

NCDEs - A state-of-the-art prediction model over irregular time series.



Jad Mounayer
St. Annes College
University of Oxford

A thesis submitted for the degree of
M.Sc. in Mathematical Modelling and Scientific Computing

Long vacation 2023

Acknowledgements

I am grateful to all those with whom I had the pleasure to work on this project.

From Oxford's side: I would like to thank Professor Terry Lyons for teaching me how to think like a true mathematician and to be patient when confronting a problem. Dr. Elena Gal for supporting me, especially near the beginning when my knowledge was limited, and for teaching me how to think out of the box. Ben Walker for never getting bored of my numerous questions and always being ready to help. Dr. Yixuan for continuously working on maintaining a good relationship between Oxford and Roche.

From Roche's side: I would like to thank Dr. Marius Garmhausen for all the support, whether it was walking me step by step to access Roche's datasets or other valuable advice for presentations and data analysis. Dr. Otto Fajardo for supporting me whenever I needed help with any Roche-owned code.

I am grateful for all the support my girlfriend Joyce, my parents Alain Soura, and my sisters Elsa and Joy gave me. I am thankful for all the help from my colleagues in Oxford, whether students of the MMSC, other maths courses, or St Annes College.

Abstract

This project is based on Hoffman-La Roche’s previous work, where they used irregular time series to perform survival predictions for patients diagnosed with lung cancer. The models that the research group in Roche used were a Multilayer Perceptron (MLP) [34] as a baseline and a Random Forest (RF) [5] as the state-of-the-art. However, the irregularity made these models inefficient, especially on longer time series. Roche’s work had two main limitations:

1. Stuck in early prognostic: Early prognostic means on short time series at the beginning of the patient journey, before any treatment has started. Since the longer the time series, the worse the models used by Roche were performing, they could not move along the patient journey. Hence, they were only learning on a small part of the data.
2. Missing trends: MLPs and RFs are not designed to deal with time series. Hence, they miss all sorts of information. For instance, the time difference between two observations or the fact that multiple observations have been recorded simultaneously.

The team in Roche knew the potential of using a model that adapts to irregular time series: Neural Controlled Differential Equations (NCDEs). Accordingly, they tried to apply an NCDE model, yet it struggled to learn and was outperformed by other alternatives. In this project, we show, both in theory and in practice, that even though including time series trends in training using NCDEs is not trivial, it is worth it.

Throughout the project, we first performed a detailed data analysis showing that the dataset used by Roche was closer to tabular data than it is to time series. We then extract a better subset of the data and perform two training stages. In the first stage, we demonstrate the fact that NCDEs capture trends that are missed by MLPs. In the second stage, NCDEs achieve state-of-the-art performance.




We then conclude that there is relevant information in the time series that Random Forests (currently state-of-the-art) miss. Therefore, we propose and implement a technique to introduce some of this information as input to Random Forests using the path signature. The resulting model, Random Forest with Signature (RFS), outperforms every other alternative.

Contents

1	Background	1
1.1	Introduction	1
1.2	Problem Formulation	2
1.3	Organization	3
1.4	Definitions	4
1.5	Path Signature	8
1.5.1	Introduction to Path Integrals	8
1.5.2	Geometric interpretation	9
1.5.3	Signature Properties	11
1.6	Controlled Differential Equations (CDEs)	17
1.6.1	Picard's iterations	18
1.6.2	Relationship to the Signature	18
1.6.3	CDE Properties	20
2	Methodology	25
2.1	Multilayer Perceptron	25
2.1.1	What is a Neural Network?	25
2.1.2	Data pre-processing	26
2.1.3	Forward propagation	27
2.1.4	Backward propagation	27
2.1.5	Batching	29
2.1.6	Cross Validation	29
2.1.7	Dropout	30
2.2	Continuous Models	31
2.2.1	Neurally Ordinary Differential Equations	31
2.2.2	Neurally Controlled Differential Equations	33
2.3	Ensemble Models	37

2.3.1	Random Forests	37
2.3.2	Signature as a feature map	38
2.4	Metric for evaluation	38
3	Application	40
3.1	Data analysis	40
3.2	Results	45
3.2.1	Training over first values	46
3.2.2	Training over time series	48
3.2.3	Additional Work	52
4	Future Work and Conclusion	53
4.1	Future Work	53
4.2	Conclusion	54
A	Useful details	55
A.1	Tuning parameters	55
A.2	Models parameters	57
A.3	Learning rate for NCDEs	58
A.4	Further Data Analysis	59
A.5	GPU parallelisation	59
A.6	Stacking Parametrization Vector	60
	References	61

List of Figures

1.1	Approximation of the integral using partitions of a dissection.	7
1.2	Graph showing the interpretation of Signature terms geometrically. Note that the path is drawn on an $X_2 - X_1$ axis to the left and $X_1 - X_1$ axis to the right.	10
1.3	Diagram illustrating the logic behind the proof of universality for CDEs.	24
2.1	Example of a Multilayer Perceptron with only two hidden layers of depth four.	26
2.2	Diagram illustrating which parameters are needed in forward (top) and backward (bottom) propagations ($dV _i$ is short for $\frac{\partial \mathcal{L}}{\partial V} _i$).	28
2.3	Diagram showing how a Cross-validation with four splits occurs. It is important to understand that the trainings do not depend on each other. However, they are compared to show how mixed the data is and how much the randomization affects the results.	30
2.4	Figure showing sample discrete observations (top left) and their corre- sponding interpolations: Regular Linear (top right), Rectilinear (bot- tom left), Hermite cubic spline with backward finite differences (bot- tom right). We use the colors  for observations,  for hypothetical values, and  for the approximation of a missing value.	35
2.5	Example of a Receiver Operating Characteristic (ROC) curve.	39
3.1	Histogram showing how often patients repeat the two most abundant tests in the old subset of the data.	41
3.2	Figure showing the distribution of the number of tests for the 36 most popular tests (to the left) and the remaining tests (to the right). . . .	42
3.3	Diagram illustrating how the dataset was chosen. The brackets show how the size of the tensor containing the data is changing with every step.	44

3.4	Figure showing the distribution of the number of tests over a time period of $[-90, 90]$ (to the left), and the number of patients witnessing an event/ meeting all criteria when extending the window (to the right).	44
3.5	Figure showing the time series we got after the dataset has been chosen for the most/least popular labs amongst the 20 (to the left and right, respectively).	45
3.6	Figure showing an example of the rectilinear interpolation used when training using the first values only.	46
3.7	Diagram depicting how the signature was used as a feature map for Random Forests, we chose to take the signature of every 2 labs separately.	49
3.8	ROC curves for all three models when training over the entire dataset.	50
3.9	ROC curves for RF and RFS when training over the entire dataset.	51
A.1	Graph showing the effect of learning rate on a batch's loss.	58
A.2	Visualisation of the frequency of patients taking tests.	59

List of Tables

3.1	Table showing the result for all the models proposed when training is done over the first values only. The average and STD are over five splits (cross-validation) for NCDE and MLP and 50 random states for RF.	48
A.1	Table showing the results using NCDEs for different parameters such as the method for solving ODEs, the time step, or the number of epochs when used over the window $[-50, 40]$	56
A.2	Table summarizing the increase in efficiency when parallelizing over GPUs.	60
A.3	Table Illustrating how changes in the parameterization vector affect training	60

Nomenclature

Mathematics

(f^0, \dots, f^k)	Collection of a vector field (gs as well)
δ	A map from $[t_0, t_n]$ to $[t_0, t_n]$
η, ξ	Real numbers with $\eta < \xi$
\hat{X}	A path X with at least one monotone channel.
\mathcal{D}	Dissection with partitions $r_0 < \dots < r_k$
$\mathcal{K}_S, \hat{\mathcal{K}}$	Compact subsets of $\mathcal{V}^p([t_0, t_n], \cdot)$, and $S(\mathcal{V}^p([t_0, t_n], \cdot))$ respectively.
$\mathcal{L}(\cdot_1, \cdot_2)$	Linear maps from (\cdot_1) to (\cdot_2)
$\mathcal{V}^p([t_0, t_n], \cdot)$	Space of paths from $[t_0, t_n]$ to (\cdot) with finite p -variation
\tilde{f}	A continuous function over $[t_0, t_n]$
\tilde{X}	A discrete set of data points.
$\zeta(\cdot)$	The zeta function defined as $\sum_{n=1}^{\infty} \frac{1}{n^{(\cdot)}}$
C_0, C_1, C_2, C_3	Constants
e	Function that maps \hat{K} to \mathbb{R}
e_*	Function that maps $S(K^*)$ to \mathbb{R}
f	Forcing function (ODE or CDE)
I, J	Multiple Indexes
I_b	Identity matrix of size $b \times b$

M_f	Expression for Picard's iterations
$S(\mathcal{V}^p([t_0, t_n], \cdot))$	Space of the signature of paths $X \in \mathcal{V}^p([t_0, t_n], \cdot)$
$S_{t_0, t_n}^{(i)}(\cdot)$	Truncated path signature of depth i of (\cdot) over the interval $[t_0, t_n]$
$S_{t_0, t_n}^{i_1, \dots, i_k}$	An entry of the path signature $S_{t_0, t_n}^{(i)}(\cdot)$
$S_{t_0, t_n}(\cdot)$	The path signature of (\cdot) over the interval $[t_0, t_n]$
t	Time variable
t_0, t_n	Initial/final times respectively.
$U(\cdot), \mathcal{U}(\cdot)$	Monotonically increasing functions.
u_1, \dots, u_k	Variables of integration
V, W	Banach spaces
X	A path that maps $[t_0, t_n]$ to a Banach space.
X^*	A path X with lines extended from $t_0 - 1$ to t_0
X_*	Control of a differential equation
X_1, \dots, X_k	Coordinate Paths
Y	System Response (ODE or CDE)
Y^i	The i^{th} Picard iterate
Y_0	Initial Condition (ODE or CDE)
Z_1, Z_2	Paths used in Riemann-Stieltjes integral

Machine Learning

α	Learning rate
$\Delta W^{[i]}, \Delta b^{[i]}$	Increments for affine transformation components of hidden layer i
\hat{V}	Expected output
ξ_0	Any function of the initial condition (for NCDEs)
f_θ, \hat{f}_θ	Neural Networks depending on parameters θ
$g^{[i]}$	Activation function for hidden layer i
L	Number of hidden layers
TW	Length of the time window
U_i	Output of hidden layer i before applying activation function
V_0	Neural Network input
V_i	Output of hidden layer i after applying activation function
V_{out}	Neural Network output
w	Number of channels that are represented by the Signature at once.
$W^{[i]}, b^{[i]}$	Components of the affine transformation for hidden layer i
y_i, \hat{y}_i	i^{th} column of V_{out} and \hat{V} respectively.

Chapter 1

Background

1.1 Introduction

$$Y(t) = Y_0 + \int_{t_0}^t f(Y) dt. \quad (1.1)$$

Equation 1.1 might be familiar to many, regardless of whether they are mathematicians or computer scientists. It represents a fundamental mathematical concept—ordinary differential equations (ODEs). ODEs have been important for several centuries as they provide a mathematical framework to describe and predict the behavior of dynamic systems. By capturing rates of change and evolution over time, ODEs enable us to gain insights into complex systems, make predictions, and optimize designs across various disciplines.

A more recent concept is Neural Networks. The main idea of a Neural Network is to introduce a linearly parameterized family of (nonlinear) functions that are dense enough to represent the data, turning many problems into the optimization of parameters (known as weights). Deep Neural Networks iterate the process, resulting in an optimization that requires managing the derivatives of the functions in the many weights.

The accomplishments realized through the utilization of ODEs and Deep Neural Networks have motivated researchers to amalgamate these two concepts, resulting in what is known as Neural Ordinary Differential Equations (NODEs) [7]. NODEs employ a deep neural network to reconstruct the underlying differential equation, subsequently employed as a dense function.

In the context of this project, we adopt a more advanced methodology that employs

controlled differential equations (CDEs) instead. The formulation takes the shape:

$$Y(t) = Y_0 + \int_{t_0}^t f(Y) dX_*, \quad (1.2)$$

where X_* is denoted as the control. The model we will use, Neural Controlled Differential Equations (NCDEs)[21], integrates CDEs within a neural network framework. NCDEs are poised to perform well, particularly when applied to irregular time series data. By using the entire time series as the control, we hope that the Neural Network will be able to capture different trends within the time series, which are missed by other techniques when the time dimension is shrunk.

NCDEs have demonstrated state-of-the-art performance in [21]. However, due to the novelty of the method, the application of NCDEs to real-world time series, precisely survival predictions, has yet to be explored.

1.2 Problem Formulation

This thesis is a continuation of the work done by F. Hoffman-La Roche. The research department at Roche has been working on binary/continuous survival predictions for patients diagnosed with different diseases. The team has only worked on early prognostic by predicting patient information from a small time window before any treatment started. Roche’s main problem was that they could not move along the patient journey since doing so would result in extended time series with higher irregularity, making their models less efficient. Furthermore, both MLPs and RFs are not designed to learn using time series. Accordingly, they miss all sorts of trends, such as the difference in timing between observations, or the fact that multiple observations have been done simultaneously. This is why Roche’s previous work included implementing a model that can adapt to changes within irregular time series: NCDEs. Yet, their model failed and was outperformed by every alternative, motivating them to contact us for a joint project.

In this project, we first conducted detailed data analysis and showed that the subset of the data previously used by Roche was not leading to time series. We then chose a specific prediction task: a binary prediction on whether patients diagnosed with lung cancer would witness an event¹ before or after the median survival time of the population. We then show that even though including time series data in training is not trivial, it is worth it. We illustrate this by proving it theoretically and then

¹Throughout the report, we will refer to death as an event.

showing that it holds in practice by two stages of training. Our findings demonstrate that NCDEs surpass the baseline performance set by a Multilayer Perceptron (MLP) [34], even when considering only initial values (first stage of training). Moreover, NCDEs exhibit superior predictive power over moderately long time series compared to MLPs and achieve state-of-the-art performance (i.e., match the performance of Random Forests (RFs) [5]) in the second stage of training. We deduce that there is valuable information within the time series that Random Forests miss. Therefore, we propose and implement a way, using path signatures², to include some information within the time series as inputs for Random Forests. The resulting model, Random Forest with Signature (RFS), outperforms every alternative on the prediction task chosen.

1.3 Organization

This report aims to achieve the following objectives:

1. Provide a clear explanation of Controlled Differential Equations (CDEs), mainly when the control follows a path with bounded variations (for now, think of this as a restriction on how paths can jump suddenly) (Sections 1.4, 1.5, and 1.6).
2. Describe the models we’re using. We’ll start by giving a brief overview of Neural Networks, focusing on MLPs, which we’ll use as the baseline model. Then, we’ll introduce NCDEs and explain their essential aspects. We’ll also touch on Random Forests, a leading technique in survival predictions (Sections 2.1, 2.2, and 2.3).
3. Describe the evaluation metric: Receiver Operator Characteristic curve (ROC curve) (Section 2.4).
4. Detail our data analysis process and explain how we chose a good subset of the data for training (Section 3.1).
5. Present the results showcasing that NCDEs and the Signature are proper ways to include time series trends in the models. (Section 3.2).
6. Conclude by extracting features from the dataset and outlining our plans to expand the project over the next two months (Sections 4.1 and 4.2).

²For now, think of this as an informative way to define a path; a definition is in Section 1.5.

We begin this report by presenting the essential mathematical background required to comprehend the subject matter. The rest of this chapter is divided into three primary parts: We commence by defining all the mathematical concepts needed, and we then introduce the Signature, which is later used to prove the interesting properties of CDEs.

1.4 Definitions

A useful mathematical concept used in this project is the Tensor Product. To give a proper definition, we start with the following Lemma,

Lemma 1 *The Universal Property of the Tensor Product*

For any vector space U and any bilinear map $f : V \times W \rightarrow U$, there exists a unique linear map $g : V \otimes W \rightarrow U$ such that for all $v \in V$ and $w \in W$:

$$g(v \otimes w) = f(v, w). \quad (1.3)$$

Accordingly, we can define the Tensor Product as follows.

Definition 1 *Tensor Product [2]*

The tensor product $V \otimes W$ of two vector spaces V and W is a new, unique vector space (up to isomorphism) that possesses the universal property.

In simpler terms, the tensor product $V \otimes W$ is defined by the characteristic that any bilinear map from the Cartesian product of V and W to another vector space U can be uniquely extended to a linear map from $V \otimes W$ to U .

As an illustrative example, consider $V = \mathbb{R}^2$ and $W = \mathbb{R}^3$ with basis $B_V = \{e_1, e_2\}$ and $B_W = \{j_1, j_2, j_3\}$. In this case, the tensor product is the space of 2×3 matrices ($\mathbb{R}^{2 \times 3}$).

We also define a notation for the **tensor powers**, which will be useful in some proofs later on:

$$V^{\otimes n} = \underbrace{V \otimes \dots \otimes V}_{n-1 \text{ times}}. \quad (1.4)$$

Next, we will define Banach spaces since these will be where our data lies. To do so, we start by defining a Cauchy Sequence,

Definition 2 *Cauchy Sequence*

A Cauchy sequence is a sequence of elements in the vector space where, as the Sequence progresses, the gap or distance between consecutive elements continuously diminishes and approaches zero.

Accordingly, we define a Banach Space as follows,

Definition 3 *Banach Space*

A Banach space is a normed vector space that is complete concerning the metric induced by its norm. Completeness implies that every Cauchy sequence within the space converges to a limit that also exists within the space.

Definition 4 $\text{Lip}(\alpha, F, W)$, and the $\|\cdot\|_{\Gamma_\alpha}$ norm

Let V and W be two Banach spaces, $k \in \mathbb{N}$, $\alpha \in (k, k+1]$ a real number, F a close subset of V and $f^0 : F \rightarrow W$. We define $f^i : F \rightarrow \mathcal{L}(V^{\otimes i}, W)$ for $i = 1, \dots, k$, with $\mathcal{L}(V^{\otimes i}, W)$ denoting the space of linear mappings $V^{\otimes i} \rightarrow W$. The collection $(f^0, \dots, f^k) \in \text{Lip}(\alpha, F, W)$ if there exists an M such that the following is true [39]:

$$\left\{ \begin{array}{l} \text{For all } i: \\ \sup_{x \in F} \|f^i(x)\|_{\mathcal{L}(V^{\otimes i}, W)} \leq M, \\ \text{For all } i, \text{ for all } x, y \in F, \text{ and for every } v \in V^{\otimes i}: \\ \left\| f^i(y)(v) - \sum_{l=0}^{k-i} \frac{f^{i+l}(x)(v \otimes (y-x)^{\otimes l})}{l!} \right\|_{\mathcal{L}(V^{\otimes i}, W)} \leq M|x-y|_V^{\alpha-i}. \end{array} \right. \quad (1.5)$$

We also define the corresponding norm [19]:

$$\|f\|_{\Gamma_\alpha} = \sup \left\{ \frac{|f(s) - f(t)|}{(s-t)^\alpha}, s \neq t \right\}. \quad (1.6)$$

Even though the bounds in (1.5) seem complicated, they usually provide simple meanings. For example, when $\alpha = 1$, saying a function is in $\text{Lip}(\alpha, F, W)$ is equivalent to saying it is bounded and Lipschitz.

Definition 5 *Finite p -variation and the $\|\cdot\|_{I,p}$ norm*

For a given Banach space V and $p \geq 1 \in \mathbb{R}$, we say that the path $X : [t_0, t_n] \rightarrow V$ has finite p -variation if [26]:

$$\omega_{I,p}(X) = \sup \sum_P |X_{t_{i+1}} - X_{t_i}|^p < +\infty, \quad (1.7)$$

where the supremum is over all partitions P of $I = [t_0, t_n]$.

We use this to define the following norm [24]:

$$\|\cdot\|_{I,p} = |\omega_{I,p}(\cdot)|^{1/p}. \quad (1.8)$$

Using this definition, we now prove a Lemma that will be useful later in proofs.

Lemma 2 *Relationship between $\|\cdot\|_{\Gamma_\alpha}$ and $\|\cdot\|_{I,p}$.*

Let W be a Banach space, F be a closed subset of a Banach space V , and g be a vector field with a collection $(g^0, \dots, g^k) \in \text{Lip}(\alpha, F, W)$. Let $X : F \rightarrow V$ be a path with finite p -variation. We can write the following bound:

$$\|g(X)\|_{F,p/\alpha} \leq \|g\|_{\Gamma_\alpha} (\|X\|_{F,p})^\alpha \quad (1.9)$$

Proof: By definition, for t_{i-1} , and $t_i \in F$:

$$\|g\|_{\Gamma_\alpha} \geq \frac{|g(X_{t_i}) - g(X_{t_{i-1}})|}{|X_{t_i} - X_{t_{i-1}}|^\alpha}. \quad (1.10)$$

Hence, by rearranging the equation and raising it to the power p/α , we can write the following,

$$|g(X_{t_i}) - g(X_{t_{i-1}})|^{p/\alpha} \leq \|g\|_{\Gamma_\alpha}^{p/\alpha} |X_{t_i} - X_{t_{i-1}}|^p. \quad (1.11)$$

Taking the sum over all the partitions P of F , taking the supremum over these, and raising to the power α/p , we can write:

$$\left(\sup_{t_i \in F} \sum_P |g(X_{t_i}) - g(X_{t_{i-1}})|^{p/\alpha} \right)^{\alpha/p} \leq \|g\|_{\Gamma_\alpha} \left(\sup_{t_i \in F} \sum_P |X_{t_i} - X_{t_{i-1}}|^p \right)^{\alpha/p}, \quad (1.12)$$

which concludes the proof by the definition of $\|\cdot\|_{F,p}$.

Next, we intend to define the shuffle product. To do so, we define a (k, m) -shuffle.

Definition 6 *(k, m) -shuffle*

A permutation σ of a set $\{1, \dots, k+m\}$ is a (k, m) -shuffle if $\sigma^{-1}(1) < \dots < \sigma^{-1}(k)$ and $\sigma^{-1}(k+1) < \dots < \sigma^{-1}(k+m)$. We define $Sh_{k,m}$ as the collection of all (k, m) -shuffles.

Definition 7 *Shuffle product* Consider two multi-indexes $I = i_1, \dots, i_k$ and $J = j_1, \dots, j_m$ with all their entries between 1 and a certain constant d . We define the multi-index:

$$\{r_1, \dots, r_{k+m}\} = \{i_1, \dots, i_k, j_1, \dots, j_m\}. \quad (1.13)$$

Hence, the shuffle product of the two multi-indexes I and J , denoted by $I \sqcup J$ is defined as follows:

$$I \sqcup J = \{ (r_{\sigma(r_1)}, \dots, r_{\sigma(r_{k+m})}) \mid \sigma \in Sh_{k,m} \} \quad (1.14)$$

To avoid any confusion, we compute the shuffle product for an example in Section 1.5 (Signature Property 4).

Definition 8 *Combinatorial Increments and Integrals*

On an interval $[t_0, t_n]$, let \mathcal{D} be a dissection such that $t_0 = r_0 < \dots < r_k = t_n$. For a certain function \tilde{f} over $[t_0, t_n]$, we define the following increments:

$$\begin{cases} \Delta r_i^+ = r_{i+1} - r_i, & \Delta r_i^- = r_i - r_{i-1}, \\ \Delta \tilde{f}_i^+ = \tilde{f}(r_{i+1}) - \tilde{f}(r_i), & \Delta \tilde{f}_i^- = \tilde{f}(r_i) - \tilde{f}(r_{i-1}). \end{cases} \quad (1.15)$$

We also define the approximate integrals over the partitions of \mathcal{D} as follows,

$$\int_{\mathcal{D}} \tilde{f} dr := \sum_{i=0}^{k-1} \tilde{f}(r_i) \Delta r_i^+ \approx \int_{t_0}^{t_n} \tilde{f} dr. \quad (1.16)$$

The truncated approximate integral is then defined as follows,

$$\int_{\mathcal{D}^i} \tilde{f} dr = \int_{\mathcal{D}} \tilde{f} dr - \tilde{f}(r_i) \Delta r_i^+. \quad (1.17)$$

In other words, the approximate integrals are the sum of rectangles $[r_i, r_{i+1}] \times [\tilde{f}(r_{i-1}), \tilde{f}(r_i)]$, as is illustrated for $k = 3$ in figure 1.1. We hope that through the figure, one can understand why the sum is only until $k - 1$ and note that the approximation is a sum of rectangles beneath the curve. The truncated approximate integrals are defined similarly with the i^{th} rectangle removed.

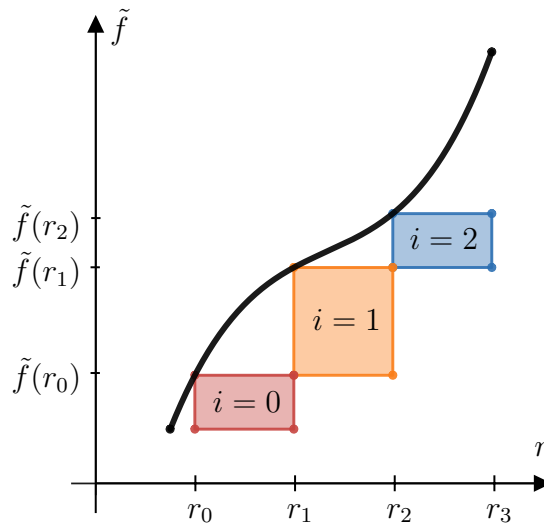


Figure 1.1: Approximation of the integral using partitions of a dissection.

1.5 Path Signature

In this section, we define the concept of a path signature and explore its essential properties. The significance of the signature in the report is its connection to controlled differential equations, which will be established in Section 1.6. When presenting results, we also show how these can be useful to ameliorate the performance of Random Forests if used as feature maps. To start, we provide a formal definition of a path.

Definition 9 *A path*

Let $m \in \mathbb{N}$, and $t_0 < t_n$ both real numbers. A path X in the space \mathbb{R}^m is a continuous mapping from an interval $[t_0, t_n] \in \mathbb{R}$ to \mathbb{R}^m , denoted as $X : [t_0, t_n] \rightarrow \mathbb{R}^m$.

Throughout this report, we refer to a path as smooth when its derivatives of every order are well-defined.

1.5.1 Introduction to Path Integrals

To introduce the concept of the path signature, we will first review the idea of a path integral, as it forms the basis of the signature. Consider two paths $Z_1 : [t_0, t_n] \rightarrow \mathbb{R}$ and $Z_2 : [t_0, t_n] \rightarrow \mathbb{R}$. The Riemann-Stieltjes integral [9], assuming that Z_2 is differentiable, allows us to integrate one path against another in the following manner:

$$\int_{t_0}^{t_n} Z_1(t) dZ_2(t) = \int_{t_0}^{t_n} Z_1 \frac{dZ_2}{dt} dt, \quad (1.18)$$

since the path Z_2 depends only on time. We will define different orders of path integrals as special cases of (1.18). For a general path $X : [t_0, t_n] \rightarrow \mathbb{R}^m$, we can define the coordinate paths (X_1, \dots, X_m) , where each component is a path $X_i : [t_0, t_n] \rightarrow \mathbb{R}$. The first-order iterated integral is given by:

$$S_{t_0, t_*}^{i_1}(X) = \int_{t_0 \leq u_1 \leq t_*} dX_{i_1}(u_1) = \int_{X_{i_1}(t_0)}^{X_{i_1}(t_*)} dX_{i_1}, \quad (1.19)$$

The superscript $i_1 \in [1, m]$ specifies which coordinate path we are integrating against, and the subscripts refer to the start and end times for integration. The variable u_1 serves as the integration variable. Notably, (1.19) is equivalent to (1.18) with $Z_1 = 1$ and $Z_2 = X_{i_1}$, as the constant 1 is also a path.

Building on this, the second-order iterated integral is defined as follows:

$$\begin{aligned} S_{t_0, t_*}^{i_1, i_2}(X) &= \int_{t_0 \leq u_2 \leq t_*} S_{t_0, t}^{i_1}(X) dX_{i_2}(u_2) \\ &= \int_{t_0 \leq u_1 \leq u_2 \leq t_*} dX_{i_1}(u_1) dX_{i_2}(u_2) = \int_{X_{i_2}(t_0)}^{X_{i_2}(t_*)} \int_{X_{i_1}(t_0)}^{X_{i_1}(t)} dX_{i_1} dX_{i_2}. \end{aligned} \quad (1.20)$$

Here, two superscripts are required, and they can be the same. The first superscript specifies which first-order path integral should be the integrand, and the second designates the coordinate path to integrate against. The integration limits are $t_0 \leq u_1 \leq u_2$ and $t_0 \leq u_2 \leq t_*$. Only the outer integral is evaluated at t_* , with the inner one expressed as a function of time. Again, the second-order iterated integral is itself a path.

Extending this pattern, the k -th order iterated integral is defined as:

$$\begin{aligned}
S_{t_0, t_*}^{i_1, i_2, \dots, i_k}(X) &= \int_{t_0 \leq s \leq t_*} S_{t_0, t}^{i_1, i_2, \dots, i_{k-1}}(X) dX_{i_k}(u_k) \\
&= \int_{t_0 \leq u_1 \leq \dots \leq u_k \leq t_*} dX_{i_1}(u_1) dX_{i_2}(u_2) \cdots dX_{i_k}(u_k) \\
&= \int_{X_{i_k}(t_0)}^{X_{i_k}(t_*)} \int_{X_{i_{k-1}}(t_0)}^{X_{i_{k-1}}(t)} \cdots \int_{X_{i_1}(t_0)}^{X_{i_1}(t)} dX_{i_1} dX_{i_2} \cdots dX_{i_k}.
\end{aligned} \tag{1.21}$$

Finally, we define the path signature as the collection of path integrals of all orders (infinitely many values) evaluated at the final time t_n :

$$S_{t_0, t_n}(X) = \left[1, S_{t_0, t_n}^1, \dots, S_{t_0, t_n}^m, S_{t_0, t_n}^{1,1}, S_{t_0, t_n}^{1,2}, \dots, S_{t_0, t_n}^{k,k}, S_{t_0, t_n}^{1,1,1}, \dots \right]. \tag{1.22}$$

Note that when there is no confusion over X , we use $S_{t_0, t_n}^{i_1, \dots, i_k}$ instead of $S_{t_0, t_n}^{i_1, \dots, i_k}(X)$ as the entries of the vector.

In practice, as we will justify later in this section, we include only the path integrals up to a particular order (say N), resulting in what we refer to as the truncated signature of depth N . We use the notation $S_{t_0, t_n}^{(N)}(\cdot)$.

1.5.2 Geometric interpretation

To give the reader an idea of what the signature means, we include a simple example that explains the geometrical meaning of the terms of a truncated signature of depth 2. Let's consider the following dataset,

$$\tilde{X} = \begin{bmatrix} 2 & 1 \\ 4 & 4 \\ 6 & 5 \\ 8 & 9 \\ 10 & 10 \end{bmatrix} \implies X_1 = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 9 \\ 10 \end{bmatrix}. \tag{1.23}$$

The purpose is to discuss the terms of truncated signature of depth 2:

$$S_{1,5}^{(2)}(\tilde{X}) = [1, S_{1,5}^1, S_{1,5}^2, S_{1,5}^{1,2}, S_{1,5}^{2,1}, S_{1,5}^{1,1}, S_{1,5}^{2,2}]. \tag{1.24}$$

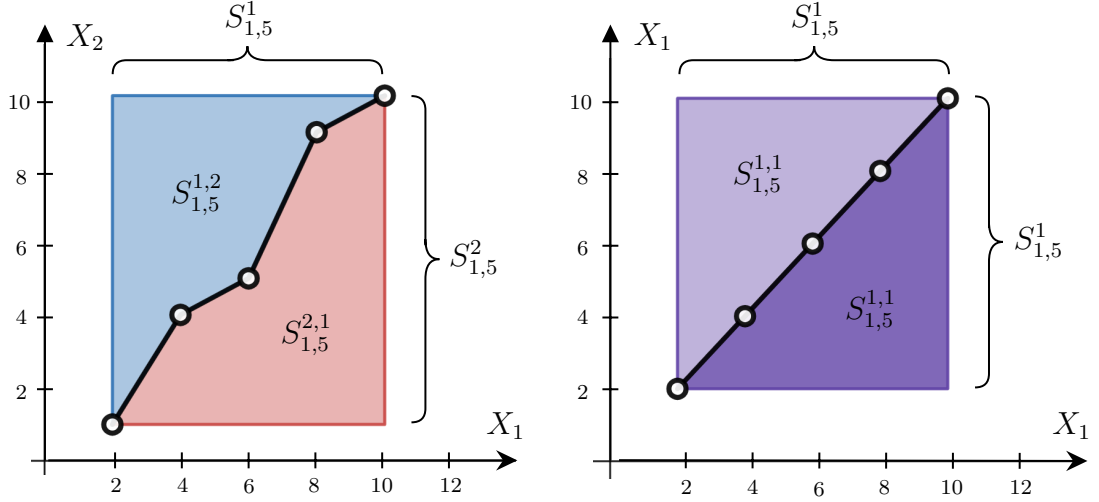


Figure 1.2: Graph showing the interpretation of Signature terms geometrically. Note that the path is drawn on an $X_2 - X_1$ axis to the left and $X_1 - X_1$ axis to the right.

Note that \tilde{X} is discrete, and it does not make sense to talk about the path signature if there is no path! Hence, we transform our discrete data into a path by simply interpolating linearly in this case (a detailed discussion about interpolation is in Section 2.2.2), as can be seen in black to the left of figure 1.2. Let X be the resulting path.

First-order terms of the signature can be computed as follows.

$$S_{1,5}^1(X) = \int_{X_1(1)}^{X_1(5)} dX_1 = X_1(5) - X_1(1) = 10 - 2 = 8, \quad S_{1,5}^2(X) = X_2(5) - X_2(1) = 9. \quad (1.25)$$

One can see that the first-order path integral against a certain coordinate path will always be the difference between the path's first and last values. In other words, the 1st order terms of the signature are the length from one point to another (the braces to the left of figure 1.2).

We compute two other terms of the signature as follows:

$$\begin{cases} S_{1,5}^{1,2}(X) = \int_1^{10} \int_2^{X_1(t)} dX_1 dX_2 = 33, \\ S_{1,5}^{2,1}(X) = \int_2^{10} \int_1^{X_2(t)} dX_2 dX_1 = 39. \end{cases} \quad (1.26)$$

Looking closely at the double integrals, we note that $S_{1,5}^{1,2}(X)$, for instance, is the area under the curve, bounded by the smallest and largest values of X_2 and X_1 respectively (i.e., the red area to the left of figure 1.2). Similarly, $S_{1,5}^{2,1}(X)$ represents the blue area.

We proceed now to find the last two terms:

$$\begin{cases} S_{1,5}^{1,1}(X) = \int_2^{10} \int_2^{X_1(t)} dX_1 dX_1 = 32, \\ S_{1,5}^{2,2}(X) = \int_1^{10} \int_1^{X_2(t)} dX_2 dX_2 = 40.5. \end{cases} \quad (1.27)$$

These terms are remarkably similar to the previous ones, with one exception: we use the same coordinate path twice inside the integral. Accordingly, we can again find their values by looking at the area under/above the curve, but now, when the path is drawn on an $X_1 - X_1$ or an $X_2 - X_2$ axis, respectively. For example, the path was drawn against $X_1 - X_1$ to the right of figure 1.2. In this case, either purple areas represent the path signature term $S_{1,5}^{1,1}(X)$. Another thing to note here is that we can relate $S_{1,5}^{1,1}(X)$ and $S_{1,5}^1(X)$ as follows:

$$S_{1,5}^{1,1}(X) = \frac{(S_{1,5}^1(X))^2}{2} = 32. \quad (1.28)$$

The same can be said about $S_{1,5}^{1,2}(X)$ and $S_{1,5}^2(X)$. We will show later in this section that it is always possible to represent higher-order terms in the signature using its previous terms by what we call the shuffle product identity (Section 1.5.3, Signature Property 4).

1.5.3 Signature Properties

In the upcoming subsection, we outline the critical properties of the signature. We aim to highlight the most crucial aspects and provide proof for the most essential properties in our case. We do not provide all the proofs since our primary interest is CDEs, not the signature.

Signature Property 1 *Uniqueness [17]:*

Let V be a Banach space, and $X : [t_0, t_n] \rightarrow V$ be a path with bounded variation, the signature $S_{t_0, t_n}(X)$ determines X up to a tree-like equivalence.

Elaboration: The signature can define the path except where it retraces itself. One can augment the path with the parametrization vector (let's say time) to solve this issue. If the vector is monotonically increasing/decreasing, this ensures that the path will never retrace itself, hence making the signatures entirely define the path. The proof was first elaborated by [6] for a class of piecewise smooth polynomials, and it was extended for paths with bounded variations by [17].

Signature Property 2 *Invariance under parametrization [25]:*

Let V be a Banach space and $X : [t_0, t_n] \rightarrow V$ be a path of bounded variation. For any map $\delta : [t_0, t_n] \rightarrow [t_0, t_n]$ that is continuously differentiable, surjective, and increasing, $S_{t_0, t_n}(X) \equiv S_{t_0, t_n}(X \circ \delta)$.

Elaboration: The result is interesting yet expected, especially since, as detailed in the previous section, the signature elements have geometrical meanings. The signature does not need to know anything about parametrization. It stores information about the order in which the events happen but not the exact time at which they happened. This could be useful in many cases (e.g., for cancer patients, the order of the events, such as which treatment happened earlier, could contain some relevant information for training). However, time can always be augmented to the data as a channel if needed.

Signature Property 3 *Exponential Decay [26]:*

Let U be a continuous increasing function, V be a Banach space, and $X : [t_0, t_n] \rightarrow V$ a path with its coordinate paths $(X_i)_{i=1}^\infty$, such that $\|X_i\|_{[t_0, t_n], p} \leq (U(t_n) - U(t_0))^{1/p}$, the n^{th} iterated integral can be bounded in function of p as follows,

$$\left\| \int_{t_0 \leq u_1 \leq \dots \leq u_k \leq t_n} dX_{i_1}(u_1) \cdots dX_{i_k}(u_k) \right\|_{[t_0, t_n], p} \leq \left(1 + \zeta \left(\frac{2}{p} \right) \right)^{k-1} \left(\frac{U^k(T)}{k!} \right)^{1/p}, \quad (1.29)$$

ζ being the zeta function defined as: $\zeta(x) = \sum_{n=1}^\infty \frac{1}{n^x}$.

Proof: The proof of this property depends on the boundedness of path integrals. Hence, it is divided into three parts. First, we prove a simple Combinatorial Lemma, which we use to prove that path integrals are bounded. We then use the bound to conclude the proof by induction.

Part 1: Combinatorial Lemma

Following the steps of [24], we want to prove that within the approximate integrals defined in (1.16), there exists at least one rectangle that is small enough. In other words, we want to prove the following lemma.

Lemma 3 *Combinatorial Lemma*

Let D be a dissection as defined previously, f be a convex and increasing function, then there exists an i such that the following holds:

$$\Delta \tilde{f}_i^- \Delta r_i^+ \leq \frac{1}{4(k-1)^2} \left(\int_{t_0}^{t_n} |\tilde{f}'(s)|^{0.5} ds \right)^2. \quad (1.30)$$

Proof: We start by noting that:

$$\begin{aligned} \min_i \Delta \tilde{f}_i^- \Delta r_i^+ &= \min_i \left(\frac{1}{(k-1)^2} \left((k-1) \left(\Delta \tilde{f}_i^- \right)^{0.5} \left(\Delta r_i^+ \right)^{0.5} \right)^2 \right) \\ &\leq \left(\frac{1}{k-1} \sum_{i=1}^{k-1} \left(\Delta \tilde{f}_i^- \right)^{0.5} \left(\Delta r_i^+ \right)^{0.5} \right)^2. \end{aligned} \quad (1.31)$$

Additionally, we note the following for a general convex function $\tilde{f}(x)$:

$$\begin{cases} \delta \tilde{f} \cong \left| \frac{d\tilde{f}}{dx} \right|^{0.5} \delta x \left| \frac{d\tilde{f}}{dx} \right|^{0.5}, \\ \tilde{f}(x_2) - \tilde{f}(x_1) = \int_{x_1}^{x_2} \tilde{f}'(s) ds \leq \left(\tilde{f}'(x_2) \right)^{0.5} \int_{x_1}^{x_2} \left(\tilde{f}'(s) \right)^{0.5} ds, \end{cases} \quad (1.32)$$

where for the second equation, we used the fact that for an increasing positive g , one has $\int_t^s g^2 \leq g(s) \int_t^s g$.

We can then re-write (1.31) by using the second line of (1.32) for $\Delta \tilde{f}_i^-$ and a combination of both lines of (1.32) for Δr_i^+ to get the following,

$$\begin{aligned} \min_i \Delta \tilde{f}_i^- \Delta r_i^+ &\leq \\ &\frac{1}{(k-1)^2} \left(\sum_{i=1}^{k-1} \left(\left(\int_{r_{j-1}}^{r_j} \left(\tilde{f}'(s) \right)^{0.5} ds \right) \left(\int_{r_j}^{r_{j+1}} \left(\tilde{f}'(s) \right)^{0.5} ds \right) \right)^{0.5} \right)^2. \end{aligned} \quad (1.33)$$

Finally, using the fact that $\forall a, b, (|a| - |b|)^2 \geq 0 \implies |a| + |b| \geq 2|ab|^{0.5}$, we can simplify (1.33) to (1.30) which proves the lemma proposed.

Special case: When $\tilde{f}(s) = s^N$, the bound in (1.30) becomes:

$$\begin{aligned} \min_i \Delta \tilde{f}_i^- \Delta r_i^+ &\leq \frac{1}{(k-1)^2} \frac{N}{(N+1)^2} \left(t_n^{(N+1)/2} - t_0^{(N+1)/2} \right)^2 \\ &\leq \frac{1}{(k-1)^2} \frac{N}{N+1} \left(\frac{t_n^{N+1} - t_0^{N+1}}{N+1} \right) \\ &= \frac{1}{(k-1)^2} \frac{N}{N+1} \int_{t_0}^{t_n} s^N ds, \end{aligned} \quad (1.34)$$

where the second equation uses the fact that $\forall b > a > 0, b^2 - a^2 > (b-a)^2$.

Part 2: Bounding Path Integrals

In this subsection, we establish the following result:

Lemma 4 *Boundedness of Path Integrals*

Let $U(t)$ be a monotonically increasing function, and $f(t)$ be a continuous path with $\|\tilde{f}\|_{[\eta, \xi], p} \leq C_0|U(\eta) - U(\xi)|^{1/p}$ for all $\eta < \xi \in \mathbb{R}$, C_0 is a positive constant. Consider another path g with $g(0) = 0$ and $\|g\|_{[\eta, \xi], p} \leq C_1|U^N(\eta) - U^N(\xi)|^{1/p}$, where N is an integer and C_1 is a positive number. Define $h(t) = \int_0^t g \, d\tilde{f}$. Then, we have the bound:

$$\|h\|_{[\eta, \xi], p} \leq C_1 C_0 (1 + \zeta(2/p)) \left(\frac{U^{N+1}(\eta) - U^{N+1}(\xi)}{N+1} \right)^{1/p}, \quad (1.35)$$

Proof: The proof draws inspiration from Young's approach [45] and employs the previously established Combinatorial Lemma. We follow the steps outlined in [24]. For a given interval $[t_0, t_n]$, let \mathcal{D} and \mathcal{D}^i be dissections as defined in Section 1.4 (Definition 8), and let $\int_{\mathcal{D}}(\cdot)$ and $\int_{\mathcal{D}^i}(\cdot)$ represent their corresponding approximated integrals. We choose \mathcal{D} in such a way that the approximate integral $\int_{\mathcal{D}} g \, d\tilde{f}$ provides a suitably accurate approximation, i.e.,

$$\left| \int_{\mathcal{D}} g \, d\tilde{f} - \int_{t_0}^{t_n} g \, d\tilde{f} \right| < \epsilon, \quad (1.36)$$

where ϵ is a small positive value. From the definition in (1.17), we know that:

$$\begin{aligned} \int_{\mathcal{D}} g \, d\tilde{f} - \int_{\mathcal{D}^i} g \, d\tilde{f} &= \Delta g_i^- \Delta \tilde{f}_i^+ \\ &\leq (C_1(\Delta U_i^-)^N C_0 \Delta U_i^+)^{1/p} \\ &\leq \left(\frac{C_1 C_0}{(k-1)^2} \left(\frac{U^{N+1}(t_n) - U^{N+1}(t_0)}{N+1} \right) \right)^{1/p}, \end{aligned} \quad (1.37)$$

where the second line utilizes the boundedness assumed in the Lemma, and the third line invokes the Combinatorial Lemma established in Part 1.

Through the elegant reasoning of [45], while capitalizing on the fact that $g(0) = 0$, the process of removing multiple rectangles from the approximate integral yields the following bound:

$$\left| \int_{\mathcal{D}} g \, d\tilde{f} \right| \leq \sum_{k=2}^{\infty} \left(\frac{C_1 C_0}{(k-1)^2} \left(\frac{U^{N+1}(t_n) - U^{N+1}(t_0)}{N+1} \right) \right)^{1/p} + |g(t_n)(\tilde{f}(t_n) - \tilde{f}(t_0))| + \epsilon, \quad (1.38)$$

Notably, the term in absolute value is dominated by $C_1 C_0 U(t_n)^N (U(t_n) - U(t_0))$. Consequently, it can be inferred that:

$$|g(t_n)(\tilde{f}(t_n) - \tilde{f}(t_0))| \leq \frac{C_1 C_0 (U^{N+1}(t_n) - U^{N+1}(t_0))}{N+1}, \quad (1.39)$$

given that U is monotonically increasing and $N > 0$. Therefore, as ϵ approaches zero, the following inequality holds:

$$\left| \int_{\mathcal{D}} g d\tilde{f} \right| \leq C_1 C_0 \left(1 + \zeta \left(\frac{2}{p} \right) \right) \left(\frac{U^{N+1}(t_n) - U^{N+1}(t_0)}{N+1} \right)^{1/p}. \quad (1.40)$$

Ultimately, the utilization of this inequality on sub-intervals confirms the bound in (1.35), which concludes the second part of the proof.

Part 3: Proof by induction

Base case: $k = 2$

When $k = 2$, the iterated integral is simplified to the following,

$$\int_{t_0 \leq u_1 \leq u_2 \leq t_n} dX_{i_1}(u_1) dX_{i_2}(u_2) = \int_{X_{i_2}(t_0)}^{X_{i_2}(t_n)} \int_{X_{i_1}(t_0)}^{X_{i_1}(t)} dX_{i_1} dX_{i_2}. \quad (1.41)$$

We note that the result is a path integral just like the one defined in Lemma 4 with,

$$\begin{cases} g = \int_{X_{i_1}(t_0)}^{X_{i_1}(t)} dX_{i_1} = X_{i_1}(t) - X_{i_1}(t_0), \\ \tilde{f} = X_{i_2}(t). \end{cases} \quad (1.42)$$

Accordingly, setting $C_1 = 1$ and $N = 1$, we can write the following bound,

$$\left\| \int_{t_0 \leq u_1 \leq u_2 \leq t_n} dX_{i_1}(u_1) dX_{i_2}(u_2) \right\|_{[t_0, t_n], p} \leq \left(1 + \zeta \left(\frac{2}{p} \right) \right) \left(\frac{U^2(t_n) - U^2(t_0)}{2} \right)^{1/p}. \quad (1.43)$$

The rest of the proof follows by assuming that the result is valid for, say, $k = \kappa \in \mathbb{N}$, and proving that the result remains valid for $k = \kappa + 1$ using Lemma 4 with g set to be the iterated integral from the previous iteration, \tilde{f} set to be the κ^{th} coordinate path, C_1 set to be a constant found of the prior case, $C_0 = 1$, and $N = \kappa - 1$.

Signature Property 4 *Shuffle Product Identity [8]:*

Let V be a Banach space, X be a path $X : [t_0, t_n] \rightarrow V$, and I, J be the multi-indexes $\{i_1, \dots, i_k\}$ and $\{j_1, \dots, j_m\}$ respectively, with $i_1, \dots, i_k, j_1, \dots, j_m \in \{1, \dots, d\}$, we can write the following,

$$S_{t_0, t_n}^I(X) S_{t_0, t_n}^J(X) = \sum_{K \in I \sqcup J} S_{t_0, t_n}^K(X). \quad (1.44)$$

Explanation: In other words, the property, first derived by [33], claims that a multiplication of lower-level signature terms can be written as a linear combination of higher-order signature terms. This is extremely useful when computing the signature. It is also the basis of the signature's universal nonlinearity (proved later in Signature Property 5). Since the shuffle product can be confusing theoretically, we give two simple examples.

$$\begin{cases} S_{t_0, t_n}^1(X) S_{t_0, t_n}^1(X) = (S_{t_0, t_n}^1(X))^2 = 2S_{t_0, t_n}^1(X) \\ S_{t_0, t_n}^{1,2}(X) S_{t_0, t_n}^1(X) = 2S_{t_0, t_n}^{1,1,2}(X) + S_{t_0, t_n}^{1,2,1}(X) \end{cases} \quad (1.45)$$

We showed the first example in (1.28) geometrically. We detail briefly how the second one was computed:

$$\begin{cases} I = \{1, 2\}, \quad k = 2, & J = \{1\}, \quad m = 1, \\ Sh_{k,m} = \{\sigma^1, \sigma^2, \sigma^3\}, & \text{where:} \\ \sigma^1(1) = 1, \quad \sigma^1(2) = 2, \quad \sigma^1(3) = 1, \\ \sigma^2(1) = 1, \quad \sigma^2(2) = 1, \quad \sigma^2(3) = 2, \\ \sigma^3(1) = 1, \quad \sigma^3(2) = 1, \quad \sigma^3(3) = 2, \\ \text{Note that for all } l \in [1, 2, 3]: \quad (\sigma^l)^{-1}(1) < (\sigma^l)^{-1}(2). \end{cases} \quad (1.46)$$

Signature Property 5 *Universal Nonlinearity* [23]:

Let V be a Banach space, $\mathcal{V}^p([t_0, t_n], V)$ be the space of paths from $[t_0, t_n]$ to V with finite p -variation, and $\hat{\mathcal{K}} \subset \mathcal{V}^p([t_0, t_n], V)$ be compact such that any $\hat{X} \in \hat{\mathcal{K}}$ has at least one monotone coordinate. For any $\epsilon > 0$ and any continuous $F : \hat{\mathcal{K}} \rightarrow \mathbb{R}$, there exists a level of truncation $N \in \mathbb{N}$ and $w \in \mathbb{R}^N$, such that:

$$\left| F(X) - \langle w, S^N(\hat{X}) \rangle \right| \leq \epsilon, \quad (1.47)$$

$\langle \cdot, \cdot \rangle$ being the Euclidean scalar product on \mathbb{R}^N .

Proof: The statement above is equivalent to claiming that the linear forms of the signature are dense in $\mathcal{C}(\hat{\mathcal{K}}, \mathbb{R})$. Let $S(\mathcal{V}^p([t_0, t_n], V))$ denote the space of the signatures of the paths $X \in \mathcal{V}^p([t_0, t_n], V)$. The proof will be limited to the case $p < 2$ since we never encounter rougher paths and will be based on the Stone-Weierstrass theorem.

Theorem 1 *Stone-Weierstrass theorem* [41]

Let (X, d) be a compact metric space, and let A be a subalgebra of the space of continuous real-valued functions $\mathcal{C}(X, \mathbb{R})$. Then A is dense in $\mathcal{C}(X, \mathbb{R})$ if and only if A contains the constant functions and it is rich enough to separate the points.

Proof: The proof consists of first showing that the linear forms of the signature are dense in $\mathcal{C}(\mathcal{K}_S, V)$, \mathcal{K}_S being a compact subset of $S(\mathcal{V}^p([t_0, t_n], V))$. We then extend the density to $\mathcal{C}(\hat{\mathcal{K}}, \mathbb{R})$.

Let A be the algebra of real-valued functions formed by linear forms on \mathcal{K}_S . By definition, the algebra will have the shuffle product defined in Signature Property 4. [26] proves that the algebra A with such a shuffle product is a subalgebra of $\mathcal{C}(\mathcal{K}_S, \mathbb{R})$, it contains the constant functions and separates the points. Accordingly, we conclude by the Stone-Weierstrass theorem that A is dense in $\mathcal{C}(\mathcal{K}_S, \mathbb{R})$, hence concluding the first part of the proof.

Note the uniqueness property of the signature previously mentioned (Signature Property 1). Since the path \hat{X} has at least one monotonic channel, there are no tree-like equivalent paths, and $X_{t_0} = 0$ [3]. Accordingly, for any function $e : \hat{\mathcal{K}} \rightarrow \mathbb{R}$, there exists a compact set $\mathcal{K}_* \subset \mathcal{V}_0^p([t_0, t_n], \mathbb{R} \times V)$ and a function $e_* : S(\mathcal{K}_*) \rightarrow \mathbb{R}$ such that:

$$e(\hat{X}) = e_*(S(\hat{X})). \quad (1.48)$$

Since from the previous part of the proof, we know that the linear forms of $S(\hat{X})$ are dense in $\mathcal{C}(S(\mathcal{K}_*), \mathbb{R})$, we deduce from (1.48) that the linear forms of $S(\hat{X})$ are dense in $\mathcal{C}(\hat{\mathcal{K}}, \mathbb{R})$ which concludes the proof.

1.6 Controlled Differential Equations (CDEs)

With all the necessary mathematical background previously established, we are now ready to introduce CDEs and prove some of their relevant properties.

Let V and W be two Banach spaces, X and Y be two path $[t_0, t_n] \rightarrow V$ and $[t_0, t_n] \rightarrow W$ respectively, and $f(\cdot)\nu$ be a linear map from $\nu \in V$ to vector fields on W . Y solves the CDE:

$$dY = f(Y)dX, \quad (1.49)$$

if for all times $t \in [t_0, t_n]$,

$$Y = Y_0 + \int_{t_0}^t f(Y)dX, \quad (1.50)$$

where $Y_0 \in W$ is known as the initial condition of the system, X as the control, and Y as the system response.

1.6.1 Picard's iterations

Throughout this section, we will heavily rely on Picard's iterations to illustrate an example or prove results about CDEs. Accordingly, we clearly define what they are in this subsection.

For a given continuous path $X \in \mathbb{R}^m$ and its corresponding coordinate paths $X = (X_1, \dots, X_m)$. Let $(f^i)_{i=1}^m$ be differentiable vector fields on a Banach space V . To solve:

$$dY^0 = \sum_{i=1}^m f^i(Y^0) dX_i, \quad (1.51)$$

we define Picard's iteration [18] as follows,

$$M_f Y^{n-1} := Y^n = Y^0 + \int_{t_0}^{t_n} f(Y^{n-1}) dX. \quad (1.52)$$

Y^n is a solution of (1.51) if and only if it is a fixed point of M_f , in other words, if $Y^n = Y^{n-1}$.

1.6.2 Relationship to the Signature

The reader could wonder why we defined the signature and its relevance to CDEs. In this section, we illustrate the significance of this signature and how it is related to CDEs.

Linear Vector fields:

We first illustrate the relationship between the signature and CDEs using a simplified example of (1.50).

Let $\mathcal{L}(\mathbb{R}^c, \mathbb{R}^b)$ denote the vector space of the linear maps from \mathbb{R}^c to \mathbb{R}^b . We define $L(\cdot)$ as the linear map $\mathbb{R}^b \rightarrow \mathcal{L}(\mathbb{R}^c, \mathbb{R}^b)$, which can also be represented by $\mathbb{R}^c \rightarrow \mathcal{L}(\mathbb{R}^b, \mathbb{R}^b)$, $\mathcal{L}(\mathbb{R}^b, \mathbb{R}^b)$ being the vector space of $b \times b$ matrices. We also define the path $X : [t_0, t_n] \rightarrow \mathbb{R}^c$. Then $Y : [t_0, t_n] \rightarrow \mathbb{R}^b$ solves:

$$Y = Y_0 + \int_{t_0}^t L(Y) dX, \quad Y_0 \in \mathbb{R}^b. \quad (1.53)$$

To find the solution of (1.53) using Picard's theorem, the first step is to define the initial iteration value, which we will choose $Y^0 = Y_0$. The remaining iterates, keeping

in mind that L is linear, are as follows [8]:

$$\left\{ \begin{array}{l} Y^0(t) = Y_0, \\ Y^1 = Y_0 + \int_{t_0}^t L(Y^0(s)) dX = \left(\int_{t_0}^t dL(X_s) + I_b \right) Y_0, \\ Y^2 = Y_0 + \int_{t_0}^t L(Y^1(s)) dX = \left(\int_{t_0}^t \int_{t_0}^s dL(X_u) dL(X_s) + \int_{t_0}^t dL(X_s) + I_b \right) Y_0, \\ \vdots \\ Y^k = Y_0 + \int_{t_0}^t L(Y^{k-1}(s)) dX = \left(\sum_1^k \int_{t_0 \leq t_1 \leq \dots \leq t_k \leq t} dL(X_{t_1}) dL(X_{t_k}) + I_b \right) Y_0, \\ \vdots \end{array} \right. \quad (1.54)$$

where I_b is the identity matrix in $\mathcal{L}(\mathbb{R}^b, \mathbb{R}^b)$. A closer look at the expression inside the sum clearly shows that the solution of the CDE can be represented as a linear combination of the k^{th} order truncated signature terms and the initial condition Y_0 . Next, we establish a more generic claim.

Generic Vector fields:

In this section, we explain why the following lemma makes sense.

Lemma 5 *For any path $\hat{X} : [t_0, t_n] \rightarrow V$ with at least one monotone channel, there exists an f , and a large enough N such that the CDE:*

$$Y = Y_0 + \int_{t_0}^t f(Y) d\hat{X}, \quad Y_0 = (1, 0, \dots, 0), \quad (1.55)$$

has a solution $Y_{t_n} \approx S_{[t_0, t_n]}^{(N)}(\hat{X})$.

Sketch Proof: Proving the Lemma follows trivially from the definition of the signature. As previously mentioned, the first term of the signature is equal to 1, and the latter terms are simply the iterated integrals. Accordingly, the signature can be defined as the solution of:

$$Y = Y_0 + \int_{t_0}^t Y \otimes d\hat{X}, \quad Y_0 = (1, 0, \dots, 0). \quad (1.56)$$

We note that by the definition of the tensor product, $Y \otimes d\hat{X}$ can be written as $f(Y) d\hat{X}$ where f depends on the signature transform³. Finally, the exponential decay of the signature allows us to deduce the Lemma.

³The detailed relationship can be found in the Appendix of [21].

1.6.3 CDE Properties

This subsection shows why CDEs are attractive when learning trends in certain datasets. We start by proving the existence and uniqueness of the solution of a CDE for controls with bounded variation with $p < 2$ and then show why CDEs can be considered universal approximators under light conditions on the control.

CDE Property 1 *Existence of a solution*

If $X \in \mathbb{R}^k$ is continuous and has finite p -variation with $p < 2$, and the collection $(f^0, \dots, f^k) \in \text{Lip}(\gamma)$ with $\gamma > p - 1$, then there exists solutions to (1.51) with finite p' variations for any $p' \in (p, 2)$, which all lie within a ball of a finite radius R .

Proof: The proof follows by showing that a solution to Picard's iterations exists. Young's [45] basic inequality can be used to bound the n^{th} iterate of Picard's iteration as follows

$$\begin{aligned} \|Y^{n+1}\|_{[t_0, t_n], p} &\leq \left(\zeta \left(\frac{1+\gamma}{p} \right) \|f(Y^n)\|_{[t_0, t_n], p/\gamma} + \|f(Y)\|_\infty \right) \|X\|_{[t_0, t_n], p} \\ &= \left(\zeta \left(\frac{1+\gamma}{p} \right) \|f\|_{\Gamma_\gamma} \|Y^n\|_{[t_0, t_n], p}^\gamma + \|f(Y)\|_\infty \right) \|X\|_{[t_0, t_n], p} \\ &= (A \|Y^n\|_{[t_0, t_n], p}^\alpha + B) \|X\|_{[t_0, t_n], p}, \end{aligned} \quad (1.57)$$

where we defined $A = \zeta \left(\frac{1+\alpha}{p} \right) \|f\|_{\Gamma_\alpha}$ and $B = \|f(Y)\|_\infty$ to work with simpler expressions. Following the steps of [24], we choose t and R such that $(A/R^{1-\alpha} + B/R)\|X\|_{[t_0, t_n], p} < 1$. This choice along with the bound in (1.57) show that for $\|Y^n\|_{[t_0, t_n], p} > R$, we will always have $\|Y^{n+1}\|_{[t_0, t_n], p} < \|Y^n\|_{[t_0, t_n], p}$, hence proving that all fixed points will lie within a ball⁴ of radius R .

In addition, since M_f maps the closed ball for p -variation paths to itself, it also maps its compact closure in p' -variation paths to itself. By the claim made by [24] that the set is convex, we can use Tychonoff's fixed point theorem [42] to show that M_f must have a fixed point, hence proving that solutions exist under the specified conditions.

What goes wrong with $p \geq 2$:

Fortunately, all the paths encountered in this project were bounded with variations $p < 2$. On the other hand, rougher paths, such as Brownian motion, only have bounded variations when $p > 2$. In this case, the integral $\int f dX$ is not defined in the Young sense, requiring a more sophisticated approach. These are part of the rough

⁴The expression of R can be found in [24].

path theory, which will not be detailed in the report. Interested readers can refer to [13], [15], [14], [16], and [27].

CDE Property 2 *Uniqueness of the solution*

If $X \in \mathbb{R}^k$ is continuous and has finite p -variation with $p < 2$, and the collection $(f^0, \dots, f^k) \in \text{Lip}(\gamma)$ with $p < \gamma < 2$, then there exist a unique solution to (1.51) and convergence is very rapid.

Proof: The solution mainly relies on finding a way to write the difference between two Picard's iterations as an integral. The following lemma guarantees this.

Lemma 6 *Expression of differences [40]:*

For a function f with a collection $(f^0, \dots, f^k) \in \text{Lip}(\gamma)$, $\gamma > 1$, there exists a function g with a collection $(g^0, \dots, g^k) \in \text{Lip}(\gamma - 1)$ such that:

$$f(x_2) - f(x_1) = \sum_{i=1}^k (x_2 - x_1)^i g^i(x_2, x_1). \quad (1.58)$$

It can also be shown that the norms of f and g are equivalent if $\|f\|_\infty$ is finite.

Accordingly, by defining the difference between two consecutive Picard's iterates $D_n = Y^n - Y^{n-1}$, we can write the following,

$$D^{n+1} = \int_{t_0}^t D^n g(Y^n, Y^{n-1}) dX. \quad (1.59)$$

We will show that D^n decays to zero rapidly, which we will use to imply that the solution Y^n is unique. We start by using the fact that X has bounded variations, and we limit the choice of our initial guess Y^0 so we can write

$$\|X\|_{[t_0, t_n], p} \leq |U(t_n) - U(t_0)|^{1/p}, \quad \|Y^0\|_{[t_0, t_n], p} \leq |U(t_n) - U(t_0)|^{1/p}, \quad (1.60)$$

where U is a continuous and increasing function. Based on the previous inequality (1.57), we have the following bound,

$$\|Y^{n+1}\|_{[t_0, t_n], p} \leq \left(\zeta \left(\frac{2}{p} \right) \|\nabla f(Y^n)\|_\infty \|Y^n\|_{[t_0, t_n], p} + \|f(Y)\|_\infty \right) (U(t_n) - U(t_0))^{1/p} \quad (1.61)$$

Based on the work of [24], we impose a slightly stronger assumption on U :

$$|U(t_n) - U(t_0)|^{1/p} < \frac{1 - \epsilon}{\zeta(2/p) \|\nabla f\|_\infty}, \quad (1.62)$$

where $\epsilon < 1$. We will proceed now to bound the norm of Y^{n+1} independently of other iterates. Formally, we will prove the following Lemma.

Lemma 7 *Boundedness of Picard's iterates*

When the assumptions in Lemma (2) hold along with the assumption in (1.62), we can bound the n^{th} Picard's iterate independently of the previous iterates by:

$$\begin{aligned} \|Y^n\|_{[t_0, t_n], p} &\leq \left((1 - \epsilon)^n + \|f\|_\infty \sum_{k=1}^{n-1} (1 - \epsilon)^k \right) (U(t_n) - U(t_0))^{1/p} \\ &\leq (1 + 2\|f\|_\infty)(U(t_n) - U(t_0))^{1/p}. \end{aligned} \quad (1.63)$$

Proof: The first line can easily be shown using induction. The base case trivially follows from (1.60), (1.61) and (1.62). We then assume that the result holds for Y^{n-1} and plug it in (1.61) to show that it still holds for Y^n .

For the second line, since $0 < \epsilon < 1$, we can bound $(1 - \epsilon)^n \leq 1$. On the other hand, the sum is a geometric series that can be expressed as:

$$\sum_{k=1}^{n-1} (1 - \epsilon)^k = 1 - \left(\frac{1}{2}\right)^{n-1} \leq 2, \quad (1.64)$$

which concludes the proof.

Since by Lemma 6, $\|g\|_{\Gamma_{(\gamma-1)}}$ is finite, using Lemma 2, we can also bound the p -norm of g by just changing the constant in Lemma 7. In other words,

$$\|g(Y^n, Y^{n-1})\|_{[t_0, t_n], p/(\gamma-1)} \leq C_2 (U(t_n) - U(t_0))^{(\gamma-1)/p}, \quad (1.65)$$

where C_2 is a constant. Now, let $H^n = \int_0^{t_n} g(Y^n, Y^{n-1}) dX$. Using Young's inequality and the previously proved result in (1.65), we can write the following bound,

$$\begin{aligned} \|H^n(\cdot)\|_{[t_0, t_n], p} &\leq \left(\zeta \left(\frac{\beta}{p} \right) \|g\|_{[t_0, t_n], p/(\gamma-1)} + \|g\|_\infty \right) \|X\|_{[t_0, t_n], p} \\ &\leq \left(\zeta \left(\frac{\beta}{p} \right) C_2 (U(t_n) - U(t_0))^{(\gamma-1)/p} + \|g\|_\infty \right) (U(t_n) - U(t_0))^{1/p}. \end{aligned} \quad (1.66)$$

This insinuates that there exists an increasing function $\mathcal{U}(\cdot)$ and a value T , for which for all $t_0, t_n < T$:

$$\|H^n(\cdot)\|_{[t_0, t_n], p} \leq \mathcal{U}(t_n)(U(t_n) - U(t_0))^{1/p}. \quad (1.67)$$

We are finally ready to prove that Picard's iterations converge to a unique solution. Note that (1.59) can be rewritten as:

$$D^{n+1} = \int_{t_0}^t D^n dH^n. \quad (1.68)$$

Since by definition $\|D^1\|_{[t_0, t_n], p} \leq L|U(t_n) - U(t_0)|^{1/p}$, the bound in (1.67) allows us to use Signature Property 3 to bound the iterates of (1.68) as follows,

$$\|D^N\|_{[t_0, t_n], p} \leq \left(1 + \zeta \left(\frac{2}{p}\right)\right)^{N-1} \mathcal{U}(T)^N L^N \left| \frac{U^N(t_n) - U^N(t_0)}{N!} \right|^{1/p}. \quad (1.69)$$

The fact that $N!$ dominates in the equation proves that Picard's iterations will converge quickly. This implies the uniqueness of the CDE's solution since for a solution Y^* , we can write:

$$Y^{n+1} - Y^* = \int_{t_0}^t (Y^n - Y^*) g(Y^n, Y^*) dX. \quad (1.70)$$

Accordingly, over any interval \mathcal{I} , we have:

$$\|Y^n - Y^*\|_{\mathcal{I}, p} = \mathcal{O} \left(\left(\frac{C_3^N}{N!} \right)^{1/p} \right), \quad (1.71)$$

This confirms that Y^* is unique and that Picard's iteration converges rapidly to Y^* .

CDE Property 3 *Universal Approximation using CDEs [21]*

For $t_0, t_n \in \mathbb{R}$ with $t_0 < t_n$, and $a, b \in \mathbb{N}$, let

$$\begin{cases} F = \{f : \mathbb{R}^c \rightarrow \mathbb{R}^{c \times (a+1)} \mid f \text{ is continuous}\}, \\ Q = \{q : \mathbb{R}^{a+1} \rightarrow \mathbb{R}^c \mid q \text{ is continuous}\}. \end{cases} \quad (1.72)$$

Further, let $X : [t_0, t_n] \rightarrow \mathbb{R}$ be a path, and Y be the unique solution of the following CDE,

$$Y_t = Y_0 + \int_{t_0}^t f(Y) dX, \quad Y_0 = q(X_{t_0}), \quad (1.73)$$

where $q \in Q$, and $f \in F$. Let $K \subseteq \mathcal{V}^2([t_0, t_n] \rightarrow \mathbb{R}^a)$ be compact. Then for any continuous function $G : \mathcal{K} \rightarrow \mathbb{R}^b$ and any $\epsilon > 0$, there exists a $w \in \mathbb{R}^b$ such that:

$$|G(X) - \langle w, Y_{t_n} \rangle| \leq \epsilon, \quad (1.74)$$

or in other words, Y_{t_n} is dense in $\mathcal{C}(K, \mathbb{R}^b)$.

Proof: For a certain path $X : [t_0, t_n] \rightarrow \mathbb{R}^a$, we define $\hat{X} : [t_0, t_n] \rightarrow \mathbb{R}^{a+1}$ to be the path with at least one monotone channel (e.g. time as a channel), and $X^* : [t_0 - 1, t_n] \rightarrow \mathbb{R}^a$ the extension of the path with lines as follows,

$$X^* = \begin{cases} (t - t_0 + 1)X(t_0), & t \in [t_0 - 1, t_0], \\ X(t), & t \in [t_0, t_n]. \end{cases} \quad (1.75)$$

We also define $\mathcal{K}^* = \{X^* | X \in \mathcal{K}\}$.

From Lemma 5, we know that there exists an f , and a large enough N for all \hat{X}^* such that the solution Y of:

$$Z = Z_0 + \int_{t_0}^t f(Z) d\hat{X}^*, \quad Z_0 = (1, 0, \dots, 0), \quad (1.76)$$

is equal to $S_{t_0, t_n}^{(N)}(\hat{X}^*)$. We now define $q \in Q$ as $q(X(t_0)) = Z(t_0)$, which is well defined since on the first interval in (1.75), X^* only depends on $X(t_0)$. We now consider the following CDE,

$$Y = Y_0 + \int_{t_0}^t f(Y) d\hat{X}^*, \quad Y_0 = q(X(t_0)). \quad (1.77)$$

By the uniqueness of the solution proved in CDE Property 1, over the interval $[t_0, t_n]$, one should have $Z(t) = Y(t) = S_{t_0, t_n}^{(N)}(\hat{X}^*)$. In particular, this holds for $t = t_n$.

Now note that from Signature Property 5 (universality), we know that $S_{t_0, t_n}^{(N)}(\hat{X}^*)$ should be ϵ -close to any function $\beta \in \mathcal{C}(\mathcal{K}^*, \mathbb{R}^c)$.

On the other hand, the map $X \rightarrow X^*$ is a homeomorphism, which means that for every $\alpha \in \mathcal{C}(\mathcal{K}, \mathbb{R}^c)$, there exists a $\beta \in \mathcal{C}(\mathcal{K}^*, \mathbb{R}^c)$, such that $\alpha(X) = \beta(X^*)$, which concludes the proof. To make the logic behind the proof simpler to understand, the proof is illustrated with a diagram below.

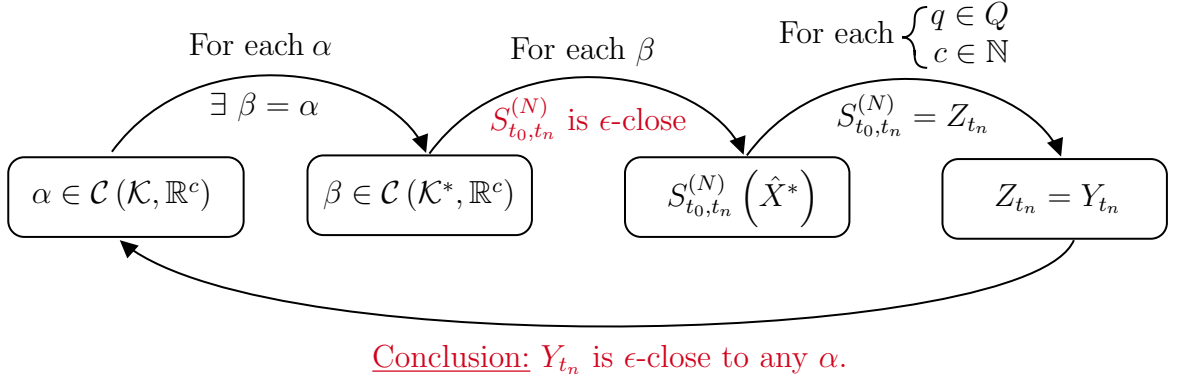


Figure 1.3: Diagram illustrating the logic behind the proof of universality for CDEs.

Chapter 2

Methodology

2.1 Multilayer Perceptron

The first method introduced is a Multilayer Perceptron (MLP), a neural network with fully connected nodes. Since we assume no previous knowledge of neural networks, we start with a brief review while explaining all the aspects implemented in the project.

2.1.1 What is a Neural Network?

A Neural Network can be thought of as a non-linear regression. We use Neural Networks to find a function that fits a set of data points non-linearly, hoping that the introduced non-linearity will be enough to come as close as possible to the data. An MLP consists of **layers**: The first/last layers are the input data and the output prediction, respectively. Any layer in between is known as a **hidden layer**. An example of an MLP with only two hidden layers can be seen in figure 2.1. For a sample data $V_0 \in \mathbb{R}^{s \times n}$ and output $\hat{V} \in \mathbb{R}^{m_2 \times n}$ (s being the size of the data, n the number of observations, and m_2 the dimension of the required output), V_0 is the input layer. The parameters of the Neural Network are $W^{[1]} \in \mathbb{R}^{m_1 \times s}$, $b^{[1]} \in \mathbb{R}^{m_1 \times n}$, $W^{[2]} \in \mathbb{R}^{m_2 \times m_1}$, $b^{[2]} \in \mathbb{R}^{m_2 \times n}$, where m_1 and m_2 are the size of the first and second hidden layers respectively (for example 4 for both in figure 2.1). The last hidden layer will generally always have the required output size (otherwise, the output shape will differ from the expected output). On the other hand, the second dimension of the first-layer matrix ($W^{[1]}$) will always be equal to the size of the data (otherwise, the matrix multiplication in the first hidden layer won't make sense). The functions g_1 and g_2 are known as **activation functions**, which add the non-linearity to the

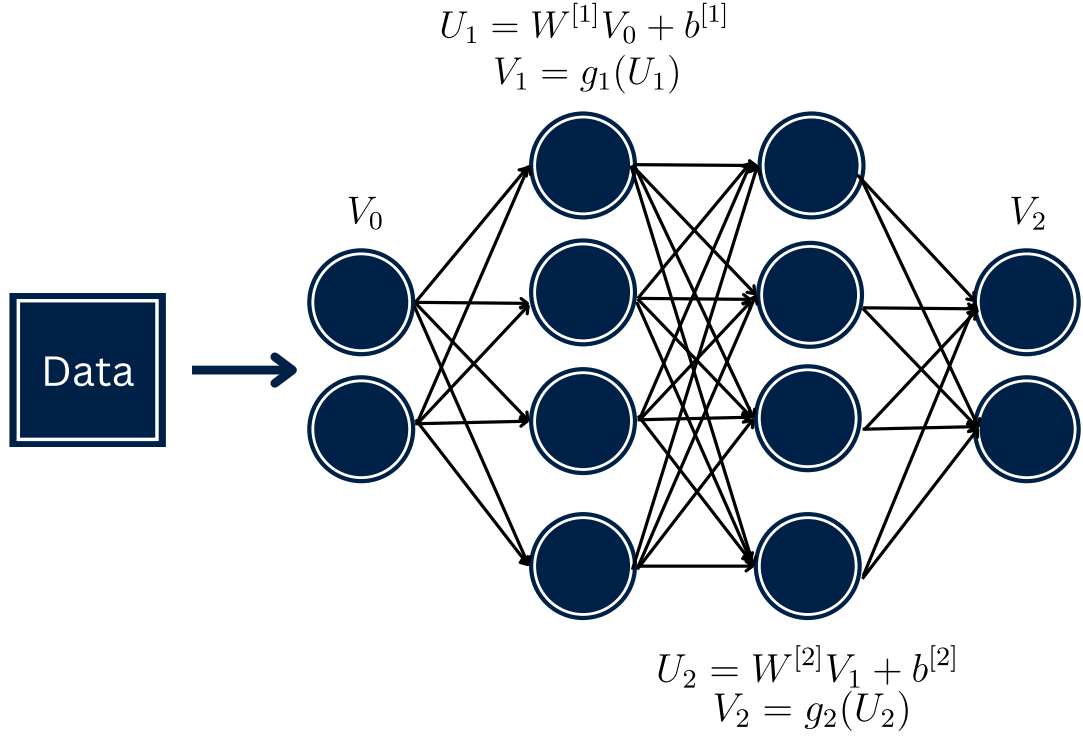


Figure 2.1: Example of a Multilayer Perceptron with only two hidden layers of depth four.

system. These are typically chosen to be the Sigmoid, hyperbolic tangent, or Rectifier. Finally, the Neural Network, by the steps explained in the following sections, will fine-tune the entries of the matrices and vectors to find a prediction $V_2 \in \mathbb{R}^{m_2 \times n}$, as close as possible to \hat{V} . Throughout the report, we will use a more general notation V_{out} for the output of the Neural Network.

2.1.2 Data pre-processing

Before explaining Neural Networks further, it is essential to mention that, in practice, not the entire dataset is used for training. Since Neural Networks are very powerful, we can quickly end up with a model that memorizes the data given and has not learned anything. Hence, even though the larger the dataset, the better the results, we need some part of the data to ensure that the Neural Network is training correctly. Accordingly, V_0 is chosen as a subset of the data labeled **Train dataset**. On the other hand, the rest of the data, labeled **Test dataset**, is used to compute the accuracy and examine whether the Neural Network learned adequately. Usually, the majority of the data is consecrated for training (e.g., 85-15 percent for train/test datasets, respectively).

2.1.3 Forward propagation

The first step followed by a Neural Network is forward propagation, which consists, as the name insinuates, of moving forward through the Neural Network and finding a prediction. For this project, only affine transformations are considered. In other words, as can be seen on the top/bottom of figure 2.1, each layer i multiplies its input by a matrix $W^{[i]}$, add to it another matrix $b^{[i]}$ and applies to it an activation function $g^{[i]}$. Accordingly, each layer can be thought of as follows:

$$V_{t+1} = f_{\theta}(V_t, \theta_t), \quad (2.1)$$

where f_{θ} includes the transformations explained, the subscripts insinuate a certain hidden layer, and θ includes the parameters within a layer ($W^{[t]}$, $b^{[t]}$, and $g^{[t]}$). A specific type of Neural Network is Residual Neural Networks (ResNets), which use a modified version of (2.1) as follows:

$$V_{t+1} = V_t + f_{\theta}(V_t, \theta_t). \quad (2.2)$$

The idea is to approximate the residual (or the difference between the input and the output) using a Neural Network $f_{\theta} : \mathbb{R}^{m_t \times n \times |\theta|} \rightarrow \mathbb{R}^{m_{t+1} \times n}$, m_t and m_{t+1} being the size of hidden layers t and $t + 1$ respectively, and $|\theta|$ being the number of parameters in the Neural Network.

All in all, after L layers, the output of the Neural Network is:

$$V_L = V_{out}, \quad (\text{since we start with } V_0). \quad (2.3)$$

The following subsection explains how to use the obtained result to modify the parameters of the Neural Network for V_{out} to approach the expected output \hat{V} .

2.1.4 Backward propagation

The parameters should be modified depending on how far the computed output V_{out} is from \hat{V} . The difference between the expected/actual output is quantified by the **Loss function**. The loss function is a quantity that we want to minimize while training. An example of a loss function for binary prediction (i.e., $V_{out} \in \mathbb{R}^{1 \times n}$) is the binary cross entropy function as follows:

$$\mathcal{L}(y, \hat{y}) = \frac{-1}{n} \sum_{i=1}^n \hat{y}_i \log(y_i) + (1 - \hat{y}_i) \log(1 - y_i), \quad (2.4)$$

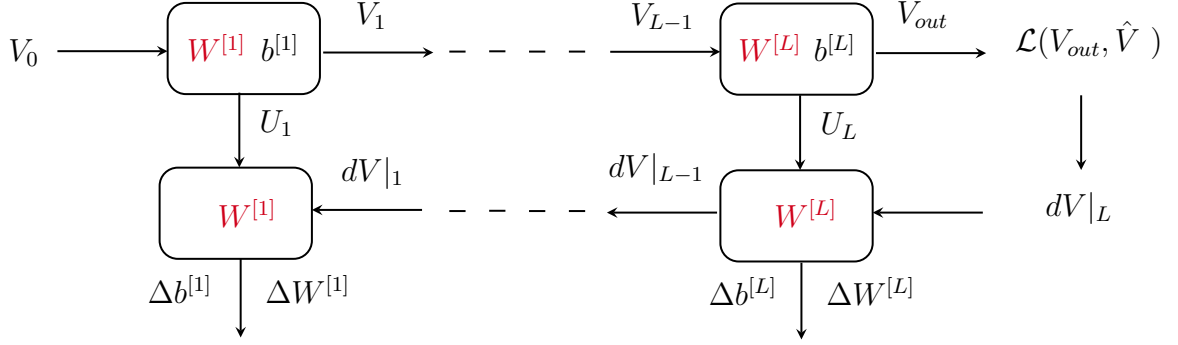


Figure 2.2: Diagram illustrating which parameters are needed in forward (top) and backward (bottom) propagations ($dV|_i$ is short for $\frac{\partial \mathcal{L}}{\partial V}|_i$).

where y_i and \hat{y}_i are the i^{th} columns (in this case scalars) of V_{out} and \hat{V} respectively. The loss function penalizes $y_i = 0$ when $\hat{y}_i = 1$ and vice versa. The loss function is not defined when the prediction y_i is exactly 0 or 1. In practice, the value of the Loss is just set to a large constant (e.g., -100 in PyTorch [31]).

After choosing a loss function, each parameter is updated as illustrated in figure 2.2, where $dV|_i$ is short for $\frac{\partial \mathcal{L}}{\partial V}|_i$. The figure's top part represents the forward path (affine, as explained in the previous section), and the bottom represents backpropagation. Backpropagation was re-introduced in 1986 by [35] and has been a very popular way to learn the parameters of a Neural Network. It's mainly based on Chain's rule of differentiation, where we use the derivative of the loss function with respect to the output to initiate the backpropagation process. As can be noticed in the figure, we find the required increments $\Delta W^{[i]}$ and $\Delta b^{[i]}$ from right to left (hence the name backpropagation). Another thing to note is that we need the matrices $W^{[i]}$ as well as the output of every hidden layer before applying an activation function (labeled U_i). Accordingly, we need to store all of these while performing forward propagation, which can be expensive on memory. We will show later how continuous models based on differential equations, such as NODEs and NCDEs, can overcome this issue. The last step would be to increment the old weights (i.e., matrices and vectors) as follows:

$$\begin{cases} W^{[i]} = W^{[i]} + \alpha \Delta W^{[i]}, \\ b^{[i]} = b^{[i]} + \alpha \Delta b^{[i]}, \end{cases} \quad (2.5)$$

where α is a hyperparameter, known as the **learning rate**, that we specify. The equations that relate the inputs to the outputs of the blocks can be found in [30]. We will not detail these here since backpropagation is covered in different libraries (e.g., PyTorch) by the use of optimizers such as Adam (Adaptive moment estimation) [22].

2.1.5 Batching

Even though the forward/backward process of a Neural Network looks simple enough, things become tougher in practice when the dataset is large. Since in this project, we work on large data (check section 3.1 for details on the dataset), we use batches of small sizes to reduce memory usage and computational overhead. The idea of a batch is to define the input of the Neural Network to be $\tilde{V}_0 \in \mathbb{R}^{s \times b}$ where b is the batch size and evaluate the entire dataset by going over it by batches. This means that parameters will be updated over batches (instead of the entire dataset), which helps avoid local minima and enhance stability.

All in all, we first divide the data into train/test datasets, we define the Neural Network (number of hidden layers, size of hidden layers, optimizer, activation functions, learning rate, and loss function), we give a batch of the data as an input, we step forward, find the loss, use its derivative to step backward, and update the parameters. Both forward and backward propagation processes over all the batches form an **epoch**. After a certain number of epochs (usually stopped when the Loss is below a certain threshold), we expect that the optimizer will enforce that the output V_{out} to get as close as possible to the expected output \hat{V} . Finally, the Neural Network is used to estimate the output of the test dataset, which is how the accuracy of the model is evaluated.

Next, we present an aspect of machine learning that guarantees that the used model is reliable: Cross Validation.

2.1.6 Cross Validation

Since the choice of the train/test datasets could lead to some homogeneous patterns, verifying that the data is well mixed by performing Cross-validation (CV) is a good practice. A CV consists of dividing the dataset into splits and training multiple times, each choosing another part of the data as the test set. Figure 2.3 illustrates the procedure for four splits. It is essential to note that the test sets in every case are not used for training! Cross-validation means performing different trainings that can not transfer information to each other (otherwise, the test data will be leaked to the neural network). Performing Cross-validation is common practice among machine learning models. Some of its main important aspects are listed here:

1. CV helps us make sure that the data is well mixed. In other words, no subset of the data has particular trends to learn compared to the others.

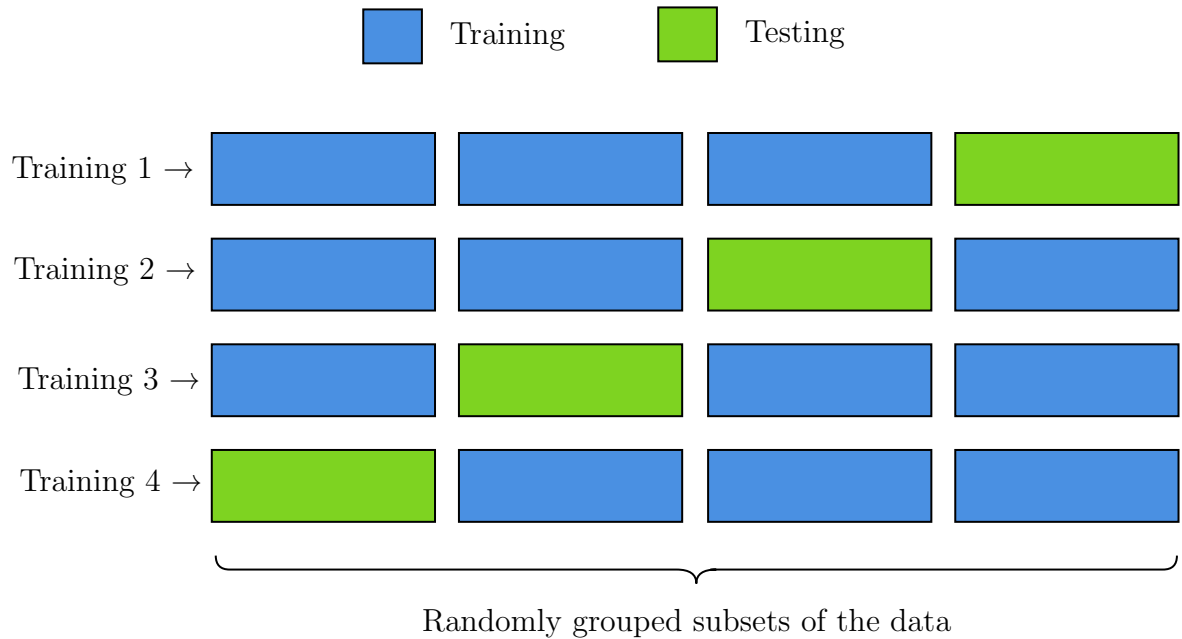


Figure 2.3: Diagram showing how a Cross-validation with four splits occurs. It is important to understand that the trainings do not depend on each other. However, they are compared to show how mixed the data is and how much the randomization affects the results.

2. CV allows us to know how much the randomization in the neural network will affect the results. Hence, we can compare methods and understand if some are actually doing better or if it is simply a matter of randomness.

The last aspect we present is a tool that limits overfitting: Dropout.

2.1.7 Dropout

Dropout consists of turning off some nodes (i.e., disregarding some of the outputs of a certain layer) **only during training**. The nodes are turned off/on according to a certain probability (e.g. 0.3). This ensures that the Neural Network is learning on a different part of the training data each time, which can guarantee to a certain extent that the model will not overfit. An important thing to note here is that dropout is turned off when using the Neural Network for evaluation (when used on the Test dataset, for instance) since, in these cases, we are interested in evaluating the Neural Network over all the data provided.

2.2 Continuous Models

MLPs are very effective on tabular data. However, we expect these not to perform as well when used to predict time series, especially irregular ones representing continuous phenomena. As detailed later, the dataset we will use illustrates changes in various blood markers. Accordingly, we expect a differential equation to do a much better job of approximating the process. The following section introduces machine learning models based on differential equations. We start by presenting Neural Ordinary Differential Equations (NODEs) and use them to introduce the method that is the core of the project, Neurally Controlled Differential Equations (NCDEs).

2.2.1 Neurally Ordinary Differential Equations

NODEs can be thought of as the continuous version of ResNets. First we can re-write (2.2) as follows,

$$\frac{V_{t+1} - V_t}{\Delta t} = \hat{f}_\theta(V_t, \theta_t) \quad \text{where: } \hat{f}_\theta = \frac{f_\theta}{\Delta t}. \quad (2.6)$$

We believe it is clear from the shape of (2.6) that with small enough timesteps, we can replace (2.6) with its continuous form, using a forward Euler approximation [1]:

$$\frac{\partial V}{\partial t} = \hat{f}_\theta(V_t, \theta_t). \quad (2.7)$$

Accordingly, an alternative way to find the output of a Neural Network of L hidden layers is as follows:

$$V(L) = V(0) + \int_0^L \hat{f}_\theta(V(t), \theta_t) dt. \quad (2.8)$$

The idea of a NODE is to use the Neural Network \hat{f}_θ to recover the differential equation in (2.7), which is then used to fit the data. Note that \hat{f}_θ does not depend explicitly on t . This can be easily adjusted by augmenting the vector $V(t)$ if required. Further, we still need to introduce non-linearity using activation functions. Therefore, NODEs can be described as follows:

$$V(0) = g^{[0]}(V_0) \quad \rightarrow \quad V(L) = V(0) + \int_0^L \hat{f}_\theta(V(t), \theta_t) dt \quad \rightarrow \quad V_L = g^{[L]}(V(L)), \quad (2.9)$$

where V_0 is a batch of the data (as defined earlier), V_L is the output of the NODE, and $g^{[0]}$ and $g^{[L]}$ are two activation functions.

Equation (2.9) can be solved using a differential equation solver such as `odeint` from

`torchdiffeq`. We will mention only three methods used to solve the ODEs since these were the only ones used in the project. For details about different solvers, refer to [32].

1. Euler’s method [1]: Starting from the initial guess $V^0 = V(0)$, we can find iteratively $V^i = V^{i-1} + h\hat{f}_\theta(V(t), \theta_t)$, where h is a fixed time step. The derivation trivially comes from a forward finite difference approximation.
2. Runge-Kutta-4 method (RK4) [36]: Again starting with an initial guess $V^0 = V(0)$, we now use more function evaluations (i.e., 4 of them) to estimate the solution by $V^n = V^{n-1} + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$ where:

$$\begin{cases} K_1 = h\hat{f}_\theta(V^n), & K_2 = h\hat{f}_\theta(V^n + 0.5K_1), \\ K_3 = h\hat{f}_\theta(V^n + 0.5K_2), & K_4 = h\hat{f}_\theta(V^n + K_3), \end{cases} \quad (2.10)$$

and again, h is a fixed time step.

3. Dopri5 method [12]: This method is also known as the RK45 method since it utilizes the RK5 method as a proxy to estimate the error of the RK4 method. What distinguishes this method is that it uses an adaptive time step. Whenever the error is not small enough, the step size is decreased. This allows the integration to achieve a global error less than a specified tolerance. However, we will show later that this method will not always be efficient since a smaller time step leads to higher computational times.

Before starting to discuss NCDEs, we note two main advantages of NODEs (these will eventually be advantages of NCDEs as well):

1. Memory efficiency: As mentioned in section 2.1.4, discrete Neural Networks require storing many parameters in the forward phase to backpropagate afterward. This results in a lot of memory consumption and additional numerical errors. For NODEs, since an ODE can be easily traversed in the opposite direction (i.e., by switching the time variable and the sign of (2.9)), we can save a lot of memory! A successful method based on this property is the adjoint sensitivity method [37], which allows us to train the Neural Network at memory cost that’s only in function of the depth of the Network. Even though this method might struggle with stability issues or feasibility (for instance, if the ODE is too complex to evaluate in either direction), the adjoint method is beneficial in practice (e.g., [38]).

2. Continuous model for time series: Since the time series we will encounter in this project represent a continuous process, the model is also expected to be continuous. In such cases, models based on differential equations show state-of-the-art performance compared to other discrete models (for instance, [10], [43], [44]).

2.2.2 Neurally Controlled Differential Equations

A limitation of NODEs is that once the parameters θ_t are learned, the learning process only depends on the initial conditions. A solution to this limitation is to include the data within the integral, allowing changes in the data to change the system response. Several methods have been tried: GRU-ODE, for instance, allowed \hat{f}_θ to depend on the dataset [11]. The main problem with this and other available methods is that the ODE was always interrupted to include the arriving data, which was inefficient. The model we will be using, NCDEs, proposes a better way to include the data by using it as the control of the differential equation. The main advantage of NCDEs is that they can capture trends in the data that other models miss. In this section, we present NCDEs along with some of its aspects.

Let D be a series of potentially irregularly observed data $D = ((t_0, x_0), \dots, (t_n, x_n))$ where $t_i \in \mathbb{R}$ contains the timestamps and $X_i \in \mathbb{R}^d$ contains the observations. Since we will be using the data as the control for the differential equation, the data better be continuous. Therefore, we start by interpolating¹ the dataset into $X : [t_0, t_n] \rightarrow \mathbb{R}^{d+1}$ where $X_{t_i} = (x_i, t_i)$. The NCDE model is then defined as follows:

$$V(L) = V(0) + \int_0^L F_\theta(V(t), \theta_t) dX_t \quad \rightarrow \quad V_L = g^{[L]}(V(L)), \quad (2.11)$$

where $V(0) = \xi_0(x_0, t_0)$ is any function of the initial observations², F_θ is a Neural Network with parameters θ and $g^{[L]}$ is a nonlinear activation function.

Using the dataset itself as the variable of integration has many benefits. First and most importantly, any change in the data will automatically change the system's behavior. We hope this will be enough to capture trends in the times series that other models miss. In addition, there is no need to interrupt the ODE anymore, which is a better/more efficient way of incorporating the incoming data. Lastly, it

¹Different interpolation techniques are presented later in this section.

²This is used to avoid translational invariance. Otherwise, the solver only depends on the derivative of the path (refer to (2.12)).

adds flexibility to the model: As discussed in the next chapter, stacking channels to the data (such as an intensity channel or the time) will ameliorate the results in certain circumstances. We now proceed to discuss an essential aspect of NCDES: interpolation.

Solving NCDEs relies heavily on NODEs. In fact, the integral in (2.11) is solved through the following:

$$V(L) = V(0) + \int_0^L F_\theta(V(t), \theta_t) \frac{dX_t}{dt} dt, \quad (2.12)$$

which can be propagated forward and backward exactly as a NODE, assuming that X_t is smooth enough for the derivative to exist. Theoretically, more stringent conditions should be enforced on X_t . First, since we might be using the adjoint method to backpropagate, the derivative of the integrand with respect to time is needed [38]. Hence, the path X_t should be at least twice differentiable. Furthermore, if an adaptive step size solver is used, having a discontinuous second derivative will slow down the solver significantly to resolve the jumps [21]. This draws to our attention that the interpolation technique should be carefully chosen. In this report, we will introduce three interpolation techniques: **Regular Linear interpolation**, **Rectilinear interpolation**, and **Hermite cubic spline with backward finite differences**. All three types of interpolation can be seen in figure 2.4 (next page). Note that in the figure, the index of the last observations ($k-2, k-1, k$) is not the same as the index of the last time points ($n-2, n-1, n$) since we allow for missing data ($n-k$ of them in this case). In addition, we assume in the figure that the observation at $t = t_1$ is missing. In the next paragraph, we will use this to explain how each interpolation technique interprets missing values (i.e., finds the value of x_*).

The purpose of interpolating is to find a path between two neighboring points (which we will name boundary points). This path should have two properties: First, satisfy the values of the observations at the boundary points, and second, be bounded over the time interval. There is certainly more than one path satisfying these properties, which is why different interpolation techniques exist.

All in all, the path that we need maps³ $P : [t_0, t_n] \rightarrow \mathbb{R}$. The i^{th} piece of P , denoted by X_i , maps $[t_i, t_{i+1}] \rightarrow \mathbb{R}$. Without loss of generality, we assume that we can map any two points (t_i, x_i) and (t_{i+1}, x_{i+1}) to $(0, x_i)$ and (Δ_i, x_{i+1}) , where $\Delta_i = t_{i+1} - t_i$. In the following paragraph, we explain how each interpolation technique finds X_i .

³This assumption is valid because, in practice, we interpolate each channel separately.

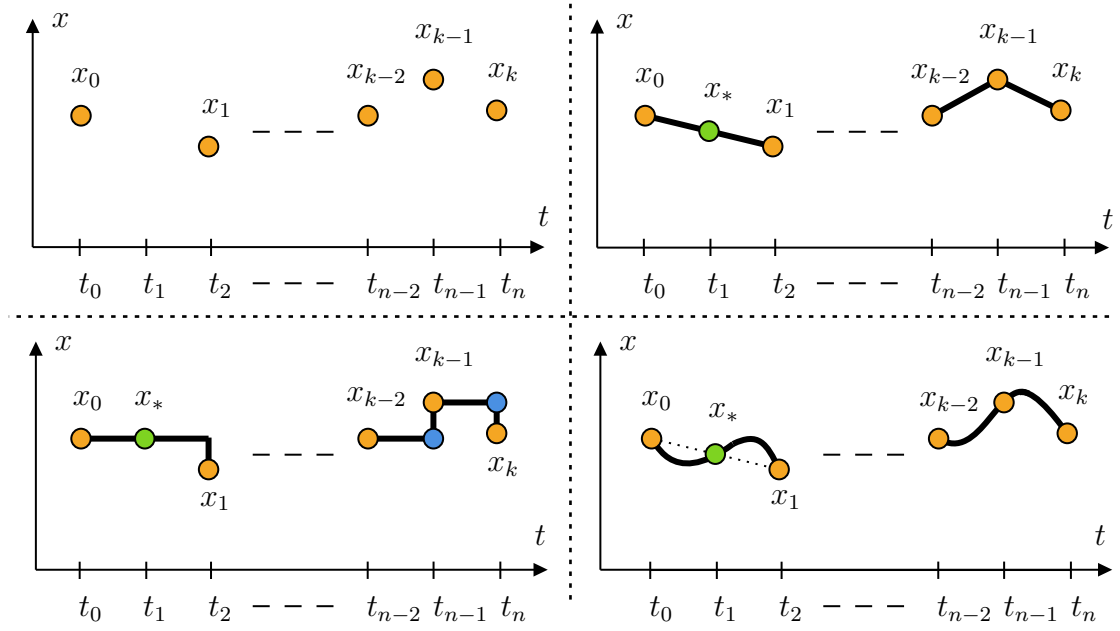


Figure 2.4: Figure showing sample discrete observations (top left) and their corresponding interpolations: Regular Linear (top right), Rectilinear (bottom left), Hermite cubic spline with backward finite differences (bottom right). We use the colors ● for observations, ● for hypothetical values, and ● for the approximation of a missing value.

Regular Linear Interpolation: This interpolation technique consists of drawing lines joining the data points (top right of figure 2.4). Accordingly, we can write:

$$X_i = \frac{x_{i+1} - x_i}{\Delta_i} t + x_i. \quad (2.13)$$

In this case, missing values are assumed to be on the line joining the points before and after. In other words, to find x_* in figure 2.4, we interpolate between x_0 , and x_2 (using (2.13)) and evaluate the resulting path at t_1 . An advantage of using this interpolation technique is that the path and its derivative are bounded. This is evident since the path will always be between the minimum and maximum values of the observations, and the differences between consecutive observations and consecutive times can bind the first derivative. Even though both the first and second derivatives risk being discontinuous, we will claim later in the report that this type of interpolation can be useful in practice, especially if the step size is constant.

Rectilinear Interpolation: This type of interpolation is a variation of linear interpolation, which stores different information about the data. The Regular Linear Interpolation technique assumes that the observations vary linearly with time, which

can be the case, especially for a short period of time in some applications. However, we might want to extract different information from the dataset. An alternative is Rectilinear interpolation, which assumes that an observation does not change until the next observation is reached. In other words:

$$\begin{cases} X_i = x_i, & \text{when: } t \leq \Delta_i, \\ X_i = x_{i+1}, & \text{when: } t = \Delta_i. \end{cases} \quad (2.14)$$

In this case, missing values are assumed to lie on the same line as the previous value. For instance, $x_* = x_0$ in figure 2.4 (bottom left). The advantage of this technique is that we keep track of both the values and the time at which the values change, represented by a discontinuity in the derivative. This information, such as in medical lab results, could be crucial for training. Furthermore, this interpolation strategy has the same boundedness properties as the previously mentioned one.

The problem with both mentioned interpolations is that they do not enforce the continuity of the second derivative. The next interpolation introduced solves this issue by using third-order polynomials.

Hermite cubic spline with backward finite differences: Using Hermite cubic polynomials allows us to enforce the derivatives at the boundary points, which will be approximated using backward finite differences. Hence, we can write the polynomial and the conditions to be enforced between each two points as follows:

$$\begin{cases} p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \\ p(0) = x_i, \quad p(\Delta_i) = x_{i+1}, \\ p'(0) = \frac{x_i - x_{i-1}}{\Delta_{i-1}}, \quad p'(\Delta_i) = \frac{x_{i+1} - x_i}{\Delta_i}, \end{cases} \quad (2.15)$$

where $(\cdot)'$ stands for a derivative in time. Replacing the conditions in the polynomial's expression would allow us to find the constants as follows:

$$\begin{cases} a_0 = x_i, \quad a_1 = \Delta_{i-1}^{-1}(x_i - x_{i-1}), \\ a_2 = 2\Delta_i^{-2}\Delta_{i-1}^{-1}(\Delta_{i-1}(x_{i+1} - x_i) - \Delta_i(x_i - x_{i-1})), \\ a_3 = \Delta_i^{-3}\Delta_{i-1}^{-1}(\Delta_i(x_i - x_{i-1}) - \Delta_{i-1}(x_{i+1} - x_i)). \end{cases} \quad (2.16)$$

The first equation in (2.15) along with the constants in (2.16) defines the path X_i . For missing values, we use regular linear interpolation to find the value at the missing point before using the cubic approximation. For instance, we can see that the green value is the same for the Regular Linear and Hermite Cubic approximations in figure

2.4. Yet, the path from one point to another is different.

Choosing this type of interpolation has multiple advantages. First, the second-order derivative is guaranteed to be continuous. Furthermore, computing the derivatives and finding their maxima shows that both the path and the derivative are bounded as follows [28]:

$$\begin{cases} \max |P| \leq c_1 \left(x_{max} + \frac{x_{max} \Delta_{max}}{\Delta_{min}} \right), \\ \max \left| \frac{dP}{dt} \right| \leq c_2 \frac{x_{max}}{\Delta_{min}}. \end{cases} \quad (2.17)$$

Hence, the interpolation is bounded and ensures the continuity of the second derivative, which is exactly what we need. It is simple to show using these properties, as done in the Appendix of [21] for example, that the resulting path is contained within a compact subset, hence proving that the universal approximation property of CDEs will hold for Neural CDEs (Section 1.6, CDE property 3). Even though this might not hold for Regular Linear or Rectilinear interpolations, we will show later in the results that they still work and could be quite useful in practice.

Lastly, we note that in all cases, having missing values at either the start or the end can be problematic. In this case, we impute these values with appropriate assumptions.

2.3 Ensemble Models

To evaluate the performance of NCDEs, we use MLPs as baselines. However, to claim that NCDEs are doing well, we also compare them to a state-of-the-art model: Random Forests [5].

2.3.1 Random Forests

This paragraph briefly discusses random forests to familiarize readers with the parameters chosen in the third chapter. Readers interested in a detailed description are referred to [4]. The algorithm followed in Random Forests can be described as follows:

1. Ensemble of Decision Trees: Random Forest creates an ensemble of decision trees during training. Each tree is trained on a different subset of the data, introducing randomness and diversity.

2. **Bootstrap Aggregating (Bagging):** It uses bootstrapping, where each tree is trained on a random sample of the training dataset with replacement. This helps prevent overfitting by ensuring each tree sees a slightly different dataset.
3. **Random Feature Selection:** At each split in a decision tree, only a random subset of features is considered. This further increases diversity among the trees and prevents them from being overly correlated.
4. **Voting or Averaging:** Finally, Random Forest combines the predictions of individual trees through majority voting.

Throughout the report, we use bootstrapping in addition to specifying three other parameters: The **Number of Estimators**, which is the number of decision trees within a Random Forest. The **Minimum Samples in Leaf Nodes** refers to the minimum number of samples required to split an internal node in a decision tree. This helps prevent nodes from being small and capture noise from the data. Finally, the **Minimum Samples in Leaf Nodes** refers to the minimum number of samples required to split an internal node in a decision tree. Any node that has less than this number won't be split further.

2.3.2 Signature as a feature map

All the properties mentioned and proved about the path signature make it ideal as a feature map. Mainly, it defines the path uniquely (or up to tree-like equivalence), is invariant under parametrization, and can be low dimensional (due to the exponential decay). Throughout this report, we propose to give as input to the Random Forest model the path signature of the data! We will refer to this model as Random Forests with Signature (RFS). We note that the signature keeps track of the order of the event, which we expect will significantly affect training, hence making Random Forests better predictors. However, since the size of the signature grows exponentially when the dimension of the data is high, implementing the signature as a feature map is not a trivial task. We detail how this was done in Chapter 3.

2.4 Metric for evaluation

In the remainder of the report, we will often want to compare different models based on their performance on binary predictions. Accordingly, we use this section to provide a reliable metric to decide whether a model performs well.

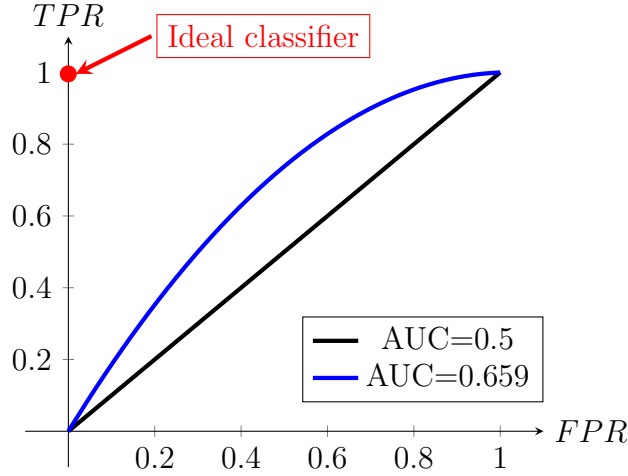


Figure 2.5: Example of a Receiver Operating Characteristic (ROC) curve.

Traditionally, binary predictions are evaluated by choosing a **threshold** (usually 0.5 for normalized positive data), according to which one decides the category of an instance. Then, either the percentage of correct predictions or the values within a confusion matrix⁴ would be the evaluation metric. Usually, the choice of the threshold plays a vital role in how accurate the binary prediction is, and the best one might be different from 0.5. Accordingly, we use the much more reliable metric, the receiver operating characteristic (ROC) curve [29]. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) for different thresholds. In other words, it summarizes the data given in confusion matrices over all thresholds.

An example of a ROC curve can be seen in figure 2.5. First, note that $FPR = TPR$, or the black line in figure 2.4, represents a random classifier. The perfect classifier would be at the red dot on the top left of the figure (i.e., a TPR of 1 and a FPR of 0). The bottom left and top right corners correspond to choosing all the data to be positive or negative, respectively. A good classifier will have a curve above the diagonal for all other points (except (0,0) and (1,1)). Throughout the report, we will quantify the curve by the Area Under the Curve (AUC). $AUC = 0.5$ refers to a random choice, and the higher the AUC, the better the classifier. Also, when using cross-validation, we will report our results by an average AUC (AVG AUC) (from all splits) and a standard deviation (STD AUC). AVG AUC will give us an idea of how good the classifier is, and STD AUC will give us an idea of the homogeneity of the data. The ROC curve is a reliable metric in this project since we will predict whether survival is above/below the median (i.e., the percentage of events is around 50%).

⁴Matrix containing: True Positives, False Positives, True Negatives, False Negatives.

Chapter 3

Application

This chapter illustrates how the mathematical and computational theories can be verified in practice. We start by giving the reader an idea about the dataset used and detail the data analysis procedure followed to choose the proper sample for training. Next, we use all models (i.e., NCDEs, MLPs, RFs, and RFS) to predict whether the patients witness an event (i.e., death) before/after the median survival time. We try to learn what was mentioned in two ways: First, by only using the first values of every test, and then by using the whole dataset. We finally discuss the results, concluding some data features and showing that NCDEs should be considered valuable techniques when working on irregular time series.

3.1 Data analysis

In this section, we introduce the dataset. We then explain why the subset previously chosen by Roche was not a good choice. We then detail how we chose a better part of the dataset for training. We use the dataset, “Flatiron Advanced Non-Small Cell Lung Cancer”, provided by Roche, which contains lab values for patients diagnosed with lung cancer. The data has mainly three dimensions:

1. The number of patients. The total number of patients is around 70,000.
2. The number of tests. Each patient will eventually be asked to perform different tests (e.g., Hemoglobin, Platelets, etc.). The results of these labs are the data that will be used for training. In total, around 170 labs are available in the dataset.
3. The time at which the lab was taken. Data is available from each patient’s

first lab results until the patient gets censored¹, by days. Since we don’t expect every patient to take every test daily, we end up with irregular time series, which motivated us to use NCDEs.

Choosing the entire dataset for training is not feasible for different reasons.

- The models’ purpose is to predict patient information (whether binary classification or survival curves, as explained in later sections). Predicting this information after the patient has witnessed an event is useless. Hence, we would prefer to perform predictions using a specified time window to be able to act on the prediction (e.g., modify the patient’s treatment) before it is too late.
- Using the entire dataset would lead to larger arrays, making the training slow and unfeasible. In addition, patients will witness an event at different times, making things much more complicated.

The subset of the data previously chosen by Roche included around 60000 patients, 20 labs, and a time window of $[-50, 40]$, relative to the diagnosis date of each patient (i.e., 50 days before, until 40 days after). We first sorted the labs in terms of number of tests taken by patients. The most popular lab tests seemed to be “Hematocrit” and “Platelets”. The histograms in figure 3.1 illustrate why the subset chosen by

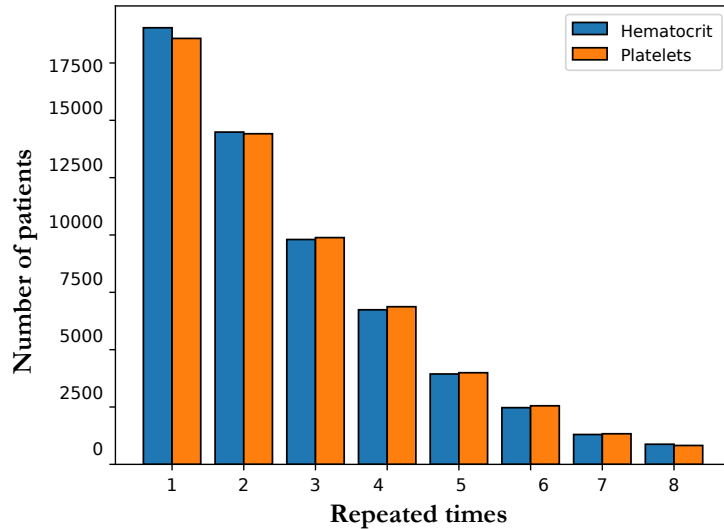


Figure 3.1: Histogram showing how often patients repeat the two most abundant tests in the old subset of the data.

¹This is usually the result of a patient witnessing an event/stopping to have tests.

Roche was not a good choice. Most patients have, on average, either 1 or 2 tests per lab for the two most popular labs. Taking a closer look, the rest of the labs have similar figures, making the dataset nearly tabular. Since we expect time series, we propose a different way to choose a more suitable part of the data for training.

In our analysis, we decided to fix the number of tests, the number of patients, and some criteria for the patients. We then extended the time window to reach what was already specified.

Choice of tests: From around 170 tests, one would wonder if they would be relevant for training. Figure 3.2 depicts how many patients took a certain test. The tests were sorted by their abundance and then numbered by what we called the “Lab Index”. In other words, the test with Lab Index 0 had been taken by patients the most. The left of figure 3.2 shows that the number of tests is large and is decreasing for the first 36 labs. Looking to the figure’s right would highlight the importance of only choosing a subset of the tests. Since the number of tests is decreasing significantly, we reach a point after which adding a dimension is not justified by the added number of tests. For this report, we only chose the first 20 labs. In other words, we take all the tests to the left of the vertical line drawn to the figure’s left. Note that the number 20 is a hyperparameter, and its choice is not unique. However, we believe the number of tests should be at most 36 from the figure provided.

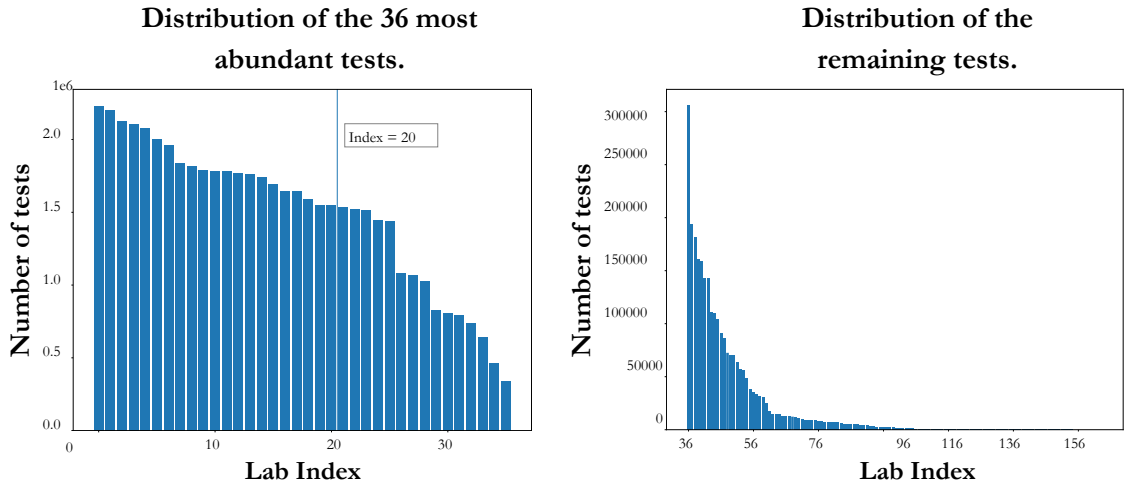


Figure 3.2: Figure showing the distribution of the number of tests for the 36 most popular tests (to the left) and the remaining tests (to the right).

Choice of patients: The same logic encouraged us to carefully choose patients instead of using all 70000 for training. The choice of this dimension, though, was limited by two factors,

1. The number of patients should be large enough for the test set to represent the data.
2. Some patients only did a small number of every test. Hence, they will not be useful for training.

The first point was taken care of by running multiple models over different numbers of patients and studying the effect on the standard deviation when cross-validation is used for a specific binary prediction. We found out that choosing less than 10000 patients would usually lead to high standard deviations, which could mean that the test dataset is small enough to have unique features in it².

On the other hand, the second point was taken care of by enforcing that every chosen patient has to have at least 200 test values in total. This would usually mean 10 tests per lab on average, but it would also allow patients to have more values of a specific test than another. We believe this choice would guarantee some long time series without creating a biased dataset (we verify this by visualizing the time series we get later).

Time window: Choosing the length of the time window is a crucial parameter. For this project, we decided to choose a time window around the date of diagnosis. Ideally, we want the window to be as long as possible to find 10000 patients with at least 200 values for the 20 most redundant labs (all the constraints mentioned before). On the other hand, the problem with extending the window is that patients will start witnessing an event within the time window! This is problematic and will affect training. All in all, what we did to find the best time window is represented in diagram 3.3. The graph illustrates the steps performed and how the tensor³ containing the data changes size.

For the first step in the diagram, choosing a window was not trivial since we had two extremities to fix. Based on our data analysis, depicted by the left part of figure 3.4, extending the window backward will probably not be worth it. Accordingly, we decided to fix the left side of the window to -50 (which was previously chosen by Roche) and extend the right side until our data meets all criteria. The output of

²Again, 10000 is a hyperparameter, standard deviations later in this section will show that this was an acceptable choice.

³The size tensor is organized as [patients, time, tests].

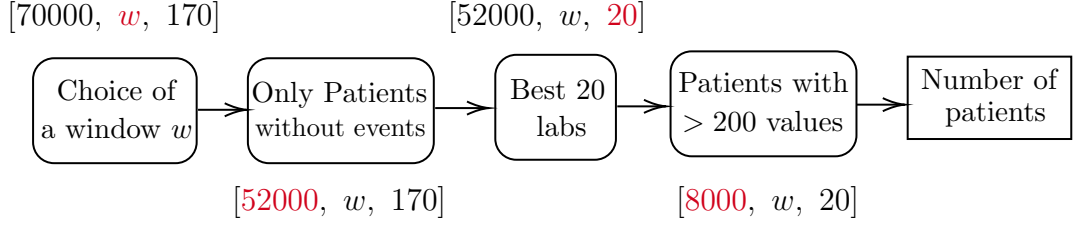


Figure 3.3: Diagram illustrating how the dataset was chosen. The brackets show how the size of the tensor containing the data is changing with every step.

diagram 3.3 for different right window bounds can be observed to the right of figure 3.4. We note that even though the number of patients witnessing an event within the window increases when the window is enlarged, a right bound of 160 guarantees that we can find 10000 patients with the criteria previously mentioned.

All in all, we summarize our dataset as follows:

- 20 tests, the ones that are the most popular amongst all patients.
- A time window of $[-50, 160]$, hopefully long enough to give us a time series (verified next) and not too long to slow the models.
- Around 10000 patients, none of them witnessed an event within the time window, and all of them have at least 200 results within the window.

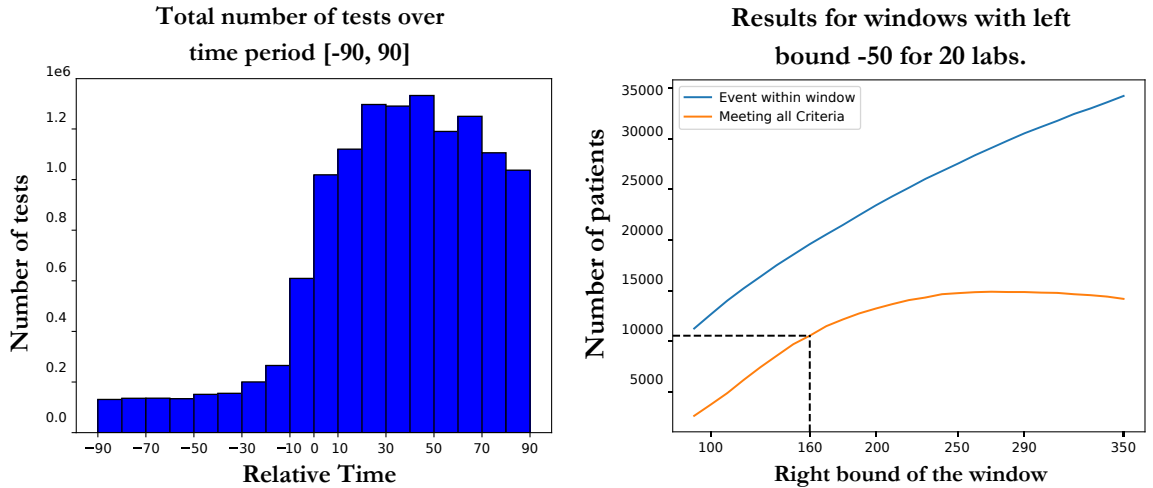


Figure 3.4: Figure showing the distribution of the number of tests over a time period of $[-90, 90]$ (to the left), and the number of patients witnessing an event/ meeting all criteria when extending the window (to the right).

To verify that we had time series, we show in figure 3.5 the number of patients that repeated the tests with lab index 0 and 20 (i.e., most and least popular tests amongst the 20 selected). As shown in the figure, we have a relatively long time series for both (on average, patients repeated the first test 20 times and the last around 15 times). An important thing to note is that for the test with a lab index of 20, there are still around 1400 patients who haven't even done the test! This is why we chose to enforce a total number of values for the patients and not a certain number of values per lab. This is a crucial property of the dataset because we believe the models should be able to learn even when some patients have no time series in some channels.

3.2 Results

In this section, we evaluate the performance of NCDEs for a binary classification task. We try to approximate whether patients will witness an event before or after the median time of the population. As a baseline, we compare NCDEs with MLPs. In addition, we also compare NCDEs with the state-of-the-art technique, Random Forests. This section will be divided into two main parts:

1. Training over first values. In this part, we will show that NCDEs outperform MLPs even when only using the first values for training. We also conclude that the timing of the first tests is important.
2. Training over all the lab values. We show that when using time series, NCDEs

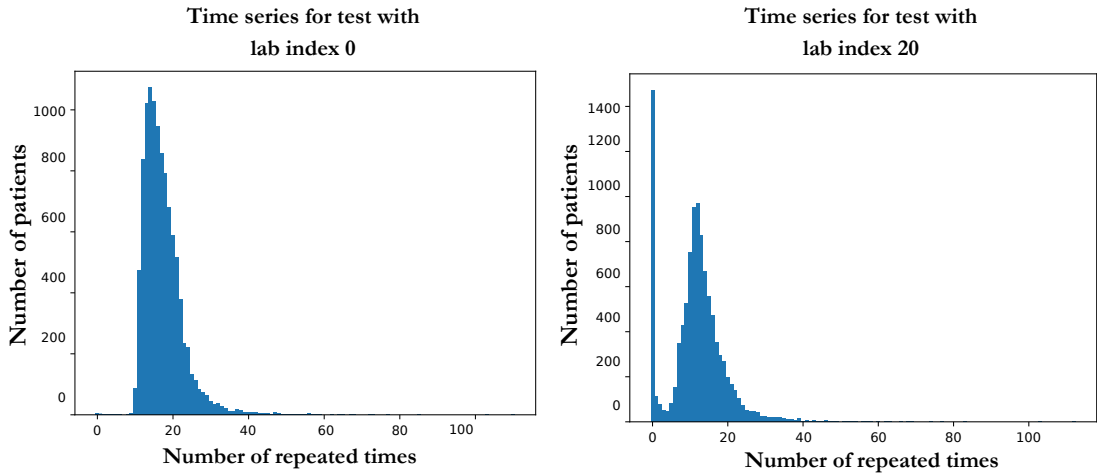


Figure 3.5: Figure showing the time series we got after the dataset has been chosen for the most/least popular labs amongst the 20 (to the left and right, respectively).

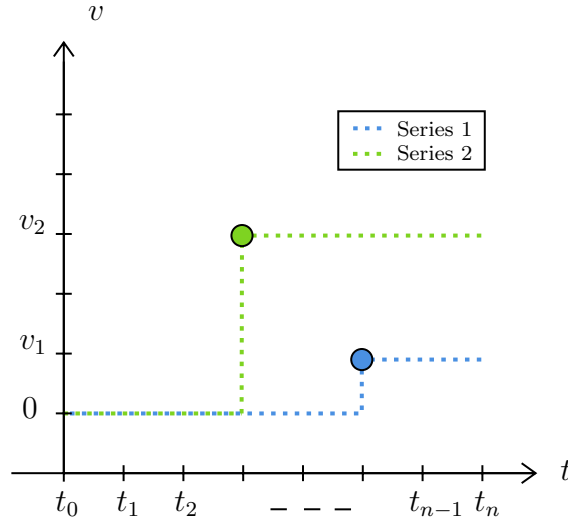


Figure 3.6: Figure showing an example of the rectilinear interpolation used when training using the first values only.

outperform MLPs by far, matching the performance of the state-of-the-art RF model. Consequently, we will use RFS (the signature) and show how they improve the results attained by RF.

For each training, we will explain how the data was pre-processed, mention the models' parameters, and present and discuss their accuracy.

3.2.1 Training over first values

3.2.1.1 Data Pre-processing

The data had to be pre-processed differently for MLPs, RFs, and NCDEs. The time dimension has to be shrunk for MLPs and RFs since these can only have a two-dimensional array as their inputs. We reduce the time dimension by choosing the first values in this case.

On the other hand, for NCDEs, we keep the time series but replace all the values by NaNs except the first value for every patient and test. One must remember that in NCDEs, an interpolation is required. In this section, we choose a rectilinear interpolation and fill the first values by 0 whenever needed, as illustrated in figure 3.6. The main idea of the interpolation procedure depicted in the figure is to include two main pieces of information: To start, the first value, which is included as the increment (or the vertical jump shown in the figure). This is why the choice of the

value at $t_0 = 0$ is important. The second is the jump timing, which the ODE solver will automatically consider a discontinuity.

Interpolating is not the only pre-processing step for NCDEs. What we found useful in practice is to stack a parameterization stream to the data. We note that throughout the proofs earlier in the report, we mentioned that a monotonic channel is sufficient for us to guarantee uniqueness. Otherwise, CDEs will exhibit tree-like invariance. In practice, we noticed that the vector starting from t_0 , ending at t_n , and having increments of 1 is usually not the best choice. It is a hyperparameter that affects the training capacity of NCDEs (more details in the Appendix A.6). In this case, we choose the vector $[0, 0.5]$, with the required spacing for the length of the vector to be equal to the time window (otherwise, stacking the vector to the dataset doesn't make sense).

3.2.1.2 Results and Discussion

Now that we have established how the data was pre-processed, we mention some of the parameters used by the different models for training. Readers are referred to Appendix A.1 and A.2 for more details about the models and their hyperparameters.

1. MLP: Cross-validation over five splits, dropout of 0.3, batches of size 64, 100 epochs, Adam optimizer with a learning rate of 0.1 (Details about the choice of the learning rate in Appendix A.3).
2. NCDE: Cross-validation over five splits, dropout of 0.3, batches of size 64, 3 epochs, Newton's method with a step size of 0.8, Adam optimizer with a learning rate of 0.1.
3. RF: 300 estimators, 50 random states, the minimum number of samples in leaf nodes is 15, and the minimum number of samples to split a node is 10.

We are now finally ready to present the results, which are documented in table 3.1. The average and standard deviation in the table are either over the folds (for MLP and NCDE) or over random states (for RF). The fact that the NCDE outperforms the MLP when only using the first values is interesting. Technically, we're giving both MLPs and NCDEs the same values. However, we expect that the advantage NCDEs have is the timing of the first values, which is extracted from the jump within the rectilinear interpolation. We conclude that the time at which a patient is asked to take the first test contains information when it comes to training. This verifies our proposition that the NCDE can capture some behavior the MLP misses. On the other

Method	MLP	NCDE	RF
Average AUC	0.552	0.57	0.597
STD AUC	0.0056	0.0083	0.0121

Table 3.1: Table showing the result for all the models proposed when training is done over the first values only. The average and STD are over five splits (cross-validation) for NCDE and MLP and 50 random states for RF.

hand, RF, the currently leading technique for such predictions, still outperforms the NCDE and the MLP. Yet, we expect NCDEs to do much better when the data is in time series. We cover those results in the following section.

3.2.2 Training over time series

In this section, we expect NCDEs to perform better since they can capture behaviors/trends within time series, most of which could be missed when reducing the dimension for MLPs and RFs. We also use the RFS model to show how the signature could be helpful for training.

3.2.2.1 Data Pre-processing

For MLPs and RFs, it is illogical only to use the first value now. Alternatively, we reduce the time dimension by computing the mean of every channel without considering the missing values. For NCDEs, we try a cubic and a regular linear interpolation, leading to similar results. Hence, we only show the results when cubic interpolation is used. Similarly to the previous case, we stack a parameter vector over $[0, 0.5]$.

On the other hand, we spend some time explaining how RFS used the signature as a feature map since it is not a trivial procedure. The main challenge when using the signature as a feature map is that the size of the signature increases exponentially for high-dimensional data. Accordingly, finding the signature for all 20 labs (i.e., 20 dimensional array) is impractical. Alternatively, what we did is illustrated in diagram 3.7. As seen in the figure, for each patient, we divide the two-dimensional array into width channels, say, w . Then we compute the signature of these channels (which will

always be of width 1 and variable length depending on the signature depth N), then stack all of these into a long vector. Hence, we reduced the two-dimensional array for each patient into a vector. We finally stack these vectors to get the 2D array, which is then used as the input for Random Forests. Experimentally, we found that choosing $w = 2$ and $N = 10$ provides a decent choice. A larger w would cause the runtime to be higher, and a higher depth no longer affects training.

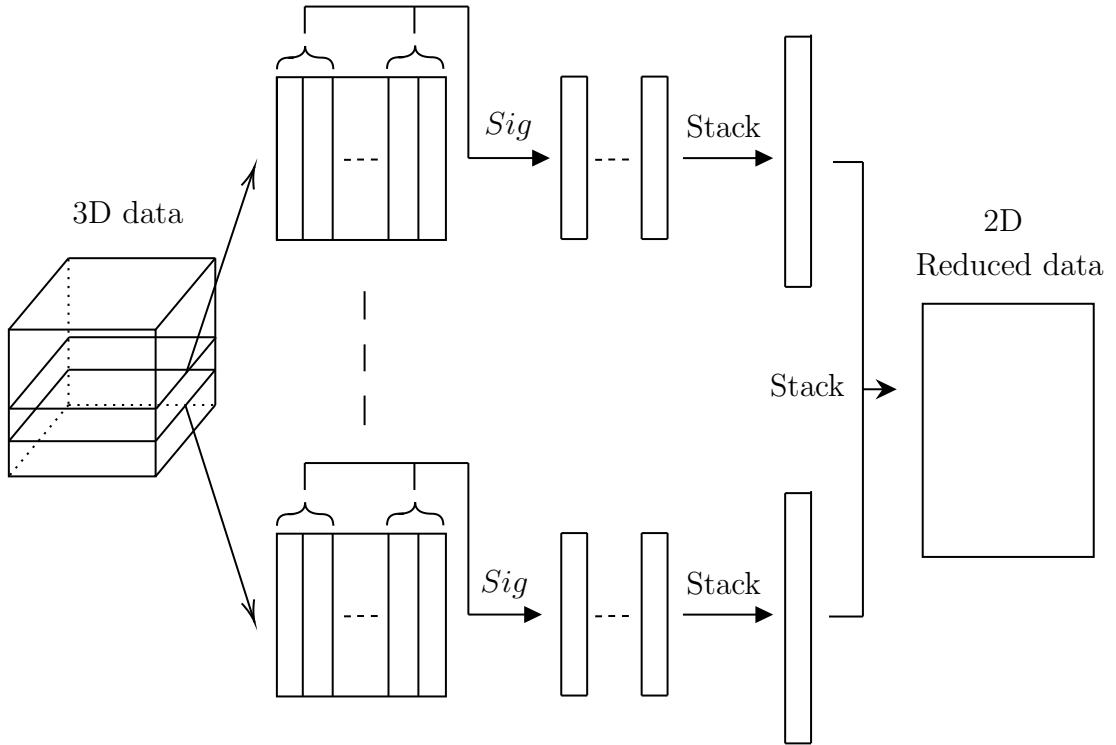


Figure 3.7: Diagram depicting how the signature was used as a feature map for Random Forests, we chose to take the signature of every 2 labs separately.

Note: For simplicity and to be able to present results nicely, we don't use cross-validation (for MLPs and NCDEs) and choose only one random state (for RFs and RFSs). We believe that the results in the previous section show that the data is well-mixed; hence, it is unnecessary to go through this. From now on, we assume a model is better/worse than any other model if the difference between their AUCs is at least 0.01 (from STDs in table 3.1).

3.2.2.2 Results and Discussion

The models used in this case are very similar to the ones detailed in the previous section. We highlight the differences below,

1. MLP: Same parameters with no cross-validation.
2. NCDE: The time step is now $0.8/TW$ where TW is the length of the time window. This is necessary since the ODE solver must be fine enough to capture all the lab values. Otherwise, the same parameters with no cross-validation.
3. RF: The same parameters but only for random state 0 (we ensured this is equivalent to the data used for NCDEs and MLPs).
4. RFS: Same parameters as RF but using the signature to reduce the dimension instead of the mean.

The ROC curves for all methods proposed can be seen in figure 3.8. As can be seen from the figure, the NCDE model outperforms the MLP with a much more significant gap now. This verifies that NCDEs can capture some trends within the time series, but most are missed when taking the mean for MLPs. For example, NCDEs can see that test number 1 has been taken 5 days before test number 3. In addition, maybe the fact that patient number 200 had 5 tests on the same day is essential for training. All this information is lost in MLPs by just taking the mean value. Even more

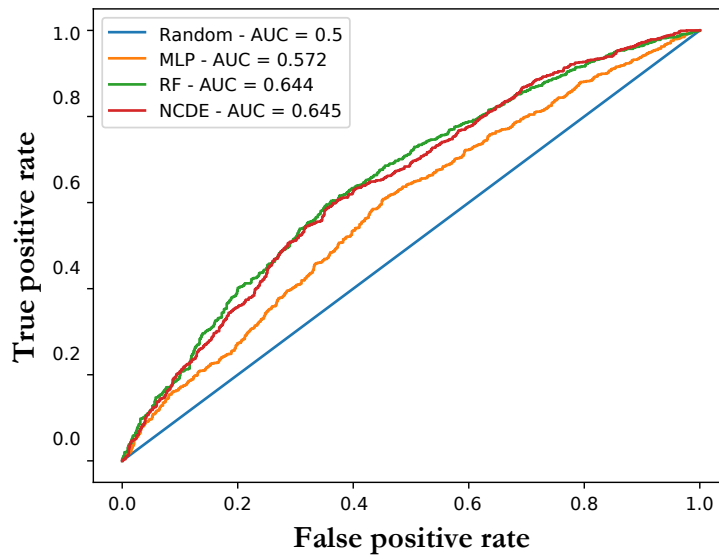


Figure 3.8: ROC curves for all three models when training over the entire dataset.

interesting is that the NCDE model does as well as RF, the state-of-the-art model! This verifies that both theory and practice agree that NCDEs should be considered a great model whenever irregular time series are considered. Many questions arise from these results. As a reminder, we chose our time series with an average length of 10. What would happen if we had even longer time series? Would RF struggle enough until NCDEs become the state-of-the-art model? On the other hand, would NCDEs also be good predictors for different types of tasks, for instance, continuous survival predictions?

The fact that NCDEs are doing that well means that there is more to learn from the data than just the mean. Accordingly, we hope that by taking the signature of the data instead of the mean, we could efficiently facilitate the performance of Random Forests. This can be observed as well in figure 3.9 since RFS achieves a score of $AUC = 0.667$ compared to $AUC = 0.645$ without the signature. Hence, we verify that the signature includes trends from the time series that were missed before. This highlights the significance of using the signature as a feature map when training. Our result sets the floor for much research on using the Signature as a feature map. Is it possible to find better combinations than picking two lab tests consecutively at a time? In other words, do specific labs provide more information when coupled together? On the other hand, is it worth finding the signature over more than 2 features at a time? Will the optimal truncation depth vary in this case?

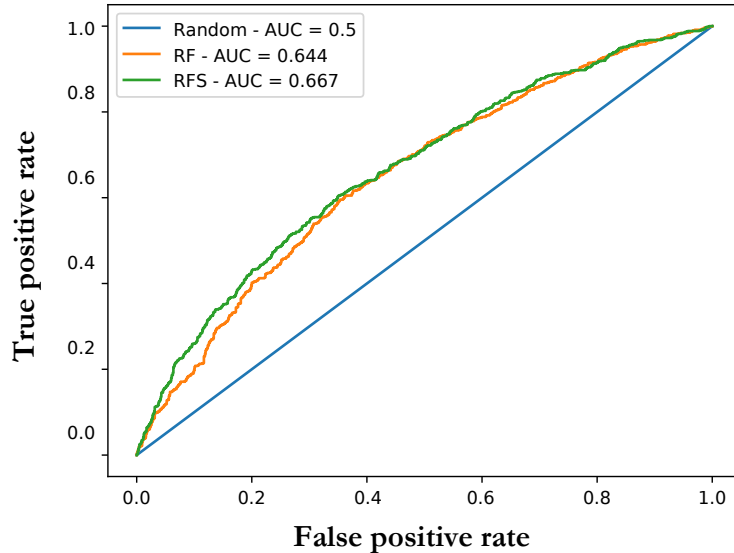


Figure 3.9: ROC curves for RF and RFS when training over the entire dataset.

3.2.3 Additional Work

In this section, we provide the additional work that was done in the project but was not detailed in the report.

1. The limitation of NCDEs is that they can be very slow when used on real-world data. For us to be able to do all the fine-tuning and debugging, we also implemented parallelization over the nodes of Roche, which reduced the run time to up to 10 times for tasks with moderate computational overhead. Details can be found in Appendix A.5.
2. The project also included going through all of Roche’s training to learn how to use Roche’s nodes and handle/access the data since it is considered confidential. We also had to go through all of Roche’s old codes to access and process the data from the dataset until it became in the organized tensor we mentioned before.
3. We also proposed and implemented a two-stage training based on first values. Since first values and their timing seem to have an important role when it comes to exercise, we implemented a two-stage training that starts with only the first values and then loads the model and the final condition into the initial state of a new model, which is trained again over the increments of the lab values. Unfortunately, the problem with that is the number of hyperparameters to be tuned is twice the original one! Hence, we couldn’t get the results before the report’s deadline. These will be part of the remainder of the project in the following two months.

Chapter 4

Future Work and Conclusion

4.1 Future Work

We believe this project is just the beginning of the exploratory phase of NCDEs. The project was not only limited to this report. We will continue working on the topic until November. We also expect to pursue the research in the CDT in Oxford starting next year. In the next few months, we hope to include the following,

1. Fine-tune the parameters for the two-stage training to see if NCDEs will perform better when the proposed strategy is used.
2. Modify the subset of the data to get a more extended time series. In practice, working on a randomly chosen dataset (i.e., with short time series) led to RFs consistently outperforming NCDEs. However, more extended time series are what made NCDE's score rise. We believe that if time series are long enough, NCDEs could become the state-of-the-art model.
3. Work on different prediction tasks. This report only covers the work we did on binary predictions. However, the project also entitled other tasks, such as predicting survival curves. Using algorithms like DeppSurv [20], these are currently being implemented, and we hope to see how NCDEs scores are compared to MLPs and Random Survival Forests (a survival model based on RFs) on these tasks.

4.2 Conclusion

To conclude, the project positively impacted the company Roche and the research group in Oxford. From Oxford’s perspective:

1. Since the group in Oxford has not used NCDEs on real-world data before, the project allowed us to understand how to tune parameters such as the initial condition, interpolation techniques, and others to lead to a trainable model.
2. The project showed that NCDEs also perform well for first values. This initiated the idea of a two-stage training that could be implemented in the library `torchcde`.
3. We showed that the theory claimed by the team working on NCDEs can be validated in practice: NCDEs are a great model for irregular time series and could be the state-of-the-art technique over longer ones.
4. Our results highlight the feasibility and practicality of using the Signature as a feature map to improve most of Machine Learning’s models, especially Random Forests.

On the other hand, from Roche’s side:

1. We solved Roche’s problem by successfully implementing NCDEs for survival predictions.
2. We performed a detailed data analysis where we showed researchers from Roche why their chosen subset of the data was not a good choice, then implemented the required codes to extract the required subset of data.
3. We implemented GPU parallelization, which helped make the codes up to 10 times faster for moderate tasks.

As a takeaway from this report, even though state-of-the-art models such as Random Forests can do well on irregular time series, any technique that doesn’t include the whole data in the model will struggle when the time series become longer. Whether using NCDEs, or the Signature as a feature map allows us to track interesting trends from the time series, which other models miss. Even though implementing both of these is not a trivial task, and there is not enough documentation on the topic yet, we hope that the results in this report show that it is worth it and that both proposed methodologies could lead to models with state-of-the-art performances.

Appendix A

Useful details

A.1 Tuning parameters

NCDEs have numerous parameters that highly affect their effectiveness. Accordingly, finding the correct parameters when using NCDEs is not trivial. In this subsection, we show how we chose parameters by running the model over the small time window (i.e., window of $[-50, 40]$, previously used by Roche) and using only the first values. Throughout this section, cross-validation with five folds is used. Hence, we use the mean and standard deviation of the AUC to evaluate the model performance.

1. ODE solver, step size, and vector field dimension:

As seen in table A.1, with the first method (i.e., Dopri5 with 16 hidden layers, the default), NCDEs are learning, and both the average and standard deviation are acceptable. However, the time is very long, which we expect is because of the small timestep¹. Reducing the number of hidden layers (the second column) solves the time issue. However, the model is not learning ($\text{AUC} \approx 0.5$). We try methods with fixed timesteps (whether Euler or RK4). Unfortunately, since RK4 requires 4 function evaluations, it is also prolonged! Euler’s method (in the third column) was the most promising one. We then fine-tuned other parameters (in the last two columns). Since we are training only on using the first values, a fine timestep of 0.1 days would not be necessary. The last column shows that we could use a timestep of up to 0.8, which decreases the time significantly while keeping a steady AUC. Further, we noticed from observing loss plots over the epochs that most learning happens within the first few epochs. Reducing the

¹Remember, Dopri5 is an adaptive time step method. Hence, we can not predict what would happen to the time step.

Method	Dopri5 (16 hidden layers)	Dopri5 (8 hidden layers)	Euler (step 0.1)	RK4 (step 0.1)	Euler (step 0.1)	Euler (step 0.8)
avg AUC	0.5813	0.5063	0.6171	0.6134	0.6125	0.6142
std AUC	0.0057	0.0024	0.0034	0.0031	0.0022	0.0033
Epochs	8	8	8	8	3	8
Time	6.33 (d)	17.79 (m)	1.9 (d)	7.4 (d)	0.78 (d) = 18 (h)	5.23 (h)

Table A.1: Table showing the results using NCDEs for different parameters such as the method for solving ODEs, the time step, or the number of epochs when used over the window $[-50, 40]$.

epochs' number showed that only 2 or 3 epochs perform as well as 8 or more! We used 16 hidden channels and Euler's method with a step of 0.8 for 3 epochs when training on first values. We reduce the effort to $0.8/T$, T being the time window length when we train over the whole dataset.

2. Initial condition for the ODE solver: Surprisingly, the choice of the solver's initial condition significantly affected the training results. We proposed many initial conditions, including the first value for each patient (which seemed to be the best for both trainings), the mean of the population for each lab, and the mean value for each patient. Not only the first proposition was the best, NCDEs were not trained at all when the other two offers were tried. Theoretically, the initial condition must depend on the initial value because otherwise, the ODE solver only knows about the derivative of the control. Hence, there is a risk of translational invariance [21].
3. Interpolation Techniques Different types of interpolation were used for both trainings. When training on the first values, a rectilinear interpolation was used. We also had to pre-process the data as mentioned in the report. On the

other hand, when training using the entire dataset, we tried both linear and cubic interpolations. Both interpolation techniques gave similar results and took similar times to run. We expect the time aspect since we used an ODE solver with a fixed time step.

A.2 Models parameters

In this subsection, we detail the parameters used in both MLPs and NCDEs (parameters used for Random Forests can be found in the report). The parameters are as follows:

1. Batch size: A batch size of 64 was used for both models.
2. Dropout: Training NCDEs and MLPs without dropouts led to the model memorizing the data. We finally settled on using a dropout with a probability of 0.3.
3. Loss function: For both MLPs and NCDEs, we use the binary cross entropy with logits loss function defined as follows:

$$l_n = -w_n[y_n \log \sigma(x_n) + (1 - y_n) \log(1 - \sigma(x_n))] \quad (\text{A.1})$$

4. Activation functions and number of Layers: For MLPs, one hidden layer of depth 512 with a hyperbolic tangent as an activation function was used. For NCDEs, one layer of depth 64, 16 channels for the vector field, a hyperbolic tangent as an activation function, and a Relu as the initial function xi_0 .

A.3 Learning rate for NCDEs

Again, they may be very slow because running NCDEs requires using the NODE solver many times. Alternatively, it is not easy to try different hyperparameters. A crucial element was the learning rate. As shown in figure A.1, the optimal choice is around 0.1, which we chose as our learning rate for the Adam optimizer. This is a good choice, especially since we only need three epochs for training NCDEs.

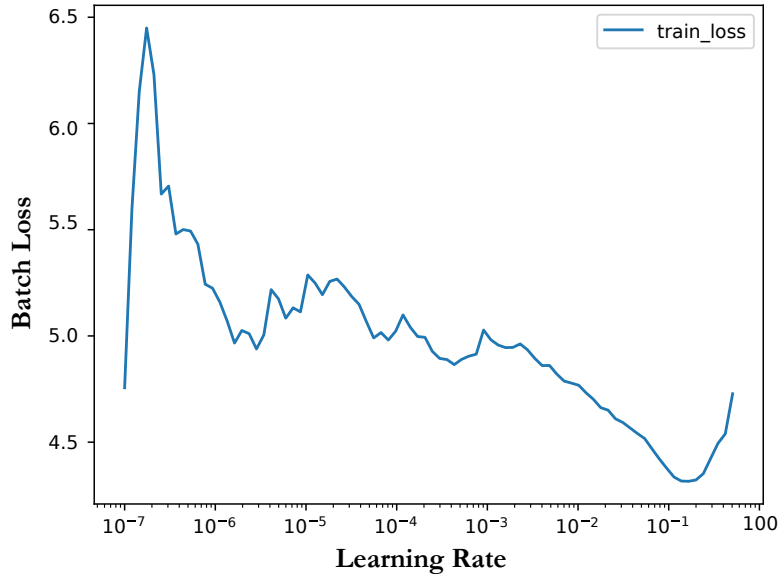


Figure A.1: Graph showing the effect of learning rate on a batch's loss.

A.4 Further Data Analysis

We show what motivated us to choose a subset of the patients instead of selecting all 70000. As shown in figure A.2, most patients took between 0 and 200 tests (to the figure’s left). In the zoomed version (to the right of the figure), we can see that many patients are taking even below 10 or 20 tests in total! Since we chose 20 labs, this means around 1 test per channel or less. Accordingly, we enforced that patients have 200 lab values as a selection criterion to ensure that the subset of the data has time series.

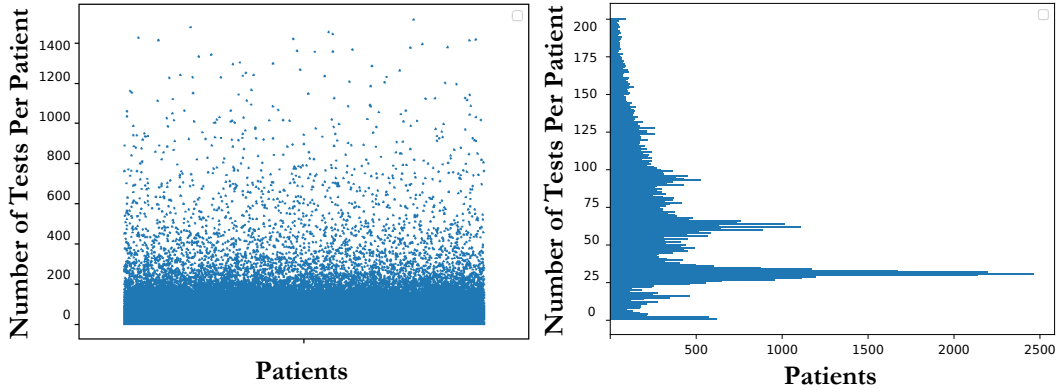


Figure A.2: Visualisation of the frequency of patients taking tests.

A.5 GPU parallelisation

An important part of the project was parallelizing over GPUs to make the models run more efficiently. This was done by using Pools in Python, and the increase in efficiency can be summarized in the following table: As seen in the table, we tried to use as much as possible from Roche’s nodes, successfully reducing the computational time more than ten times!

#	1	2	3	4
GPU	0	0	5	5
CPU	48	48	5	64
Time (mins)	496.5	112.3	79.9	41

Table A.2: Table summarizing the increase in efficiency when parallelizing over GPUs.

A.6 Stacking Parametrization Vector

As mentioned in the report, we stacked a monotonically increasing channel to the data to ensure that CDEs won't experience tree-like equivalence. Even though whenever NCDEs were used before, the vector chosen would usually be from 1 to the length of the time window with increments of 1, or just a vector from 0 to 1 with the required step to have the correct dimension. We discovered that, in practice, we could use different vectors to improve training. A representation of training when using all the lab values with different parameter vectors can be seen in the following table: Therefore, we chose to work with the vector $[0, 0.5]$ as the parametrization vector.

Vector	$[0, 0.1]$	$[0, 0.5]$	$[0, 1]$	$[0, 2]$
AVG AUC	0.638	0.645	0.624	0.627

Table A.3: Table Illustrating how changes in the parameterization vector affect training

References

- [1] BN Biswas, Somnath Chatterjee, SP Mukherjee, and Subhradeep Pal. A discussion on euler method: A review. *Electronic Journal of Mathematical Analysis and Applications*, 1(2):2090–2792, 2013.
- [2] David P Blecher and Vern I Paulsen. Tensor products of operator spaces. *Journal of Functional Analysis*, 99(2):262–292, 1991.
- [3] Horatio Boedihardjo, Xi Geng, Terry Lyons, and Danyu Yang. The signature of a rough path: uniqueness. *Advances in Mathematics*, 293:720–737, 2016.
- [4] Anne-Laure Boulesteix, Silke Janitza, Jochen Kruppa, and Inke R König. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):493–507, 2012.
- [5] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
- [6] Kuo-Tsai Chen. Integration of paths—a faithful representation of paths by non-commutative formal power series. *Transactions of the American Mathematical Society*, 89(2):395–407, 1958.
- [7] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.
- [8] Ilya Chevyrev and Andrey Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.
- [9] JA Clarkson. On double riemann-stieltjes integrals. *Bulletin of the American Mathematical Society*, 1933.
- [10] Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Juan-Pablo Ortega, and Josef Teichmann. Discrete-time signatures and randomness in reservoir comput-

- ing. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6321–6330, 2021.
- [11] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *Advances in neural information processing systems*, 32, 2019.
 - [12] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.
 - [13] Peter Friz and Nicolas Victoir. A note on the notion of geometric rough paths. *Probability Theory and Related Fields*, 171(3-4):669–726, 2018.
 - [14] Peter K Friz and Martin Hairer. *A Course on Rough Paths*. Springer, 2014.
 - [15] Peter K Friz and Nicolas B Victoir. *Rough Path Analysis and Stochastic Differential Equations*. Cambridge University Press, 2010.
 - [16] Massimiliano Gubinelli. Controlling rough paths. *Journal of Functional Analysis*, 216(1):86–140, 2004.
 - [17] Ben Hambly and Terry Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, pages 109–167, 2010.
 - [18] Christopher O Imoru and Memudu O Olatinwo. On the stability of picard and mann iteration processes. *Carpathian Journal of Mathematics*, pages 155–160, 2003.
 - [19] Jerry Johnson. Lipschitz spaces. *Pacific Journal of Mathematics*, 51(1):177–186, 1974.
 - [20] Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18(1):1–12, 2018.
 - [21] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series, 2020.
 - [22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

- [23] Maud Lemerrier, Cristopher Salvi, Theodoros Damoulas, Edwin Bonilla, and Terry Lyons. Distribution regression for sequential data. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3754–3762. PMLR, 13–15 Apr 2021.
- [24] Terry Lyons. Differential equations driven by rough signals (i): An extension of an inequality of lc young. *Mathematical Research Letters*, 1(4):451–464, 1994.
- [25] Terry Lyons and Andrew D McLeod. Signature methods in machine learning. *arXiv preprint arXiv:2206.14674*, 2022.
- [26] Terry J Lyons, Michael Caruana, and Thierry Lévy. *Differential equations driven by rough paths*. Springer, 2007.
- [27] Terry J Lyons and Zhongmin Qian. *An Introduction to Rough Paths*. Oxford University Press, 2007.
- [28] James Morrill, Patrick Kidger, Lingyi Yang, and Terry Lyons. Neural controlled differential equations for online prediction tasks. *arXiv preprint arXiv:2106.11028*, 2021.
- [29] Francis Sahngun Nahm. Receiver operating characteristic curve: overview and practical use for clinicians. *Korean journal of anesthesiology*, 75(1):25–36, 2022.
- [30] Michael A. Nielsen. <http://neuralnetworksanddeeplearning.com/chap2.html>, Jan 1970.
- [31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [32] Ricky T. Q. Further documentation. https://github.com/rtqichen/torchdiffeq/blob/master/FURTHER_DOCUMENTATION.md.
- [33] Rimhak Ree. Lie elements and an algebra associated with shuffles. *Annals of Mathematics*, 68(2):210–220, 1958.

- [34] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [35] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.
- [36] Carl Runge and Martin Kutta. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1901.
- [37] R. D. Skeel and B. J. Berzins. Adjoint sensitivity analysis of differential-algebraic equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):519–539, 1990.
- [38] R. D. Skeel and B. J. Berzins. Adjoint sensitivity analysis of differential-algebraic equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):519–539, 1990.
- [39] Elias M Stein. *Singular integrals and differentiability properties of functions*. Princeton university press, 1970.
- [40] Elias M Stein. *Singular integrals and differentiability properties of functions*. Princeton university press, 1970.
- [41] Marshall H Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254, 1948.
- [42] Andrei Tychonoff. Ein fixpunktsatz. *Mathematische Annalen*, 111(1):767–776, 1935.
- [43] Min Wu, Yamei Yu, Lunjie Luo, Yuehao Wu, Jian Gao, Xiangming Ye, and Benyan Luo. Efficiency of repetitive transcranial direct current stimulation of the dorsolateral prefrontal cortex in disorders of consciousness: a randomized sham-controlled study. *Neural plasticity*, 2019, 2019.
- [44] Y Xie, L Li, Catherine Rychert, Nicholas Harmon, Petros Bogiatzis, and Daniel B Peter. Computation of hessian vector product by spectral-element and adjoint methods using wavefield storage scheme. In *AGU Fall Meeting Abstracts*, volume 2020, pages S064–0005, 2020.
- [45] William Henry Young. On multiple integrals. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 93(647):28–41, 1917.