subseteq \{x_1,...,x_p\} \setminus \{x_j\}} \frac{|\mathcal{S}|!(p - |\mathcal{S}| - 1)!}{p!} \big(C(\mathcal{S} \cup \{x_j\}) - C(\mathcal{S})\big). \tag{D.3}$$

We see that ϕ_j is a weighted sum evaluating the effect of including the player x_j over all possible coalitions of players.

Shapley values have a strong axiomatic justification, they are unique under the four following axioms:

Axiom D.2.1 (Dummy)**.** *A player that doesn't contribute to any subset of players receives zero attribution.*

Axiom D.2.2 (Efficiency)**.** *The sum of the Shapley values of all players equals the value attained on inclusion of all players. In this way, the gain is exactly distributed among players.*

Axiom D.2.3 (Symmetry)**.** *If x_i and x_j are two players who are equivalent in the sense that $C(\mathcal{S} \cup \{x_i\}) = C(\mathcal{S} \cup \{x_j\})$ for every subset $\mathcal{S}$ not containing x_i or x_j then $\phi_i = \phi_j$.*

Axiom D.2.4 (Linearity)**.** *Attribution of the sum of two games must be the same as the sum of the attributions for each of the games.*

$$\phi_i(C + K) = \phi_i(C) + \phi_i(K) \tag{D.4}$$

D.2.1 Shapley values for ML interpretability

To make the link between the cooperative game theory interpretation of the Shapley values and machine learning, we need to think of our features $\mathbf{x}$ as the individual players, and the payout as the model output. Take as a concrete example the task of estimating the daily sales of a supermarket based on a collection of features such as `num_people_entered_the_store`, `time_of_year`, `size_of_store` and so on. We wish to estimate the contribution each of these features has on the payout, in this case the daily sales. This can be done with Shapley values, our aim will be to evaluate the contribution of each feature against the daily sales using Equation (D.3).

In an ideal world, we would use Equation (D.3) directly, however, note that the Shapley values as defined here requires an evaluation of the value function with players

being removed. In general, a machine learning model requires inputs for all features that it has been trained with. We therefore have to fill features that are not part of the current coalition with random samples from the dataset.

To put this in a more concrete context for ML, we now suppose we have a collection of features $\{x_1, ..., x_p\}$ and a model f_y that is used to predict some target variable y. Our aim is to estimate

$$\phi_y(x_i; x) = \sum_{i \notin \mathcal{S}} \frac{|\mathcal{S}|!(p - |\mathcal{S}| - 1)!}{p!} \big(f_y(x_\mathcal{S} \cup x_i) - f_y(x_\mathcal{S})\big), \tag{D.5}$$

for each x_i. The factor $f_y(x_\mathcal{S} - f_y(x_\mathcal{S}))$ is the marginal contribution to $f_y(x)$ that the feature x_i makes when added to the coalition $\mathcal{S}$.

Štrumbelj and Kononenko (2014) showed that a simple way of estimating this is by Monte-Carlo sampling with

$$\hat{\phi_j} = \frac{1}{M} \sum_{m=1}^{M} \big(f_y(x_{j+}^m) - \hat{f}(x_{j-}^m)\big), \tag{D.6}$$

where $f_y(x_{j+}^m)$ is the prediction for the point x, but with a random number of feature values replaced by the feature values from some randomly sampled datapoint z, except for the feature j. The $f_y(x_{j-}^m)$ value is the same except its value for the feature j is taken to be that of sampled datapoint z.

This is the procedure used in Chapter 8. For references we recommend Lundberg and Lee (2017); Lundberg et al. (2018a,b).