Constrained Machine Learning Methods for Biomedical Data Analysis

By

Dominic Danks

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

Institute of Cancer & Genomic Sciences
College of Medical and Dental Sciences
University of Birmingham

January 2023

ABSTRACT

In recent years, machine learning (ML) methods have shown great promise in a variety of application settings, with ML-based systems now representing the state of the art in a wide variety of intelligence-based computational tasks ranging from image classification to natural language processing. Their impact in health data science is also becoming more tangible, with ML-based systems beginning to be adopted as genuinely attractive prospects by clinicians and health professionals as tools to improve disease understanding and clinical pathways. The importance of such models is also likely to continue to grow as health settings become increasingly data-aware and data-driven.

However, the field of ML-based health data science is far from fully developed, with many clinically relevant settings not yet fully explored and with limited efforts to consider how general ML methodology can be tailored to the health data science setting. In this thesis, we show that by considering the constraints relevant to health-based ML problems and explicitly involving these in our models, it is possible to derive novel approaches which tend to retain the performance synonymous with machine learning whilst offering the plausibility and interpretability of model outputs typically associated with more traditional statistical approaches.

This thesis demonstrates this approach in three health-relevant settings. We begin by considering the problem of pseudotemporal modelling, where we highlight that it is often the case that some prior knowledge is present about the pseudotemporal trajectory which can

be exploited to aid learning and reduce human burden. Next, we consider the problem of time-to-event modelling (survival analysis) from a novel machine learning perspective and demonstrate that by suitably constraining a neural network it is possible to model general survival functions and in turn obtain strong model performance with little model tuning or computational difficulty. Finally, we consider the problem of survival analysis in the presence of longitudinal data and present a series of approaches which derive from our constrained general survival model and offer various benefits over established methods.

ACKNOWLEDGEMENTS

I have numerous people to thank for making my PhD experience so enjoyable and enriching. I would first like to thank Chris Yau for his excellent supervision throughout. As well as performing the usual tasks expected of a PhD supervisor — insisting on more figures in papers and trying to decode feedback from a "Reviewer 2" — Chris went above and beyond to facilitate opportunities and collaboration outside of what would have otherwise been possible, for which I am truly grateful. He also provided a fountain of great advice at various points throughout the PhD and made for an entertaining Slack buddy during England football matches!

I would also like to thank Alastair Denniston and Pearse Keane for welcoming me into their community and for reminding me of the end goal for much of the work in this space, namely to improve the treatment pathways of patients. I would also like to thank Alastair for his consistent support and words of encouragement, as well as his clinical perspectives on my work. I am also indebted to Alastair, and to Andrew Beggs, for their continued assistance as internal supervisors.

I was fortunate to spend three months of my PhD at NHSX, where I was able to see how health data science is being used to improve patient journeys now and to contribute to their mission. I would like to thank NHSX and HDR UK for this opportunity and to thank Jonny, Dan and Sarah in particular for their guidance and hospitality.

So much of the enjoyment of a PhD derives from collaborating with great colleagues on interesting projects, which I have been fortunate enough to do on numerous occasions. Special individual thanks go to Vasilis, Fabian, Chris (Williams) and Matthew for their

contributions and to Krish and the wider OPTIMAL group for their welcome and direction. Special thanks also go to the other members of the Yau group, in particular to Kaspar for interesting discussions on Variational Autoencoders, amongst other things.

I am very grateful to The Alan Turing Institute for their funding and the opportunities and facilities they have afforded me. I would also like to thank Iain Styles, the Baskerville team, and the University of Birmingham more widely for their assistance with compute requests.

Last, but certainly not least, I would like to thank all of my family and friends for their support during my seemingly unending journey through education. Particular thanks have to go to my wife Lyd who has been a listening ear throughout on so many things, not least my monologues on the motivations for derivative-based neural modelling of probability distributions, the complications of hazard-based modelling and the quirks of autodiff through ODE solves in various frameworks.

PUBLICATIONS

The following papers were published during the PhD (reverse chronological order):

1. Fabian Falck, Christopher Williams, **Dominic Danks**, George Deligiannidis, Christopher Yau, Christopher C. Holmes, Arnaud Doucet & Matthew Willetts, *A Multi-Resolution Framework for U-Nets with Applications to Hierarchical VAEs*, The Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS 2022)

2. **Dominic Danks** & Christopher Yau, *Derivative-based Neural Modelling of Cumulative Distribution Functions for Survival Analysis*, The 25th International Conference on Artificial Intelligence and Statistics (AISTATS 2022)

3. **Dominic Danks** & Christopher Yau, *BasisDeVAE: Interpretable Simultaneous Dimensionality Reduction and Feature-Level Clustering with Derivative-Based Variational Autoencoders*, Proceedings of the 38th International Conference on Machine Learning, PMLR 139:2410-2420 (ICML 2021)

4. Vasilis Stavrinides, Georgios Papageorgiou, **Dominic Danks**, Francesco Giganti, Nora Pashayan, Bruce Trock, Alex Freeman, Yipeng Hu, Hayley Whitaker, Clare Allen, Alex Kirkham, Shonit Punwani, Geoffrey Sonn, Dean Barratt, Mark Emberton & Caroline M. Moore, *Mapping PSA density to outcome of MRI-based active surveillance for prostate cancer through joint longitudinal-survival models*, Prostate Cancer and Prostatic Diseases (2021)

Chapters 2 and 3 of this thesis are based on papers 2 and 3 respectively. Paper 4 represents an example application of joint longitudinal-survival models which are a key topic of Chapter 4. Paper 1 considers the theoretical underpinning of Hierarchical VAEs, an extension of the Variational Autoencoder which is a core topic of Chapter 2.

Contents

Page

1 Introduction & Background **1**

1.1 Latent Variable Modelling 2

1.1.1 Motivations 2

1.1.2 Mathematical Formalism 4

1.1.3 Variational Inference 5

1.1.4 Variational Autoencoders 8

1.2 Ordinary Differential Equations (ODEs) 9

1.2.1 Fundamentals 10

1.2.2 Computational Aspects of ODEs 10

1.2.3 ODEs in Modern Machine Learning 20

1.3 Survival Analysis 24

1.3.1 Single-Risk Survival Analysis 25

1.3.2 Competing Risks Survival Analysis 26

1.3.3 Cox Proportional Hazards 27

1.3.4 Joint Longitudinal-Survival Modelling 28

1.3.5 Metrics 30

2 Dimensionality Reduction and Feature-Level Clustering via Derivative-Based VAEs **33**

2.1 Introduction 34

2.1.1 Extensions of VAEs . . . 34
2.1.2 Contribution . . . 37
2.1.3 Background: BasisVAE . . . 38
2.2 Derivative-Based VAE (DeVAE) . . . 40
2.3 Basis Derivative-Based VAE (BasisDeVAE) . . . 42
2.3.1 BasisDeVAE: Example . . . 42
2.4 Experiments . . . 46
2.4.1 Synthetic data . . . 46
2.4.2 OASIS . . . 48
2.4.3 Single-cell expression analysis . . . 51
2.5 Discussion . . . 55

3 Derivative-Based Neural Modelling of CDFs for Survival Analysis 57

3.1 Introduction . . . 58
3.1.1 Background . . . 58
3.1.2 Motivation . . . 59
3.1.3 Contribution . . . 60
3.1.4 Related Work . . . 61
3.2 Neural CDF Modelling: DeCDF . . . 63
3.2.1 Problem Setup . . . 63
3.2.2 DeCDF . . . 63
3.2.3 Distribution Estimation From Data . . . 65
3.2.4 CDF & PDF Estimation From Data via DeCDF: Example . . . 65
3.2.5 Normalising Flow Viewpoint . . . 66
3.2.6 Alternative Derivative-Based Parameterisations . . . 67
3.3 DeCDF for Single-Risk Survival Analysis . . . 68
3.3.1 Problem Setting . . . 68

3.3.2 Applying DeCDF . . . 69
3.4 Extending to Competing Risks: DeSurv . . . 70
3.4.1 Problem Setting . . . 70
3.4.2 DeSurv . . . 70
3.5 Experiments . . . 72
3.5.1 Reproducibility . . . 72
3.5.2 Single-Risk Survival Analysis . . . 72
3.5.3 Competing Risks Survival Analysis . . . 74
3.5.4 Results Discussion . . . 77
3.6 Discussion . . . 78

4 Neural ODE-Based Longitudinal Survival Analysis 81
4.1 Introduction . . . 82
4.1.1 Motivation . . . 82
4.1.2 Background . . . 83
4.1.3 Contributions . . . 84
4.2 Hazard-Based Modelling in the Longitudinal Setting . . . 85
4.3 Extending the Canonical Joint Longitudinal-Survival Model . . . 87
4.3.1 Derivative-Based Neural Joint Longitudinal-Survival Modelling . . . 87
4.3.2 Flexible Latent Neural ODE-based Joint Longitudinal-Survival Modelling 89
4.3.3 Discriminative Longitudinal Survival Analysis . . . 91
4.4 Experiments . . . 94
4.4.1 Datasets . . . 94
4.4.2 Joint Longitudinal-Survival Model Experiments . . . 98
4.4.3 Discriminative Model Experiments . . . 100
4.5 Discussion . . . 103

5 Conclusion 104

5.1 Summary 104
5.2 Future Directions 105

A Appendix to Chapter 2 **108**
A.1 OASIS features 108

B Appendix to Chapter 3 **109**
B.1 Experimental Details 109
B.1.1 Dataset Preprocessing 109
B.1.2 Model Hyperparameters 110
B.2 Additional Synthetic Data Experiment 110
B.3 Example Survival Functions and CDFs 111

List of Figures

1.1 **Diagrammatic representation of the Latent Neural ODE model (Rubanova et al., 2019).** 23

2.1 **Simultaneous dimensionality reduction and feature-level clustering.** Given an observed data matrix $X \in \mathbb{R}^{N \times d}$, we would like to i) learn a low-dimensional representation $\mathbf{z}$ (dimensionality reduction) and ii) cluster features according to their behaviour as a function of $\mathbf{z}$ (feature-level clustering). The standard VAE framework allows us to perform only i). With our proposed BasisDeVAE model we can perform both tasks simultaneously and guarantee *interpretability* of the inferred cluster assignments. 36

2.2 **Modelling biological progression.** If the dominant source of variation in a cross-sectional dataset is associated with biological or disease progression, we can use a VAE to encode and map the samples onto a one-dimensional latent space (pseudotime) and decode to understand the feature-level variability with pseudotime. 37

2.3 **Feature behaviours.** Schematic representations of the monotonically increasing, monotonically decreasing and transient feature behaviours often present within biological data that BasisDeVAE aims to capture. 40

2.4 **Example $G_{t_0}(t)$ transients.** Left: $h_{t_0}(t)$ of the form (2.9) with $f(\tau)$ a random feature-dependent constant. Middle: $h_{t_0}(t)$ of the form (2.9) with $f(\tau)$ a Xavier-initialised neural network. Right: Same functional form as the middle panel, but after training on the Ernst et al. (2019) data via BasisDeVAE. . . 44

2.5 **Synthetic data.** BasisVAE and BasisDeVAE are applied to synthetic data with the ground-truth trajectories and clusterings shown in the upper-left panel. BasisDeVAE infers both feature behaviour and cluster assignments correctly (bottom left), whereas BasisVAE learns degenerate basis functions (bottom right), frustrating clustering performance (top right). 47

2.6 **OASIS regional brain volumes.** Inferred pseudotemporal profiles of 4 regional brain volumes. DeVAE extracts the monotonically decreasing pseudotemporal trajectories associated with cognitive decline. 49

2.7 **Dependence of BasisVAE cluster predictions on translation parameter δ.** Variation of the appearance of BasisVAE1 (top) and BasisVAE2 (bottom) components with $\delta = 1.5$ (solid lines) and $\delta = -1.5$ (dashed). If the clustering were optimal, each plot would display qualitative similarity between same-colour lines and distinct behaviour between different-colour lines. This is not seen, supporting the observations made during synthetic data experiments that BasisVAE's basis functions can i) demonstrate multiple qualitatively different behaviours within a single basis function and ii) collapse such that each cluster's basis function has a very similar appearance. 52

2.8 **Single-cell spermatogenesis data.** Inferred clustering and pseudotemporal gene expression trajectories of eight genes from Ernst et al. (2019) with BasisVAE1, BasisVAE2 and BasisDeVAE. BasisDeVAE provides a superior fit to observed data as well as greater cluster interpretability. 53

3.1 **Competing Risks Survival Analysis (CRSA).** Given a patient's covariates $\mathbf{x}^{(i)}$, CRSA provides a personalised risk profile for each disease event, for example death due to cancer, cardiovascular disease (CVD) or neurological disease (ND). 59

3.2 **Modelling Distributions with DeCDF.** DeCDF allows flexible modelling of general time-to-event distributions. Left: Example CDFs and corresponding PDFs modelled by DeCDF. Right: An example of DeCDF learning to model a rich distribution from data, in this case a Mixture of Gaussians (MoG) distribution. 64

3.3 **DeSurv Model Architecture.** DeSurv flexibly models the cumulative incidence function of each event by performing elementwise multiplication between the outputs of a DeCDF block and a softmax neural network block. In the absence of competing risks, only the DeCDF block is required. 71

3.4 **METABRIC Survival Functions.** 25-year survival trajectories for five random test patients according to three of the considered models. Note that whilst the models agree on patient risk ranking, DeepHit has limited trajectory separation. 78

3.5 **SEER CIFs.** 25-year cumulative incidence functions for five random test patients according to DeepHit and DeSurv. As in the single-risk case (Figure 3.4), whilst the models agree on patient risk ranking DeepHit has limited trajectory separation. 79

4.1 **Longitudinal Survival Analysis.** Longitudinal survival analysis aims to understand the relationship between time-to-event and time-varying covariates $x(t)$ in the presence of repeated time-labelled measurements. 84

4.2 **Joint DeSurv Model Diagram.** A pictorial representation of the model we refer to as Joint DeSurv (Section 4.3.1). 88

4.3 **NODESurv Model Diagram.** A pictorial representation of the model we refer to as NODESurv (Section 4.3.2). 90

4.4 **D-NODESurv Model Diagram.** A pictorial representation of the model we refer to as D-NODESurv (Section 4.3.3). 92

4.5 **D-NODESurv Predictions**. D-NODESurv predictions for two test datapoints from Synthetic Dataset D. D-NODESurv provides an estimate of the conditional CDF $P(T \leq t|T > t_n, \mathcal{X}_{i,t\leq t_n})$ and associated PDF for each observation time t_n. 100

B.1 **METABRIC Survival Functions.** 25-year survival trajectories for five random test patients according to four of the considered models. (Appendix B.3) 112

B.2 **SUPPORT Survival Functions.** Five-year survival trajectories for five random test patients according to four of the considered models. (Appendix B.3) 113

B.3 **Synthetic Cumulative Incidence Functions.** Cumulative incidence functions for five random test subjects according to DeepHit and DeSurv. (Appendix B.3) 114

B.4 **SEER Cumulative Incidence Functions.** 25-year cumulative incidence functions for five random test patients according to DeepHit and DeSurv. (Appendix B.3) 115

List of Tables

2.1 **Synthetic data clustering performance.** Adjusted Rand Index (ARI) values associated with the clusterings learnt from the "low z" ($\mathcal{C}_{\text{l}}$) and "high z" ($\mathcal{C}_{\text{h}}$) datasets (see Section 2.4.1 text). GT represents the ground-truth clustering. 48

2.2 **OASIS performance metrics.** Test set predictive log-likelihood and ELBO values for the OASIS-3 data. 50

2.3 **Single-cell data performance metrics.** Training time and model fit metrics for the simultaneous dimensionality reduction and feature-level clustering methods applied to the Ernst et al. (2019) scRNA-seq data. 54

3.1 **Survival Datasets.** Details of the datasets used to demonstrate model performance. 73

3.2 **METABRIC Performance Metrics.** 74

3.3 **SUPPORT Performance Metrics.** 75

3.4 **Competing Risks Performance Metrics.** 76

3.5 **Additional Competing Risks Metrics.** 76

4.1 **Joint Model Synthetic Data Performance.** Metric values representing the performance of each of the joint longitudinal-survival models considered in Section 4.3 on synthetic datasets A, B & C as defined in Section 4.4.1. The metrics considered are defined in Section 4.4.2. 99

4.2 **Discriminative Model Performance on Synthetic and Real-World Data.** 102

B.1 **Effect of Increased Number of Discretisation Points on Synthetic Data DeepHit Performance.** 111

Chapter One

Introduction & Background

Due to the focus of this thesis being on the development of machine learning methodology for a variety of health-based problem settings, a variety of background content is required to fully contextualise the core novel content in Chapters 2, 3 & 4. This chapter provides this background content, in turn facilitating the reader's understanding of later chapters and reducing the need for repetition of fundamental content within such chapters. It also attempts to motivate the development of the methodology presented in later chapters.

This introductory chapter is split into three parts, each of which introduces one of the three core themes around which the novel contributions of the thesis are based, namely Latent Variable Modelling, Ordinary Differential Equations (ODEs), and Survival Analysis. While this chapter does not contain novel contributions per se, efforts have been made to produce a concise yet didactic introduction which is hopefully of value to those interested in the themes in question or looking to pursue research directions in the area.

1.1 Latent Variable Modelling

Latent Variable modelling is a core theme of Chapters 2 & 4, within which new latent variable-based models are introduced (BasisDeVAE and NODESurv). This section first introduces the motivations for latent variable models (LVMs) at a high-level (Section 1.1.1). It then moves on to define such models in a more formal mathematical way (Section 1.1.2). Following this (Section 1.1.3), the notion of Variational Inference (the technique used for performing inference for Latent Variable Models utilised within future chapters) is introduced and explained. The chapter is then concluded (Section 1.1.4) by introducing a particular type of LVM — the Variational Autoencoder (of which BasisDeVAE is an extension). This acts as an example of a modern realisation of an LVM and serves as a valuable grounding when approaching Chapter 2.

1.1.1 Motivations

Many forms of data are inherently high-dimensional. For example, it is not unusual for single-cell RNA Sequencing (scRNA-seq) experiments to target multiple thousands of genes ($g \approx 5000$). Whilst data in this raw format contains information and is often useful in a computational context, it is difficult for humans to interpret. More concretely, it is difficult for a human to look at a spreadsheet of such data, which may be a table of size 10000 (cells) $\times$ 5000 (genes), and infer from that, say, which cells have a similar expression pattern. Latent variable models exploit the fact that this high-dimensional data can often be modelled as arising from a structured process which may be lower-dimensional or more interpretable.

An early example of such an approach can be found in Spearman (1904), where (potentially high-dimensional) test score data is modelled as resulting from a transformed

low-dimensional latent "intelligence" variable. Extensions of this idea are the basis of psychometric tests still in use today — high-dimensional data from detailed questionnaires is projected to a low-dimensional latent variable argued to encode various forms of intelligence or personality traits. This provides those interested in the subject with a low-dimensional vector characterising their responses, rather than raw individual questions and answers. It is important to note that the extent to which such latent variables actually describe what we mean by intelligence or personality is hotly debated. This highlights a crucial but often overlooked aspect of latent variable modeling, namely that the role of a statistical latent variable model is to extract latent variables which explain the variation of the observed data in a way consistent with the modelling assumptions. It is not the role of the latent variable model to assign meaning beyond that to such latent variables. Concretely, in the case of psychometric tests, it is not the validity of the latent variable model which is questionable — it is true that there exist latent variables which capture the variation and correlation in test scores very well — it is rather questionable that those latent variables are what we really mean by "personality" or "intelligence". Such ambiguities can be reduced by appropriate constraints within the latent variable model and with careful use of language when interpreting the latent variables, which we endeavour to do in this thesis.

However, latent variable models can be extremely useful even if their latent variables are not necessarily semantically meaningful. For example, one may encounter a setting with very high-dimensional data and simply wish to represent such data in a low-dimensional latent space, in turn making the modelling problem at hand less computationally demanding. It should also be noted that whilst the latent variables argued to correspond to personality or intelligence in psychometric tests are typically continuous, discrete latent variables are also often valuable. For example, returning to the scRNA-seq setting, it is often useful to cluster cells demonstrating similar expression behaviours. This can be done by modelling each cell as being associated with a discrete latent variable, with the value of that latent

variable denoting the cluster most suited to that cell's observations. The net result of running a suitable latent variable model is therefore the assignment of each cell to a cluster, with each cell in that cluster demonstrating similar expression patterns. Such approaches have been known to, for example, uncover cell subtypes which respond heterogeneously to cancer therapies (Hu et al., 2020).

1.1.2 Mathematical Formalism

Further insight into latent variable models can be gained by viewing them from a mathematical perspective. For the sake of brevity and focus we restrict our attention to the form of LVMs relevant in later chapters. These are LVMs of the form

$$\mathbf{z}_i \sim p(\mathbf{z}_i), \ i = 1, \ldots, n$$
$$\mathbf{x}_i|\mathbf{z}_i \sim p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_i).$$

Intuitively, this model posits that each observation $\mathbf{x}_i$ is obtained by first sampling the latent variable $\mathbf{z}_i$ associated with that observation from the prior $p(\mathbf{z})$, and then sampling $\mathbf{x}_i$ from the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_i)$, parameterised by $\boldsymbol{\theta}$.

Two instructive examples of such models are Probabilistic Principal Component Analysis (Tipping and Bishop, 1999) and the Mixture of Gaussians model, presented in turn below.

Example: Probabilistic Principal Component Analysis (PPCA)

In PPCA, $\mathbf{z} \in \mathbb{R}^p$ and the generative model for $\mathbf{x} \in \mathbb{R}^D$ $(p < D)$ is

$$\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_p), \ i = 1, \ldots, n$$
$$\mathbf{x}_i|\mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i, \sigma^2\mathbf{I}_D).$$

In this context, $\boldsymbol{\theta} = (\mathbf{W}, \sigma^2)$.

Example: Mixture of Gaussians (MoG)

In a MoG model, $\mathbf{z} \in \{0, 1, \ldots, K\}$ and the generative model for $\mathbf{x} \in \mathbb{R}^D$ is

$$\begin{aligned} \mathbf{z}_i &\sim \text{Categorical}(\boldsymbol{\pi}), \ i = 1, \ldots, n \\ \mathbf{x}_i | \mathbf{z}_i &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}_i}, \boldsymbol{\Sigma}_{\mathbf{z}_i}). \end{aligned}$$

In this context, $\boldsymbol{\theta} = \{(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \ \forall j \in \{0, 1, \ldots, K\}\}$.

1.1.3 Variational Inference

In order to apply LVMs in practice, we must also provide an inference scheme, that is how estimates for parameters $\boldsymbol{\theta}$ and potentially the (distributions over) the latent variables themselves can be calculated given data. Concretely, given data $\mathcal{X} = \{\mathbf{x}_i\}_{i=1,\ldots,N}$, we would like to be able to derive an appropriate estimate for $\boldsymbol{\theta}$ given $\mathcal{X}$. We may also like to target the latent variables $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1,\ldots,N}$ given $\mathcal{X}$.

A purist Bayesian approach would be to target the posterior distribution

$$p(\boldsymbol{\theta}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{X})}, \tag{1.1}$$

or indeed

$$p(\boldsymbol{\theta}, \mathcal{Z}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{\theta}, \mathcal{Z})p(\boldsymbol{\theta}, \mathcal{Z})}{p(\mathcal{X})}, \tag{1.2}$$

however this is generally challenging in the machine learning setting as the integrals

$$\begin{aligned} p(\mathcal{X}|\boldsymbol{\theta}) &= \int p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta}) \, \mathrm{d}\mathcal{Z} \\ p(\mathcal{X}) &= \int p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \end{aligned}$$

are intractable. One can partially circumvent this by performing MCMC on (1.2), however this does not scale well for large dataset size or parameter dimension.

In machine learning settings, where $\dim(\boldsymbol{\theta}), \dim(\mathcal{X})$ and $\dim(\mathcal{Z})$ are typically large, it is common to accept a point estimate of $\boldsymbol{\theta}$ in return for an expressive model and efficient computation. In simple settings, such as PPCA, the log-likelihood

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) = \log p(\mathcal{X}|\boldsymbol{\theta})$$

can be computed in closed form, allowing $\mathcal{L}(\boldsymbol{\theta}; \mathcal{X})$ to be maximised using e.g. gradient ascent. However, due to $p(\mathcal{X}|\boldsymbol{\theta})$ being intractable in typical ML-based models, variational methods, which avoid the need to compute this integral, are typically employed. To derive such methods, consider the log-likelihood to be maximised with respect to $\boldsymbol{\theta}$, i.e.

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) = \log \int p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta}) \, \mathrm{d}\mathcal{Z},$$

and insert an identity factor $q(\mathcal{Z})/q(\mathcal{Z})$, where $q(\mathcal{Z})$ is a distribution over $\mathcal{Z}$:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) &= \log \int p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta}) \frac{q(\mathcal{Z})}{q(\mathcal{Z})} \, \mathrm{d}\mathcal{Z} \\ &= \log \mathbb{E}_q \left[\frac{p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta})}{q(\mathcal{Z})} \right]. \end{aligned}$$

Applying Jensen's inequality to the above expression yields

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) \geq \underbrace{\mathbb{E}_q \left[\log \frac{p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta})}{q(\mathcal{Z})} \right]}_{\coloneqq \tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})}.$$

We have therefore shown that there exists an objective function

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X}) \coloneqq \mathbb{E}_q \left[\log \frac{p(\mathcal{X}, \mathcal{Z}|\boldsymbol{\theta})}{q(\mathcal{Z})} \right]$$

which is a lower bound on the log-likelihood. $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ is often referred to as the *Evidence Lower Bound* (ELBO), a term that is derived from applying similar ideas in the context of model selection, where one attempts to maximise the evidence.

Note that by writing $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ as

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X}) = \mathcal{L}(\boldsymbol{\theta}; \mathcal{X}) - \mathrm{KL}\left[q(\mathcal{Z}) \| p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta})\right],$$

one can see that if $q(\mathcal{Z})$ is set to the posterior $p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta})$, then the bound is tight, i.e.

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta}); \mathcal{X}) = \mathcal{L}(\boldsymbol{\theta}; \mathcal{X}).$$

It is natural to ask why $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ is a useful concept. In short, it is because rather than attempting to optimise $\mathcal{L}(\boldsymbol{\theta}, q; \mathcal{X})$ with respect to $\boldsymbol{\theta}$ directly, which is in general difficult, one can instead either alternatingly or jointly optimise $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ with respect to both $\boldsymbol{\theta}$ and q, which is typically easier.

For example, in some models, computing $p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta}_0)$ and minimising $\tilde{\mathcal{L}}(\boldsymbol{\theta}, p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta}_0); \mathcal{X})$ with respect to $\boldsymbol{\theta}$ is straightforward — in the MoG model it can be done in closed form. Hence the full likelihood $\mathcal{L}(\boldsymbol{\theta}, q; \mathcal{X})$ can be (locally) maximised by alternating optimisation with respect to q and $\boldsymbol{\theta}$. In practice, this becomes repetition of the following two step iteration process: Let $\boldsymbol{\theta}_{\text{est}}$ be the current estimate of the maximum point of $\boldsymbol{\theta}$. In step 1, define $u(\boldsymbol{\theta}, \boldsymbol{\theta}_{\text{est}}) = \tilde{\mathcal{L}}(\boldsymbol{\theta}, p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta}_{\text{est}}); \mathcal{X})$ — this is akin to optimisation over q. In step 2, set $\boldsymbol{\theta}_{\text{est}} = \arg\max_{\boldsymbol{\theta}} u(\boldsymbol{\theta}, \boldsymbol{\theta}_{\text{est}})$ — this is optimisation over $\boldsymbol{\theta}$. This approach is known as Expectation-Maximisation (EM) (Dempster et al., 1977), with step 1 known the E-step, as it involves taking expectations, and step 2 the M-step, as it involves explicit maximisation.

In complex latent variable models, closed form EM updates may not be possible as the posterior $p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta}); \mathcal{X})$ may be intractable. In particular, standard EM is typically used for LVMs involving exponential family distributions, e.g. PPCA, Factor Analysis, MoG, Hidden Markov Models and Kalman Filters. However, the notion of $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ being a useful proxy for $\mathcal{L}(\boldsymbol{\theta}; \mathcal{X})$ remains. Specifically, if one is faced with a model where the exact posterior $p(\mathcal{Z}|\mathcal{X}, \boldsymbol{\theta})$ is intractable, one can introduce an *approximate variational posterior* with parameters ϕ, i.e. $q_\phi(\mathcal{Z})$. $\tilde{\mathcal{L}}(\boldsymbol{\theta}, q; \mathcal{X})$ can then be treated as $\tilde{\mathcal{L}}(\boldsymbol{\theta}, \phi; \mathcal{X})$. An approximate

form of EM can then be applied — in the E-step, one optimises $\tilde{\mathcal{L}}$ with respect to ϕ keeping $\boldsymbol{\theta}$ fixed and in the M-step one optimises $\tilde{\mathcal{L}}$ with respect to $\boldsymbol{\theta}$ keeping ϕ fixed. One can also abstract away from the original EM approach and simply perform gradient ascent on $\tilde{\mathcal{L}}(\boldsymbol{\theta}, \phi; \mathcal{X})$ jointly.

This concept of optimising a lower bound of the log-likelihood via approximate variational posteriors is at the heart of inference in modern machine learning-based latent variable models. In the next section, we demonstrate how this abstract concept can be applied to learn a practically useful machine learning-based latent variable model, namely the variational autoencoder, which is the basis of the novel contributions of Chapter 2.

1.1.4 Variational Autoencoders

The original VAE framework (Kingma and Welling, 2014) is constructed from a latent variable model with the generative process for observation $\mathbf{x}_i$ given by

$$\mathbf{z}_i \sim p(\mathbf{z}_i),$$
$$\mathbf{x}_i|\mathbf{z}_i \sim p_{\tilde{\theta}}(\mathbf{x}_i|\mathbf{z}_i).$$

Here $\mathbf{z}_i \in \mathbb{R}^p$ is the latent variable corresponding to the data point $\mathbf{x}_i \in \mathbb{R}^d$ in the dataset $\mathcal{X} = \{\mathbf{x}_i\}_{i=1,\ldots,N}$. Draws are assumed independent and identically distributed (i.i.d.) over data points $i = 1, \ldots, N$. The conditional likelihood $p_{\tilde{\theta}}(\mathbf{x}_i|\mathbf{z}_i)$ is modelled via a deep neural network f_θ (the *decoder*), typically such that $\mathbb{E}[\mathbf{x}_i|\mathbf{z}_i] = f_\theta(\mathbf{z}_i)$. For example, in the case of a Gaussian conditional likelihood, $p_{\tilde{\theta}}(x_j|\mathbf{z}_i) = \mathcal{N}\left(x_j|f_\theta^{(j)}(\mathbf{z}_i), \sigma_j^2\right)$, with j denoting output dimension, θ representing neural network parameters and $\tilde{\theta} = \left(\theta, \{\sigma_j^2\}\right)$. The prior $p(\mathbf{z}_i)$ can be chosen to match the problem of interest but is generally restricted to distributions which are easy to sample from. In this thesis, we set $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_p)$ throughout.

Parameter inference for $\tilde{\theta}$ is based on maximum likelihood estimation, however the

log-likelihood

$$\sum_{i=1}^{N} \log p_{\tilde{\theta}}(\mathbf{x}_i) = \sum_{i=1}^{N} \log \int p_{\tilde{\theta}}(\mathbf{x}_i|\mathbf{z}_i) p(\mathbf{z}_i) \mathrm{d}\mathbf{z}_i \tag{1.3}$$

is in general intractable and cannot be optimised directly, hence an approximate variational posterior is introduced as described in Section 1.1.3. In the VAE setting, *amortised* variational inference is performed, meaning that the posterior's distributional parameters (e.g. mean and covariance for a Gaussian) are computed directly from $\mathbf{x}_i$ by a neural network mapping $h_\phi(\mathbf{x}_i)$ referred to as the *encoder*. We denote this variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$. In this context, the ELBO takes the form

$$\mathrm{ELBO}(\tilde{\theta}, \phi; \mathcal{X}) = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{\mathbf{z}_i \sim q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{x}_i)} \left[\log p_{\tilde{\theta}}(\mathbf{x}_i|\mathbf{z}_i)\right] - \sum_{i=1}^{N} \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{x}_i) \| p(\mathbf{z}_i)\right]. \tag{1.4}$$

Inference in the VAE framework therefore reduces to minimisation of the loss

$$\mathcal{L}(\tilde{\theta}, \phi; \mathcal{X}) = -\mathrm{ELBO}(\tilde{\theta}, \phi; \mathcal{X}),$$

with respect to the decoder parameters $\tilde{\theta}$ and encoder parameters ϕ. This minimisation is typically performed jointly over $(\tilde{\theta}, \phi)$ using variants of stochastic gradient descent and the reparameterisation trick (Kingma and Welling, 2014; Rezende et al., 2014) with a Gaussian variational posterior such that $q_\phi\left(\mathbf{z}_i|\mathbf{x}_i\right) = \mathcal{N}\left(\mathbf{z}_i|\mu_\phi\left(\mathbf{x}_i\right), \sigma^2_\phi\left(\mathbf{x}_i\right)\right)$ in order to facilitate the calculation of ϕ gradients.

1.2 Ordinary Differential Equations (ODEs)

The DeSurv and NODESurv time-to-event models introduced in Chapters 3 and 4 respectively involve the use of Ordinary Differential Equations (ODEs). This section provides the reader with the background content underlying the novel contributions of that methodology. In particular, it covers what an ODE is (Section 1.2.1), how they are managed in computational settings including how ODE parameters can be estimated (Section 1.2.2), and research

directions in machine learning involving ODEs and how our contributions relate to these (Section 1.2.3).

1.2.1 Fundamentals

ODE-based models represent the dynamics of a variable of interest $\mathbf{x}(t)$ via specification of its derivatives. More precisely, an ODE of order n expresses the n^{th} derivative $\mathbf{x}^{(n)}$ of a state vector $\mathbf{x}(t)$ with respect to an independent variable t in terms of $\mathbf{x}, t$, derivatives $\mathbf{x}^{(m)}$ with $m < n$, and possibly parameters $\boldsymbol{\theta}$. That is, it expresses that $\mathbf{x}^{(n)}(t) = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(n-1)}, \ldots, \mathbf{x}^{(1)}, \mathbf{x}, t)$, where $\mathbf{f}_{\boldsymbol{\theta}}$ is a vector-valued function parameterised by $\boldsymbol{\theta}$. As one can always rewrite an n^{th}-order ODE as n coupled first-order ODEs, attention can be restricted to first-order ODEs of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) \tag{1.5}$$

without loss of generality, as is done in the remainder of this thesis. In order to ensure a unique solution to (1.5), an initial condition of the form $\mathbf{x}(t_0) = \mathbf{x}_0$ is required. Because ODEs explicitly model the derivative of a function, rather than its actual value, an additional "ODE solving" step is required when using such models in practice. This additional step is an important consideration and can add significant complexity to the modelling process.

1.2.2 Computational Aspects of ODEs

ODE Solvers

Generating numerical solutions to ODE equations of the form of (1.5) is the role of computational ODE solvers. These are routines which take as input a derivative function $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$

corresponding to an ODE system

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t),$$

along with an initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$ and time t_{eval}, and output an estimate of $\mathbf{x}_{\boldsymbol{\theta}}(t_{\text{eval}}; \mathbf{x}_0)$, denoted $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_{\text{eval}}; \mathbf{x}_0)$. Comprehensive treatments of ODE solvers can be found in e.g. Butcher (2016) or Shampine (2018). When considering ODE solvers in the context of ML models, it is sometimes useful to refer to the notion of an ODE solver abstractly by defining the ODESolve operation. In the context of how an ODE solver was described above, this operation is defined such that

$$\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_{\text{eval}}; \mathbf{x}_0) = \text{ODESolve}(\mathbf{f}_{\boldsymbol{\theta}}, \mathbf{x}_0, t_0, t_{\text{eval}}).$$

The simplest example of such a solver is the Euler method. This separates the interval $[t_0, t_{\text{eval}}]$ into n subintervals $\{[t_i, t_{i+1}] \mid i \in \{0, 1, 2, \ldots, n-1\}\}$ with $t_{i=0} = t_0$ and $t_{i=n} = t_{\text{eval}}$, and the solution estimate is computed using the recurrence relation

$$\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_{i+1}; \mathbf{x}_0) = \hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_i; \mathbf{x}_0) + (t_{i+1} - t_i)\mathbf{f}_{\boldsymbol{\theta}}\left(\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_i; \mathbf{x}_0), t_i\right).$$

The Euler method is a first-order solver, meaning a $\mathcal{O}(h^2)$ error is introduced with each step of size h and an $\mathcal{O}(h)$ error is introduced over multiple iterations, however only one function evaluation is required per step. A wide array of more sophisticated ODE solvers exist. Many of these achieve much-improved error bounds, e.g. $\mathcal{O}(h^4)$ for fourth-order Runge-Kutta, at the cost of multiple function evaluations per step. Additional accuracy can often be achieved via adaptive step size, that is varying the size of interval $[t_i, t_{i+1}]$ as a function of t depending on the complexity of the function around t. The most commonly used high-order, adaptive solvers are typically adaptive high-order Runge-Kutta-based approaches, particularly the Dormand-Prince-Shampine method of order 5.

ODE Parameter Estimation

The application of ODE-based models in ML settings typically requires parameter estimation, that is to infer the most appropriate value of or distribution over $\boldsymbol{\theta}$ given data believed to be consistent with the ODE model in question. More formally, given data $D = \{(t_i, \mathbf{y}_i)\}_{i=1,\dots,N}$, where $\mathbf{y}_i$ are noisy $\mathbf{x}(t_i)$ observations for a known underlying $\mathbf{f}_{\boldsymbol{\theta}}$ and $\mathbf{x}_0$, we would like to infer the $\boldsymbol{\theta}$ most consistent with the observations and any prior knowledge. As with any form of parameter estimation, there are two distinct approaches that can be taken, namely point estimation and Bayesian estimation. Point estimation targets a single parameter value which is optimal in some sense, and any parameter uncertainty estimation either occurs as a by-product of the chosen technique (e.g. covariance matrix approximation from Levenberg-Marquardt optimisation, see e.g. Press et al. (2007)) or is performed via post-hoc bootstrapping (see e.g. Chernick et al. (2011)). Bayesian estimation, instead targets the posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, and therefore inherently quantifies parameter uncertainty. In this thesis, we are generally interested in the application of ODEs to machine learning models where $\boldsymbol{\theta}$ is high-dimensional and sample sizes are large. Full Bayesian estimation methods struggle to scale to these settings and are hence largely avoided in the context of ODE-based machine learning. Accordingly, we focus on point estimation in this thesis.

If one approaches ODE parameter estimation from a point estimation viewpoint, then the parameter estimation task becomes an optimisation problem, with the desired point estimate of the parameters, $\boldsymbol{\theta}^*$, arising as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \mathcal{D}),$$

where $L(\boldsymbol{\theta}; \mathcal{D})$ is a suitable loss function measuring the discrepancy between observed data $\mathcal{D}$ and corresponding model predictions with parameter setting $\boldsymbol{\theta}$. The approach which immediately presents itself in this setting is that of explicitly solving a model's ODE system and comparing the resulting trajectory with available trajectory observations. For example in

the case of $L(\boldsymbol{\theta}; \mathcal{D})$ being the squared loss, for N noise-corrupted state variable observations, $\mathbf{Y} \in \mathbb{R}^{N \times d}$, made at times $\{t_i\}_{i=1,\dots,N}$ we have

$$L_{\text{Sq}}(\mathbf{Y}, \hat{\mathbf{Y}}^{(\boldsymbol{\theta})}) = \sum_{i=1}^{N} \left\| \mathbf{Y}_{i,\cdot} - \hat{\mathbf{Y}}^{(\boldsymbol{\theta})}_{i,\cdot} \right\|^2, \tag{1.6}$$

where $\hat{\mathbf{Y}}^{(\boldsymbol{\theta})} \in \mathbb{R}^{N \times d}$ represents the corresponding state variable predictions obtained by solving the ODE system with parameters $\boldsymbol{\theta}$. i.e. $\mathcal{E}_{\boldsymbol{\theta}}$. In standard machine learning settings, e.g. neural network regression, $\hat{\mathbf{Y}}^{(\boldsymbol{\theta})}$ can be computed in closed form and differentiated with respect to $\boldsymbol{\theta}$ via automatic differentiation, allowing the optimisation of L via gradient-based optimisation. The fact that in the ODE context the loss cannot generally be expressed in terms of simple analytic functions complicates the optimisation process as it means that computing the loss requires a (potentially computationally expensive) numerical integration routine to be called and that calculating the derivative of the loss for gradient-based optimisation is non-trivial. To convey the complexity of general calculation of the required gradients for optimisation in this setting, we pursue the example of attempting to minimise (1.6).

The core requirement for the application of ML-friendly gradient-based optimisation procedures such as stochastic gradient descent, Adam (Kingma and Ba, 2014) and RMSProp (Hinton et al., 2012) is the ability to compute $\frac{\partial L}{\partial \boldsymbol{\theta}}$, which for the above is of the form

$$\frac{\partial L_{\text{Sq}}}{\partial \boldsymbol{\theta}} = 2 \sum_{i=1}^{N} \mathbf{J}_i^{\mathsf{T}} \left(\hat{\mathbf{Y}}^{(\boldsymbol{\theta})}_{i,\cdot} - \mathbf{Y}_{i,\cdot} \right), \tag{1.7}$$

where $[\mathbf{J}_i]_{mn} = \frac{\partial \hat{Y}^{(\boldsymbol{\theta})}_{i,m}}{\partial \theta_n}$. Methods to compute this derivative in the ODE setting can be roughly split into three categories, namely finite differencing, continuous sensitivity analysis (CSA) and discrete sensitivity analysis (DSA). Whilst finite differencing is conceptually simple and easy to implement, its relative inaccuracy makes it unsuitable for this setting. CSA and DSA, however, are more effective.

CSA methods (forward-mode and adjoint CSA) compute the desired derivatives by solving additional differential equations which can be derived from the original ODE system.

Forward-mode CSA utilises the sensitivity/variational equations (Dickinson and Gelinas, 1976). Given a first order initial value problem

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{1.8}$$

these express the Jacobian $J_{ij} = \frac{\partial x_i}{\partial \theta_j}$, i.e. that appearing in (1.7), via the ODE

$$\frac{\mathrm{d}\mathbf{J}}{\mathrm{d}t} = \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \mathbf{x}}\mathbf{J}. \tag{1.9}$$

Note that these sensitivity equations (1.9) are straightforwardly derived by differentiating the ODE in (1.8) with respect to $\boldsymbol{\theta}$ and assuming symmetry of second derivatives. The appropriate initial conditions from which to solve (1.9) are

$$J_{ij} = 0 \quad \forall\, i, j, \tag{1.10}$$

as $\boldsymbol{\theta}$ does not influence the value of $\mathbf{x}$ at $t = t_0$. The solution of the ODE system defined by (1.9), (1.10) at time t_i is then the desired $\mathbf{J}_i$ present in (1.7).

Observe that the terms in (1.9) are in general dependent on $\mathbf{x}$ and t, hence when implementing forward-mode sensitivity analysis to compute (1.7), ODEs (1.8) and (1.9) are solved simultaneously by forming the $(d + dp)$-dimensional ODE for $[\mathbf{x}, \mathbf{J}]$. Forward-mode sensitivity analysis therefore involves one $(d+dp)$-dimensional ODE solve for each computation of the derivative $\frac{\partial L_{\mathrm{Sq}}}{\partial \boldsymbol{\theta}}$ if $\mathbf{x} \in \mathbb{R}^d, \boldsymbol{\theta} \in \mathbb{R}^p$. Furthermore, the accuracy with which this derivative is computed can be specified via the tolerance of the chosen ODE solver.

Whilst forward-mode CSA is used to calculate $\mathbf{J}$, which can then be fed into a formula of the form of (1.7) to compute the desired quantity $\frac{\partial L}{\partial \boldsymbol{\theta}}$, adjoint CSA can be used to compute $\frac{\partial L}{\partial \boldsymbol{\theta}}$ directly. For a typical loss of the form

$$L(\boldsymbol{\theta}; \mathcal{D}) = \sum_{i=1}^{N} \mathcal{L}\left(\mathbf{x}(t_i); \mathcal{D}_i\right), \tag{1.11}$$

i.e. a sum over observations i of discrepancies between observed data and ODE solution $\mathbf{x}(t)$,

adjoint CSA defines the adjoint

$$\mathbf{a}(t;t') = \frac{\mathrm{d}\mathcal{L}(\mathbf{x}(t');\mathcal{D}_{t'})}{\mathrm{d}\mathbf{x}(t)},$$

which should be read as the rate of change of the loss value at t' (evaluated with t' data) with respect to the value of the ODE solution at time t. Furthermore, $\boldsymbol{\theta}$ is treated as a (in reality constant) function of t so that

$$\mathbf{a}_{\boldsymbol{\theta}}(t;t') = \frac{\mathrm{d}\mathcal{L}(\mathbf{x}(t');\mathcal{D}_{t'})}{\mathrm{d}\boldsymbol{\theta}(t)}$$

can be defined. The motivation for defining such quantities is that it can be shown (see e.g. Chen et al. (2018b)) that they obey the ODE

$$\frac{\mathrm{d}\mathbf{a}_{\mathrm{aug}}(t;t')}{\mathrm{d}t} \coloneqq \begin{bmatrix} \frac{\mathrm{d}\mathbf{a}(t;t')}{\mathrm{d}t} \\ \frac{\mathrm{d}\mathbf{a}_{\boldsymbol{\theta}}(t;t')}{\mathrm{d}t} \end{bmatrix} = - \begin{bmatrix} \left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \mathbf{x}(t)}\right)^{\mathsf{T}} \mathbf{a}(t) \\ \left(\frac{\partial \mathbf{f}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(t)}\right)^{\mathsf{T}} \mathbf{a}(t) \end{bmatrix}, \tag{1.12}$$

and

$$\mathbf{a}_{\boldsymbol{\theta}}(t_0;t') = \frac{\partial \mathcal{L}(\mathbf{x}(t');\mathcal{D}_{t'})}{\partial \boldsymbol{\theta}}$$

represents the derivative we desire. Note that $\mathbf{a}(t';t')$ can be calculated analytically for the relevant loss function as it is just the derivative of the component loss function with respect to its main argument. Furthermore, $\mathbf{a}_{\boldsymbol{\theta}}(t';t') = \mathbf{0}$ as changing the ODE parameters at only $t = t'$ would not affect the ODE solution at $t = t'$. This means that our desired $\mathbf{a}_{\boldsymbol{\theta}}(t_0;t')$ can be calculated by solving the ODE (1.12) backwards in time from $t = t'$. Therefore, by solving (1.12) backwards in time from t_N to t_0 (t_i are labelled such that $t_0 \leq t_1 \leq t_2 \leq \cdots \leq t_N$), the values $\{\mathbf{a}_{\boldsymbol{\theta}}(t_0;t_i)\}_{i=1,\ldots,N}$ can be obtained and then summed to yield $\frac{\partial L}{\partial \boldsymbol{\theta}}$ as desired.

This procedure appears attractive as it seems to involve one $(d+p)$-dimensional ODE solve (c.f. one $(d+dp)$-dimensional solve for forward-mode CSA). However, this does not consider the fact that the computation of $\dot{\mathbf{a}}_{\mathrm{aug}}(t)$ in (1.12) in general requires $\mathbf{x}(t)$ values. One approach to address this would be to perform a forward solve of (1.8) every time an $\mathbf{x}(t)$ value was required by the reverse solver of (1.12), however the computational demand of this scheme

makes it totally infeasible. A suggestion to lessen this burden would be to first execute a single d-dimensional forward solve of (1.8) to obtain $\mathbf{x}(t_N)$, before running the backward solve on (1.8) alongside (1.12), i.e. solving a $(2d+p)$-dimensional ODE system. This is the scheme used by Chen et al. (2018b) to learn the parameters of neural ODEs, where it appears to work well. However, in general this method is unstable. This is because the reverse-solved $\mathbf{x}(t)$ will not in general follow the trajectory of the forward solve and can often drift significantly, leading to inaccurate adjoint computations and hence derivatives (Hindmarsh et al., 2005). This instability can be reduced at a cost of higher memory usage by adopting methods which use the forward solve to create an interpolation scheme which can be called upon for accurate computation of $\mathbf{x}(t)$ during the reverse solve (Hindmarsh et al., 2005; Wang et al., 2009). These methods therefore offer a compromise, aiming to combine the lower computational cost of adjoint CSA's lower dimensional ODE solve and the numerical stability of forward-mode CSA, and this approach is indeed formidable in the case of ODEs with many parameters (Rackauckas et al., 2018).

The process of computing the desired derivative using CSA methods is also referred to as the differentiate-then-discretise approach (Zhang and Sandu, 2014), as one first expresses the desired derivative via ODEs by differentiation and then attempts to solve these ODEs using a numerical solver with discrete steps. DSA methods instead adopt a discretise-then-differentiate approach by first representing the ODEs' solutions by their discrete approximation and then computing the derivative of this function. An apparent drawback of this approach is that a specialised implementation must be created for each numerical integration routine to calculate the relevant derivatives. However, this can be avoided by instead implementing each numerical integration routine in an automatic differentiation (AD) framework. In this setup, the computation of $L(\boldsymbol{\theta}; \mathcal{D})$ defines a computational graph upon which forward- or reverse-mode AD can be applied. Applying forward-mode AD corresponds to a forward-mode DSA approach of computing $\frac{\partial L}{\partial \boldsymbol{\theta}}$, whilst applying reverse-mode AD corresponds to an adjoint

DSA approach.

Rackauckas et al. (2018) provide a detailed comparison of the performance of the various forms of sensitivity analysis for loss function derivative calculation using the Julia language (Bezanson et al., 2017), with their main conclusion being that forward-mode DSA is most appropriate for small systems ($\lesssim 100$ parameters), whilst adjoint CSA (with the use of interpolation methods) is most appropriate for larger systems. This motivates the use of adjoint CSA in modern machine learning settings involving neural networks (Chen et al., 2018b).

Quadrature

As seen by the volume of content above, the process of parameter estimation in ODE systems is in general non-trivial and can be computationally expensive. If the system being modelled in a given problem is genuinely described by an ODE of the form (1.8), then there is little that can be done to avoid these considerations. However in some machine learning settings, we may instead be able to posit a model involving an ODE of the form

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}_{\boldsymbol{\theta}}(t), \quad \mathbf{x}(t_0) = \mathbf{x}_0,$$

i.e. where the derivative at time t is specified in terms of only t, not also via $\mathbf{x}(t)$. This form of ODE implies that $\mathbf{x}(t)$ can be computed via the integral

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{f}_{\boldsymbol{\theta}}(\tau)\,\mathrm{d}\tau$$

Importantly, it is generally much more computationally tractable to compute integrals of this form reliably and also to compute their $\boldsymbol{\theta}$ derivatives as one can employ *quadrature* methods rather than via full ODE solvers. Given an integral

$$A = \int_a^b f(x)\,\mathrm{d}x\,,$$

these methods compute the estimate of A as a weighted sum of n evaluations of f:

$$\hat{A} = \sum_{i=1}^{n} w_i f(x_i),$$

with the weights $\{w_i\}$ and evaluation points $\{x_i\}$ dependent on the form of quadrature chosen. Perhaps the simplest form of such a method is the trapezoidal rule. In this setting, the interval $[a, b]$ is split into $n-1$ (usually equally spaced) subintervals $\{[x_k, x_{k+1}]\}_{k=1}^{n-1}$ with $x_k = a + (k-1)\Delta x$, where $\Delta x = (b-a)/(n-1)$. The area under $f(x)$ over the interval $[x_k, x_{k+1}]$, i.e.

$$I_k = \int_{x_k}^{x_{k+1}} f(x)\,\mathrm{d}x\,,$$

is then approximated by the area under the linear interpolation of $f(x)$ over the same region, i.e.

$$I_k \approx \hat{I}_k = \Delta x \cdot \frac{f(x_k) + f(x_{k+1})}{2}.$$

The overall approximation of A is then taken as

$$\begin{aligned}\hat{A} &= \sum_{i=k}^{n-1} I_k = \sum_{k=1}^{n-1} \frac{\Delta x}{2}(f(x_k) + f(x_{k+1})) \\ &= \sum_{i=1}^{n} w_i f(x_i)\end{aligned}$$

with $w_i = 1/2$ for $i \in \{1, n\}$ and $w_i = 1$ for $i \in \{2, \ldots, n-1\}$, and $x_i = x_k$ as defined above.

The trapezoidal rule asymptotically approaches the true integral as $n \to \infty$ but typically requires large n for satisfactory performance. As computational complexity scales with n, we would like a method with good performance for relatively small n. This is offered by Gauss-Legendre quadrature. Gauss-Legendre quadrature applies to integrals of the form

$$I = \int_{-1}^{1} g(x)\mathrm{d}x.$$

The idea behind it is to approximate the integrand $g(x)$ with a high-order polynomial and output the integral of this polynomial as the estimate of the actual integral I. More precisely,

Gauss-Legendre quadrature of order n approximates I as

$$I \approx \hat{I} = \sum_{i=1}^{n} w_i g(x_i),$$

where w_i are weights computed using

$$w_i = \frac{2}{(1 - x_i^2)\left[P_n'(x_i)\right]^2},$$

with P_n the n^{th} Legendre polynomial with $P_n(1) = 1$ and x_i as the roots of the n^{th} Legendre polynomial. Gauss-Legendre quadrature of order n is exact for polynomials of order $2n - 1$ or less.

The standard way to numerically evaluate an integral of the form

$$I = \int_a^b h(x)\mathrm{d}x$$

using Gauss-Legendre quadrature is to first write it as

$$\int_{-1}^{1} g(x)\mathrm{d}x$$

with

$$g(x) = \left(\frac{b-a}{2}\right) h\left(\left(\frac{b-a}{2}\right) x + \left(\frac{a+b}{2}\right)\right)$$

and apply a Gauss-Legendre quadrature to the resulting integral

$$I = \int_{-1}^{1} g(x)\mathrm{d}x.$$

This is how integrals are computed in the models posited in the DeVAE, BasisDeVAE and DeSurv models of Chapters 2 & 3.

The true benefit of quadrature-based estimation of $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t; \mathbf{x}_0)$ over full ODE solves in the context of parameter estimation becomes clear when the following is realised: to calculate $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t; \mathbf{x}_0)$ via a full ODE solver, $\mathbf{x}_0$ is iteratively updated, and the value of $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_i; \mathbf{x}_0)$ is required to compute $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_{i+1}; \mathbf{x}_0)$, coupling computational steps and frustrating concurrency.

However, in quadrature, the estimate $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(t_{i+1}; \mathbf{x}_0)$ consists of n independent terms, each of which can be performed concurrently before being summed, allowing quadrature to be heavily parallelised. We capitalise on this aspect in our work by performing such computations on a GPU, significantly accelerating computations relative to full ODE solvers.

1.2.3 ODEs in Modern Machine Learning

Neural ODEs

In recent years, interest in ODE-based modelling has significantly increased within the machine learning community. Much of this interest was piqued by the introduction of Neural ODEs (Chen et al., 2018b). These are models which utilise neural networks to model the derivative function $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$. In their original form, Neural ODEs are a mapping $\mathbf{x} \mapsto \mathbf{y}$ via a neural network-based ODE. Specifically, $\mathbf{y} = \mathbf{z}(T)$, with

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}(t), t); \quad \mathbf{z}(0) = \mathbf{x} \tag{1.13}$$

and $\mathbf{f}_{\boldsymbol{\theta}}(\cdot)$ a neural network. Note that if $\mathbf{f}_{\boldsymbol{\theta}}$ is autonomous and an Euler solver with step size 1 is applied to this ODE, the computation of $\mathbf{y}$ is iterative, with a series of steps of the form

$$\mathbf{z}_{i+1} = \mathbf{z}_i + \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}_i),$$

which is a ResNet (He et al., 2015) update. Applying other solvers to the fundamental ODE (1.13) generates other known architecture formats (Lu et al., 2017), e.g. applying (approximate) backward Euler gives PolyNet (Zhang et al., 2016) and applying Runge-Kutta gives FractalNet (Larsson et al., 2016).

ODERNN

Neural ODEs are particularly useful in temporal settings as they are compatible with the concept of continuous-time modelling. A good example of this is within the ODERNN (Rubanova et al., 2019), utilised in Chapter 4. The ODERNN is an adaptation of the standard Recurrent Neural Network architecture which posits that the hidden state evolves according to a learnt neural ODE in the absence of observations. More formally, in a standard RNN, the value of the hidden state for time $t_i = t_{i-1} + \Delta_t$ at which an observation $\mathbf{x}_i$ is made is computed via

$$h_i = \text{RNNCell}\left(h_{i-1}, \Delta_t, \mathbf{x}_i\right),$$

with $h(t)$ undefined or considered constant for values of t not in the set of observed times $\{t_i\}_{i=1}^{n_{\text{obs}}}$. However, in an ODERNN, if the most recent observation was at t_{i-1}, $h(t)$ is computed via an ODE solve of $\dot{h} = \mathbf{f}_{\boldsymbol{\theta}}(h(t), t)$ from t_{i-1} to t, i.e.

$$h(t) = \text{ODESolve}\left(\mathbf{f}_{\boldsymbol{\theta}}, h_{i-1}, t_{i-1}, t\right),$$

where the ODESolve operator is as defined in Section 1.2.2. At observed values of t, i.e. $t \in \{t_i\}_{i=1}^{n_{\text{obs}}}$, the hidden state is updated by observed $\mathbf{x}_i$ via

$$h_i = \text{RNNCell}\left(h_i', \mathbf{x}_i\right),$$

with

$$h_i' = \text{ODESolve}\left(\mathbf{f}_{\boldsymbol{\theta}}, h_{i-1}, t_{i-1}, t_i\right).$$

This formalism improves the ability of RNNs to reliably output predictions for $t \notin \{t_i\}_{i=1}^{n_{\text{obs}}}$, proving useful in settings with irregularly sampled data.

Latent Neural ODEs

ODERNNs offer an immediate way to perform autoregressive modelling of data in a more continuous-time way than a standard RNN. However in some settings it may be of value to

take a latent variable-based approach rather than an autoregressive approach. The latent neural ODE model Rubanova et al. (2019) addresses this scenario.

The latent neural ODE model posits that there is a continuous latent variable $\mathbf{z}(t)$ which underlies time series data $\mathbf{x}(t)$, with the generative model for some observation $\mathbf{x}_n \coloneqq \mathbf{x}(t_n)$ taking the form

$$\begin{aligned}\mathbf{z}_0 &\sim p(\mathbf{z}_0)\\ \mathbf{z}_n &= \text{ODESolve}(\mathbf{f}_{\boldsymbol{\theta}}, \mathbf{z}_0, t_0, t_n)\\ \mathbf{x}_n &\sim p_\lambda(\mathbf{x}_n|\mathbf{z_n}),\end{aligned}$$

where

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}, t),$$

$\mathbf{f}_{\boldsymbol{\theta}}$ is a neural network with parameters $\boldsymbol{\theta}$ and λ represents NN parameters mapping $\mathbf{z}$ to $\mathbf{x}$. In a setting with N trajectories, with trajectory i having N_i measurements at times $\mathcal{T}_i = \{t_{i1}, t_{i2}, \ldots, t_{iN_i}\}$, we denote the measurements as $\mathcal{X}_i = \{\mathbf{x}_{ij}\}_{j=1,\ldots,N_i}$. The likelihood $\sum_{i=1}^{N} \log p(\mathcal{X}_i|\tilde{\theta})$ with $\tilde{\theta} = (\boldsymbol{\theta}, \lambda)$ is intractable but amortised variational inference as introduced across Sections 1.1.3 & 1.1.4 can be applied, with the ELBO taking the form $\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i$, with

$$\mathcal{L}_i(\tilde{\theta}) = \mathop{\mathbb{E}}_{\mathbf{z}_0^{(i)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}_0^{(i)}|\mathcal{X}_i)} \left[\log p_{\tilde{\theta}}(\mathcal{X}_i|\mathbf{z}_0^{(i)})\right] + \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z}_0^{(i)}|\mathcal{X}_i) \| p(\mathbf{z}_0^{(i)})\right].$$

As in the case of a VAE, the parameters (e.g. mean and covariance for a Gaussian) of the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z}_0^{(i)}|\mathcal{X}_i)$ are computed by passing $\mathcal{X}_i$ through a neural network encoder, however due to the longitudinal nature of the observations, more thought is required to design a suitable architecture. Rubanova et al. (2019) make use of the ODERNN concept to design such an architecture. More specifically, the posterior parameters are calculated by running an ODERNN backwards in time, taking in $\mathcal{X}_i$ in the order $[\mathbf{x}_{N_i}, \mathbf{x}_{N_i-1}, \ldots]$. The hidden state at $t = 0$, $h(0)$, is then transformed via e.g. a linear transformation to obtain

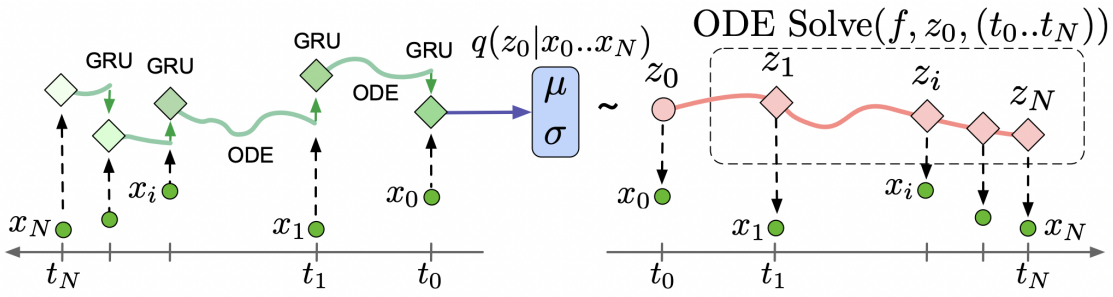

Figure 1.1: **Diagrammatic representation of the Latent Neural ODE model (Rubanova et al., 2019).**

the posterior parameters. Rubanova et al. (2019) found that this ODERNN-based encoder outperformed a standard RNN-based equivalent. A pictorial representation of the latent neural ODE model is shown in Figure 1.1, with the left hand side depicting the encoder portion and the right hand side depicting the decoder portion (the generative model).

Other work

Whilst the aforementioned concepts are those most relevant to this thesis, there exists an abundance of additional work in various related areas. For example, significant work has been carried out to extend the idea of Neural ODEs to different forms of differential equations. Notably, Kidger et al. (2020a) introduces Neural Controlled Differential Equations, which allow for continuous integration of observed data into a neural differential equation's hidden trajectory, allowing for flexible modelling of irregularly sampled data. Additionally, Neural Stochastic Differential Equations (Tzen and Raginsky, 2019; Kidger et al., 2021) have been introduced which can be thought as a generalisation of Neural ODEs within which noise is injected into the latent trajectory at each infinitesimal time step. Neural SDEs have also been considered in a latent variable framework by Li et al. (2020), where it can be seen that they can be considered an infinite-dimensional form of variational autoencoder.

Increased interest in the area has also led to notable efforts to improve the availability of quality software to support ODE-based modelling. Work within the Julia (Bezanson et al., 2017) community has led to a suite of automatic differentiation-aware differential equation solvers, namely DifferentialEquations.jl (Rackauckas and Nie, 2017b). This can be paired with companion packages, e.g. DiffEqFlux (Rackauckas et al., 2019) and DiffEqSensitivity to support numerous different differentiation methods, including explicit forward- and reverse-mode AD and various forms of the adjoint method (Rackauckas et al., 2018). Efforts have also been made within the Python community to increase support. For example, the modern, fast python-based ML framework of JAX(Bradbury et al., 2018a) is having its differential equation functionality added to via e.g. Diffrax (Kidger, 2022) and `jax.experimental.ode`. Furthermore, torchdiffeq (Chen, 2018), the original framework introduced by Chen et al. (2018b) in the context of Neural ODEs, has been extended to support Neural CDEs (Kidger, 2021a) and Neural SDEs (Kidger, 2021b), allowing neural differential equations to be easily embedded within other pytorch-based ML models. Whilst other frameworks were used in prototyping, due to HPC support and open source model availability we make use of the Python/Pytorch ecosystem for the work presented in this thesis.

Only recently has the community begun to realise the potential of ODEs to improve ML-based modelling and that it is computationally possible to work with ODEs in large-scale ML settings. Given the significant developments of the area in this time it is exciting to consider how the area and its application settings may continue to expand.

1.3 Survival Analysis

Chapters 3 & 4 introduce novel survival analysis approaches. This section (1.3) facilitates the reader's understanding of these chapters by providing the necessary background to these

contributions. It begins by introducing the general concept of single-risk survival analysis — the most basic time-to-event modelling setting — in both a qualitative and mathematical sense (Section 1.3.1). It then moves on to describe the competing risks setting, where patients are at risk of multiple event types which compete to first affect the patient (Section 1.3.2). Following this, it introduces the ubiquitous Cox Proportional Hazards model (Section 1.3.3) before covering the concept of joint longitudinal-survival modelling (Section 1.3.4), a core focus of Chapter 4.

1.3.1 Single-Risk Survival Analysis

Single-risk survival analysis considers the problem of modelling time-to-event data when all outcomes are considered either "event" or "no event", with "event" corresponding to some terminal outcome. A canonical example of this would be a simple clinical trial on advanced cancer patients where one is interested in the efficacy of a drug on death prevention. In this context, death would be the event and a single-risk model could be argued to be appropriate. Typical survival analysis of this kind assumes that a single baseline measurement is made for each patient i, denoted $\mathbf{x}^{(i)}$, and this is in some way informative about the amount of time that they will survive for. In the clinical trial context, $\mathbf{x}^{(i)}$ may include various biomarkers (the effect of which should be controlled for via randomised controlled trial methodology) as well as the potential treatment. Fitting a single-risk survival model to the data then allows for the quantitative analysis of the effect of such a treatment.

An important consideration in single-risk survival analysis, and indeed survival analysis in general, is the presence of censoring. Censoring in general refers to the idea that a patient may experience an event at a time outside of the observation window of a given study. Most relevant for our modelling purposes is right censoring. This is when a patient can be considered as part of the study but does not record an event within the observed time window.

A good example of this would be a patient surviving the observation follow-up window in the advanced cancer clinical trial. For a model to be appropriate for the survival analysis setting, it must consider such cases.

More mathematically, single-risk survival analysis considers data which for N subjects takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$, where $s^{(i)}$ denotes event time, $\mathbf{x}^{(i)}$ denotes subject covariates and $k^{(i)}$ denotes event type. In the single-risk setting, $k^{(i)} \in \{\varnothing, 1\}$, with $k^{(i)} = 1$ implying the event of focus (e.g. death) occurred at $s^{(i)}$ and $k^{(i)} = \varnothing$ implying right censorship (as described above). The aim of a survival model is to learn from this data the association between covariates $\mathbf{x}^{(i)}$ and event risk and/or to be able to predict the time-to-event distribution for an unobserved test patient. A commonly targeted quantity in single-risk survival analysis is the survival function, $S(t) \equiv P(T > t) \equiv 1 - P(T \leq t) \equiv 1 - F(t)$, where $F(t)$ is the cumulative distribution function (CDF) corresponding to the random variable T.

1.3.2 Competing Risks Survival Analysis

Competing risks survival analysis refers to the setting within which one is not just interested in whether an individual experiences an event, but is rather also interested in the subtype of event that occurred. For example, in the medical setting, if one is considering a cardiovascular cohort, it is also likely that some members of the cohort have other morbidities, e.g. kidney disease. It may therefore be desirable to consider death due to each of these diseases or groups of diseases as separate event types. In this setting, kidney disease is referred to as a competing risk, as it competes with the primary process of interest, i.e. cardiovascular disease, to kill the patient. A competing risk model allows the modelling of the cause-specific risk of an event in a setting such as this, therefore allowing greater granularity, and therefore utility, in a clinical setting.

Therefore, in the competing risks setting, survival data again takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$, but now $k^{(i)} \in \{\varnothing, 1, 2, \ldots, K\}$ for K event types of interest. In the above, these event types would correspond to "death associated with disease k". In the competing risks setting, the overall survival function is still of interest, however the quantity $P(T \leq t, k)$, i.e. the probability of experiencing event k before time t accounting for the presence of competing risks, becomes of interest, and is referred to as the Cumulative Incidence Function (CIF).

1.3.3 Cox Proportional Hazards

Perhaps the most well-known and widely applied survival analysis method, which also forms the basis of many modern approaches, is the Cox Proportional Hazards model (Cox, 1972). To appreciate this and related methods, one must define the hazard function. Mathematically, the hazard function $\lambda(t)$ is defined as

$$\lambda(t) = \lim_{\mathrm{d}t \to 0} \frac{\Pr\{t \leq T < t + \mathrm{d}t \mid T \geq t\}}{\mathrm{d}t}.$$

Qualitatively, $\lambda(t)$ expresses the probability per unit time (i.e. risk) of experiencing an event at time t given that an event has not occurred prior to time t.

The Cox Proportional Hazards model posits that the hazard function $\lambda(t|\mathbf{x})$ for an individual with covariates $\mathbf{x}$ can be expressed as

$$\lambda(t|\mathbf{x}) = \lambda_0(t) \exp\big(\beta^T \mathbf{x}\big), \tag{1.14}$$

where $\lambda_0(t)$ is a baseline hazard function and β are learnable parameters. Note that this assumes that the hazard function of all individuals has the same shape over time but is scaled by a person-dependent multiplicative factor $\exp\big(\beta^T \mathbf{x}\big)$. At first glance this may seem an arbitrary approach to the survival problem, but assuming this form of model has useful implications for downstream modelling. In particular, it can be shown that the full log-likelihood associated with the model can be well approximated by a so-called partial

log-likelihood which is *independent of* $\lambda_0(t)$. Therefore, appropriate values of β can be learnt without having to be concerned with $\lambda_0(t)$. Furthermore, if one considers the hazard ratio, that is the ratio between the hazard of patient i and patient j, it is found that

$$\frac{\lambda(t|\mathbf{x}_i)}{\lambda(t|\mathbf{x}_j)} = \frac{\lambda_0(t)\exp\big(\beta^T\mathbf{x}_i\big)}{\lambda_0(t)\exp(\beta^T\mathbf{x}_j)} = \exp\big(\beta^T(\mathbf{x}_i - \mathbf{x}_j)\big),$$

which again does not involve $\lambda_0(t)$ and is additionally completely independent of t. This allows practitioners to make easily understood statements of the form "Patient A is twice as at risk of death as patient B" without having to involve time-dependence or additional caveats, which is particularly valued by those performing medical statistical analysis on study data.

Whilst the Cox model is valuable in many settings, it is important to be mindful of its limitations. Firstly, the form of hazard posited in (1.14) could be considered quite restrictive. For example, it assumes that the time evolution profile of risk for all patients is the same up to a multiplicative factor. In reality it is likely that patients will not behave this uniformly. It also assumes a relatively restrictive functional form for the covariate-dependent multiplicative factor $\exp\big(\beta^T\mathbf{x}\big)$. Furthermore, it should be noted that the time-independence of hazard ratios in the Cox setting, which allows statements such as 'Patient A is twice as at risk of death as patient B" to be made, arises from model assumptions. In reality, the true hazard ratio may be time-dependent, but this cannot be captured by a Cox model.

1.3.4 Joint Longitudinal-Survival Modelling

In the longitudinal survival analysis setting, data takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}_0^{(i)}, \mathbf{X}(\mathbf{t})^{(i)}, k^{(i)}\}_{i=1}^N$, where $s^{(i)}$ denotes event time, $\mathbf{x}_0^{(i)}$ denotes subject covariates at baseline, $\mathbf{X}(\mathbf{t})^{(i)} \in \mathbb{R}^{d\times T}$ denotes subject covariate measurements $\mathbf{x}(t)$ at times $\mathbf{t} = [t_0, \ldots, t_T]$, and $k^{(i)} \in \{\varnothing, 1\}$ denotes event type (we assume a single-risk setting here). The goal of a joint longitudinal-survival model is to *simultaneously* model the longitudinal variables $\mathbf{x}(t)$ alongside the event

times s, thus allowing one to perform predictions on either variable at test time and to understand the relationship between longitudinal variables and survival time.

To proceed, it is typically assumed that there exist two models: a longitudinal model for $\mathbf{x}(t)$ and a survival model for s, each of which may involve parameters, here denoted $\boldsymbol{\theta}$. These are then coupled by assuming shared dependence on a set of *random effects* $\mathbf{b}$.

For example, in the most basic and commonly used form of this model, a linear mixed effects model is posited for the longitudinal part and a Cox proportional hazards model is used for the survival part. The longitudinal part therefore takes the form

$$x_i(t) = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{u}_i(t) + \mathbf{b}_i^{\mathrm{T}}\mathbf{z}_i(t) + \varepsilon_i(t)$$

with $\mathbf{b}_i \sim \mathcal{N}(0, D)$ and $\varepsilon_i(t) \sim \mathcal{N}(0, \sigma^2)$. It is common to set $\beta = [\beta_0 \quad \beta_1]^{\mathrm{T}}$, $\mathbf{u_i(t)} = [1 \quad t]^{\mathrm{T}}$ and $\mathbf{b_i} = [b_i^{(0)} \quad b_i^{(1)}]^{\mathrm{T}}$, $\mathbf{z_i(t)} = [1 \quad t]^{\mathrm{T}}$. This means that each subject's $x_i(t)$ is derived by adding Gaussian noise to a straight line with intercept $\beta_0 + b_i^{(0)}$ and slope $\beta_1 + b_i^{(1)}$. Meanwhile, the hazard is given by

$$h_i\left(t\right) = h_0(t) \exp\left\{\gamma^{\top} x_0^{(i)} + \alpha m_i(t)\right\},$$

where $m_i(t)$ denotes the mean of $x_i(t)$, i.e.

$$m_i(t) \coloneqq \boldsymbol{\beta}^{\mathrm{T}}\mathbf{u}_i(t) + \mathbf{b}_i^{\mathrm{T}}\mathbf{z}_i(t).$$

A key goal in this context is to infer $\boldsymbol{\theta} = (\boldsymbol{\beta}, \gamma, \alpha)$, as these parameters convey the mean longitudinal behaviour of the data, the association between baseline covariates and survival time, and the association between longitudinal covariates and survival time respectively. It may also be of interest to target the posterior over a subject's random effects $\mathbf{b}_i$ in order to understand individual subject behaviour.

Performing maximum likelihood inference for $\boldsymbol{\theta}$ on a joint longitudinal-survival model of this form equates to maximising a log-likelihood of the form

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_{i=1}^{N} \log p(s^{(i)}, k^{(i)}, \mathbf{X}(\mathbf{t})^{(i)} | \mathbf{x}_0^{(i)}, \boldsymbol{\theta}), \tag{1.15}$$

with

$$p(s^{(i)}, k^{(i)}, \mathbf{X}(\mathbf{t})^{(i)} | \mathbf{x}_0^{(i)}, \boldsymbol{\theta}) = \int \left\{ p(\mathbf{X}(\mathbf{t})^{(i)} | \mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i) p(s^{(i)}, k^{(i)} | \mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i) p(\mathbf{b}_i) \right\} \mathrm{d}\mathbf{b}_i \,. \quad (1.16)$$

Note that this resembles the log-likelihood format often encountered in latent variable settings (see Sections 1.1.3 & 1.1.4), with $\mathbf{b}_i$ here playing the role of the variable typically denoted $\mathbf{z}_i$ in latent variable treatments. The log-likelihood in (1.15) is therefore amenable to the same methods as discussed in the context of latent variable models, in particular Expectation Maximisation and its derived algorithms.

1.3.5 Metrics

In Chapters 3 & 4, we evaluate various survival models with respect to some metrics largely specific to the survival setting. We define these metrics below for reference.

Time-dependent Concordance Index (Antolini et al., 2005): We define the Cause-specific Time-dependent Concordance Index for event k so that the metric is immediately applicable to the competing risks setting. Replacing CIF with CDF in what follows provides the single-risk version. Consider two patients i and j. Suppose patient i experiences event $k^{(i)}$ at time $s^{(i)}$ and that patient j outlives patient i so that $s^{(i)} < s^{(j)}$. Then, the time-dependent concordance represents the probability that the predicted CIF of patient i for event $k^{(i)}$ exceeds the same event's predicted CIF for patient j. The time-dependent concordance for event k, denoted $C_{\text{td}}^{(k)}$, is therefore formally defined via

$$P\left(\hat{F}_k\left(s^{(i)} \mid \mathbf{x}^{(i)}\right) > \hat{F}_k\left(s^{(i)} \mid \mathbf{x}^{(j)}\right) \mid s^{(i)} < s^{(j)}, k^{(i)} = k\right).$$

In practice, it is empirically estimated via

$$C_{\text{td}}^{(k)} \approx \frac{\sum_{i \neq j} A_{k,i,j} \cdot \mathbb{1}\left(\hat{F}_k\left(s^{(i)} \mid \mathbf{x}^{(i)}\right) > \hat{F}_k\left(s^{(i)} \mid \mathbf{x}^{(j)}\right)\right)}{\sum_{i \neq j} A_{k,i,j}},$$

where $A_{k,i,j} \coloneqq \mathbb{1}\left(k^{(i)} = k, s^{(i)} < s^{(j)}\right)$. Note that $C_{\text{td}}^{(k)} \in [0, 1]$, with values closer to 1 representing stronger discriminative performance. The intution accompanying $C_{\text{td}}^{(k)}$ is that if

a model is good, the predicted cumulative risk at time $s^{(i)}$ should be greater for patient i who experiences the event at that time than for a patient j that remains alive beyond $s^{(i)}$.

Integrated Brier Score (IBS): The Brier score (BS) is a metric which measures the ability of a classifier to predict binary outcomes. Given N binary observations $y_i \in \{0, 1\}$ and N predictions $\hat{p}_i$ from a binary classifier for $p(y_i = 1)$, the BS is $\sum_i (y_i - \hat{p}_i)^2 / N$. In the survival analysis setting, a Brier score, $\mathrm{BS}(t)$ can be obtained for N patients at time t by considering the Brier score of the binary variable U_i which is 1 for patients who live beyond t and 0 for those that experience an event before t.

More specifically, at time t, let U_i denote the random variable which takes value 1 if $s^{(i)} \leq t$, i.e. if patient i has experienced an event, and 0 if $s^{(i)} > t$, i.e. if patient i is alive at t. The Brier Score associated with this variable over all patients $i = 1, \ldots, N$ is

$$\mathrm{BS}_0(t) = \frac{1}{N} \sum_{i=1}^{N} \left[\left(1 - \hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right)^2 \mathbb{1}\left\{s^{(i)} \leq t, k^{(i)} = 1\right\} + \hat{F}\left(t \mid \mathbf{x}^{(i)}\right)^2 \mathbb{1}\left\{s^{(i)} > t\right\} \right].$$

In the presence of random censoring the terms are reweighted as

$$\mathrm{BS}(t) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\left(1 - \hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right)^2 \mathbb{1}\left\{s^{(i)} \leq t, k^{(i)} = 1\right\}}{\hat{G}\left(s^{(i)}\right)} + \frac{\hat{F}\left(t \mid \mathbf{x}^{(i)}\right)^2 \mathbb{1}\left\{s^{(i)} > t\right\}}{\hat{G}(t)} \right],$$

where $\hat{G}$ is the Kaplan-Meier estimate of the censoring survival function $P(C > t)$ (see Graf et al. (1999) for details). Numerically integrating $\mathrm{BS}(t)$ over the range of test times and dividing by the interval provides the Integrated Brier Score (IBS), i.e.

$$\mathrm{IBS} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \mathrm{BS}(s)\, \mathrm{d}s\,,$$

for which smaller values indicate better classification performance (Graf et al., 1999).

Integrated Negative Binomial Log-Likelihood (INBLL): Following on from the above IBS discussion and definition, we can define the negative log-likelihood of the variable $U = \{U_i\}_{i=1}^{N}$ as

$$\mathrm{NBLL}_0(t) = -\frac{1}{N} \sum_{i=1}^{N} \left[\log\left[\hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right] 1\left\{s^{(i)} \leq t, k^{(i)} = 1\right\} + \log\left[1 - \hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right] \mathbb{1}\left\{s^{(i)} > t\right\} \right].$$

Applying the same censoring weighting as previously provides

$$\mathrm{NBLL}(t) = -\frac{1}{N}\sum_{i=1}^{N}\left[\frac{\log\left[\hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right]\mathbb{1}\left\{s^{(i)} \leq t, k^{(i)} = 1\right\}}{\hat{G}\left(s^{(i)}\right)} + \frac{\log\left[1-\hat{F}\left(t \mid \mathbf{x}^{(i)}\right)\right]\mathbb{1}\left\{s^{(i)} > t\right\}}{\hat{G}(t)}\right].$$

This can be integrated as in the IBS case as

$$\mathrm{INBLL} = \frac{1}{t_2 - t_1}\int_{t_1}^{t_2}\mathrm{NBLL}(s)\,\mathrm{d}s$$

to obtain the Integrated Negative Binomial Log-Likelihood (INBLL) metric for which smaller values indicate better classification performance (Kvamme et al., 2019).

Mean Absolute Error (MAE) and Root Mean Square Error (RMSE): These represent the MAE and RMSE between the mean of a model's patient-specific time-to-event distribution and that patient's observed event time calculated across all (non-censored) patients.

Mathematically, the MAE is therefore

$$\mathrm{MAE} \coloneqq \frac{\sum_i \left|s^{(i)} - \hat{s}^{(i)}\right| \cdot \mathbb{1}\left(k^{(i)} = 1\right)}{\sum_i \mathbb{1}\left(k^{(i)} = 1\right)},$$

where $s^{(i)}$ and $\hat{s}^{(i)}$ denote the observed event time and mean of the predicted survival distribution respectively for patient i.

Meanwhile the RMSE is

$$\mathrm{RMSE} \coloneqq \sqrt{\frac{\sum_i \left(s^{(i)} - \hat{s}^{(i)}\right)^2 \cdot \mathbb{1}\left(k^{(i)} = 1\right)}{\sum_i \mathbb{1}\left(k^{(i)} = 1\right)}},$$

where $s^{(i)}$ and $\hat{s}^{(i)}$ denote the observed event time and mean of the predicted survival distribution respectively for patient i.

Chapter Two

Interpretable Simultaneous Dimensionality Reduction and Feature-Level Clustering with Derivative-Based Variational Autoencoders

This chapter presents two novel latent variable models: DeVAE and BasisDeVAE. DeVAE performs prior knowledge-aware VAE-based dimensionality reduction via derivative-based methodology. BasisDeVAE extends this notion by additionally performing feature clustering simultaneously. The core content of this chapter is based on work published in the Proceedings of The Thirty-eighth International Conference on Machine Learning (ICML 2021) as Danks and Yau (2021).

The structure of this chapter follows the natural pattern for presenting such models. It begins by introducing the need for the model and how it relates to other modern methods

in the literature (Section 2.1) before proceeding to introduce the idea of constraining decoder outputs of a VAE via derivative-based modelling (Section 2.2). It then covers how this idea can be used to additionally cluster features based on their behaviour (Section 2.3). Experimental results on synthetic and real world data are then presented (Section 2.4), before a discussion section (Section 2.5) summarises the key contributions of the work, how they complement various other research directions and how they could be naturally extended in future work.

2.1 Introduction

Variational Autoencoders (VAEs) (Kingma and Welling, 2014) have become ubiquitous in modern machine learning and are applied in a wide range of settings, including the generation and disentangled latent coding of images (Gregor et al., 2015; Higgins et al., 2017; Kumar et al., 2018), text generation (Bowman et al., 2016a; Xu et al., 2020) and semi-supervised learning (Kingma et al., 2014; Maaløe et al., 2016). They have also played an important role in interpreting high-dimensional biological data, such as that produced in genomics settings (Lopez et al., 2018; Simidjievski et al., 2019; Qiu et al., 2020).

2.1.1 Extensions of VAEs

The standard VAE integrates Bayesian latent variable modelling, deep neural networks (DNNs) and variational inference, the combination of which results in a probabilistic *encoder* which maps a (high-dimensional) input onto a (low-dimensional) latent space and a corresponding probabilistic *decoder* which maps each position in the latent space onto a distribution in the observation space.

Many variants of the standard VAE have been developed, including the conditional VAE (cVAE) (Sohn et al., 2015) which conditions the behaviour of the encoder and decoder on additional fixed inputs, the neural decomposition VAE (Märtens and Yau, 2020) which imposes a functional ANOVA structure on the decoder, and the beta-VAE (Higgins et al., 2017) which incorporates additional regularisation to encourage latent variable disentanglement. Recently, an approach that combines dimensionality reduction and feature-level clustering within a VAE framework (BasisVAE) has been developed (Märtens and Yau, 2020). BasisVAE groups subsets of features together whose behaviour is similar over the latent dimensions. This type of model is of particular utility on *tabular* datasets where each feature may have a distinct standalone meaning (e.g. it represents a protein, gene or age).

Figure 2.1 demonstrates the overall pipeline of simultaneous dimensionality reduction and feature-level clustering — given an observed data matrix $X \in R^{N\times d}$ (N observations, d features; leftmost item in Figure 2.1) as input, a mapping from a latent variable z to each dimension of the d observation dimensions is learnt, and these profiles are clustered according to their behaviour (lower-right plot in Figure 2.1). Standard VAEs lack the feature clustering element (upper-right plot of Figure 2.1). Values of the latent variable z corresponding to each observation x are also assigned and structure can be seen by arranging samples (rows within X) by z-value (upper-middle plot of Figure 2.1) if feature clustering is not performed. If feature clustering is also performed, we can also group columns by their clustering assignment (bottom-middle of Figure 2.1). Hence it can be said that the application of dimensionality reduction via a standard VAE uncovers structure only in the *rows* (samples) of tabular data $X \in \mathbb{R}^{N\times d}$, whereas simultaneous dimensionality reduction and feature-level clustering aims to uncover structure in both the rows (samples) *and* columns (features).

The aforementioned VAE developments recognise the benefit of introducing additional structure into the flexible VAE framework in order to encourage desirable model behaviour. However, specifying particular functional characteristics with current VAE decoders remains

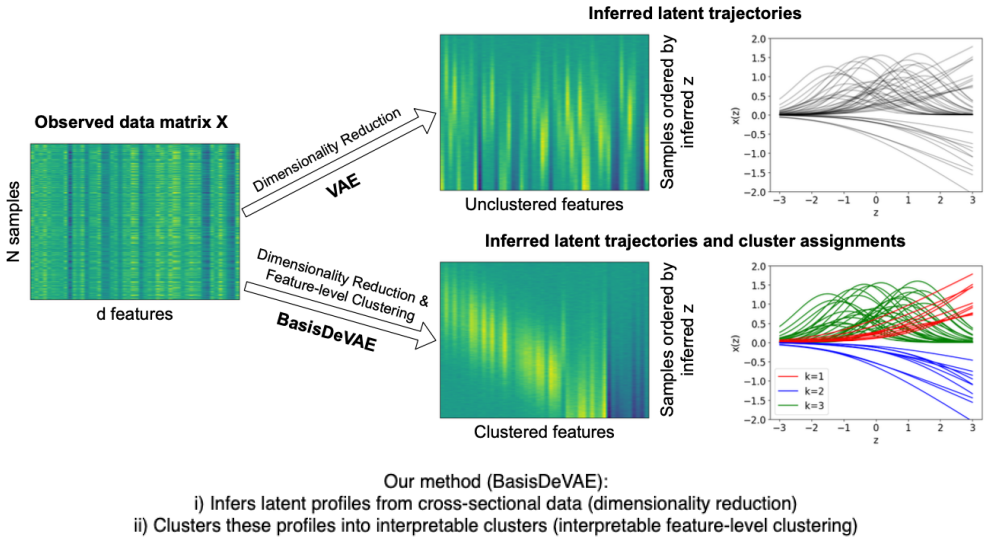

Figure 2.1: **Simultaneous dimensionality reduction and feature-level clustering.** Given an observed data matrix $X \in \mathbb{R}^{N \times d}$, we would like to i) learn a low-dimensional representation $\mathbf{z}$ (dimensionality reduction) and ii) cluster features according to their behaviour as a function of $\mathbf{z}$ (feature-level clustering). The standard VAE framework allows us to perform only i). With our proposed BasisDeVAE model we can perform both tasks simultaneously and guarantee *interpretability* of the inferred cluster assignments.

an open challenge. For example, BasisVAE allows us to identify features which share exactly the same functional shape or dynamics, but it does not provide a mechanism to define a broader group of shared patterns among features, e.g. to cluster together all monotonically increasing functions. The ability to characterise certain functional behaviours is important in applications where *a priori* knowledge or physical laws constrain feature dynamics.

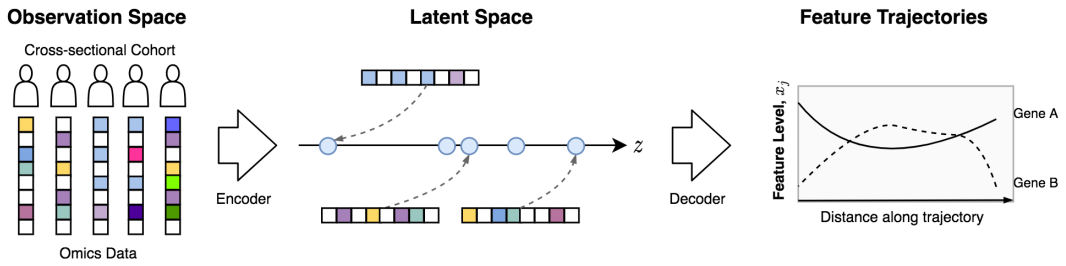

Figure 2.2: **Modelling biological progression.** If the dominant source of variation in a cross-sectional dataset is associated with biological or disease progression, we can use a VAE to encode and map the samples onto a one-dimensional latent space (pseudotime) and decode to understand the feature-level variability with pseudotime.

2.1.2 Contribution

In this chapter, we present two novel VAE models, namely DeVAE and BasisDeVAE, which model feature-level behaviour $\mathbf{x}$ in terms of *derivatives* with respect to the latent dimensions $\mathbf{z}$, i.e. $\frac{\partial \mathbf{x}}{\partial z_j} = f_\theta^{(j)}(z_j)$, where $f_\theta^{(j)}$ is a DNN. Note that defining $\frac{\partial \mathbf{x}}{\partial z_j}$ so that it does not depend on $\mathbf{x}$ removes the risk of chaotic behaviour and enables the use of quadrature rather than full ODE solves, as expanded upon later in this chapter.

We show how use of this approach allows for greater control over decoder behaviour in settings where certain forms of *a priori* knowledge and/or physical constraints, such as monotonicity and transience, are desirable without having to explicitly define parametric functional forms.

This work is particularly motivated by modelling *disease* or *biological progression* from cross-sectional data. In this problem, a cross-sectional collection of input samples is projected onto a one-dimensional latent space. If the dominant latent source of variation in the cross-sectional data is temporal, the positioning of samples in the latent space then

corresponds to relative ordering in time or *pseudotime*. This type of analysis has been used to determine temporal patterns associated with biological processes, such as cellular differentiation and cancer progression, where longitudinal time series data may be difficult or impossible to collect directly. Figure 2.2 provides a pictorial representation of this modelling context.

We demonstrate the real-world utility of DeVAE and BasisDeVAE by applying them to synthetic data, image-derived brain pathology biomarkers from the OASIS-3 dataset (LaMontagne et al., 2019) and large-scale single-cell RNA sequencing data (Ernst et al., 2019).

2.1.3 Background: BasisVAE

BasisVAE (Märtens and Yau, 2020) enables simultaneous dimensionality reduction and feature-level clustering within the VAE framework (see Section 1.1.4 for VAE background). It achieves this by modifying the form of the decoder function, introducing additional cluster-associating latent variables into the generative model and devising an efficient collapsed variational inference scheme for learning.

The generative model of BasisVAE, in the case of a Gaussian conditional likelihood, takes the form

$$\begin{aligned}
\mathbf{z}_i &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
\boldsymbol{\pi}|\boldsymbol{\alpha} &\sim \text{Dirichlet}(\boldsymbol{\alpha}) \\
\left(w^{j1}, \ldots, w^{jK}\right)|\boldsymbol{\pi} &\sim \text{Categorical}\left(\pi_1, \ldots, \pi_K\right) \\
x_i^{(j)}|\mathbf{w}^j, \mathbf{z}_i, \tilde{\theta} &\sim \mathcal{N}\left(\sum_{k=1}^{K} w^{jk}\lambda_{jk} f_{\text{basis}}^{(k)}\left(\mathbf{z}_i + \delta_{jk}\right), \sigma_j^2\right),
\end{aligned}$$

where $\tilde{\theta} = (\theta, \{\lambda_{jk}\}, \{\delta_{jk}\}, \{\sigma_j\})$. Here δ_{jk} and λ_{jk} are parameters representing the amount of translation and scaling respectively associated with feature j and component k, and a Dirichlet prior is introduced on $\boldsymbol{\pi}$ to induce sparse cluster assignments. It is therefore a

mixture model where each feature is assigned to one of K basis functions $f_{\text{basis}}^{(k)}$.

Inference is carried out on this model using *collapsed variational inference* (Hensman et al., 2012; Hensman et al., 2015) by introducing a categorical variational posterior

$$q_{\boldsymbol{\xi}}(\mathbf{w}) = \prod_{j=1}^{d} q_{\boldsymbol{\xi}}(\mathbf{w}^j) = \prod_{j=1}^{d} \text{Categorical}\left(\xi_{j1}, \ldots, \xi_{jK}\right)$$

over cluster assignments, and a variational lower bound on the marginal log-likelihood is derived by repeated application of Jensen's inequality and by additionally marginalising out $\boldsymbol{\pi}$ (see Appendix A of Märtens and Yau (2020) for details). The resulting loss function is

$$\begin{aligned}
\mathcal{L}(\tilde{\theta}, \phi, \boldsymbol{\xi}) = &- \sum_{i=1}^{N} \mathbb{E}_{q_\phi(\mathbf{z}_i|\mathbf{y}_i)} \mathbb{E}_{q_{\boldsymbol{\xi}}(\mathbf{w})} \log p_{\tilde{\theta}}\left(\mathbf{x}_i|\mathbf{z}_i, \mathbf{w}\right) && (2.1)\\
&- \gamma(\log \mathrm{B}(\mathbf{n} + \boldsymbol{\alpha}) - \log \mathrm{B}(\boldsymbol{\alpha})) && (2.2)\\
&+ \gamma \mathbb{E}_{q_{\boldsymbol{\xi}}(\mathbf{w})} \log q_{\boldsymbol{\xi}}(\mathbf{w}) && (2.3)\\
&+ \beta \sum_{i=1}^{N} \mathrm{KL}\left[q_\phi\left(\mathbf{z}_i|\mathbf{y}_i\right) \| p\left(\mathbf{z}_i\right)\right] && (2.4)\\
&- \sum_{j=1}^{d} \sum_{k=1}^{K} \left[\log\left(\mathcal{N}\left(\delta_{jk}|0,1\right) \Gamma\left(\lambda_{jk}|0,1\right)\right)\right] && (2.5)
\end{aligned}$$

where

$$\mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}$$

is the multivariate beta function. Term (2.5) arises from priors on the translation and scale parameters (these are estimated via MAP), and β and γ are hyperparameters allowing the relative importance of the prior $p(\mathbf{z})$ and sparse clustering prior to be edited in the case of large d and/or N (Higgins et al., 2017). Inference in the BasisVAE setting therefore equates to minimising $\mathcal{L}$ jointly over the decoder parameters $\tilde{\theta}$, posterior cluster assignments $\boldsymbol{\xi}$ and encoder parameters ϕ, which is performed via mini-batch gradient descent.

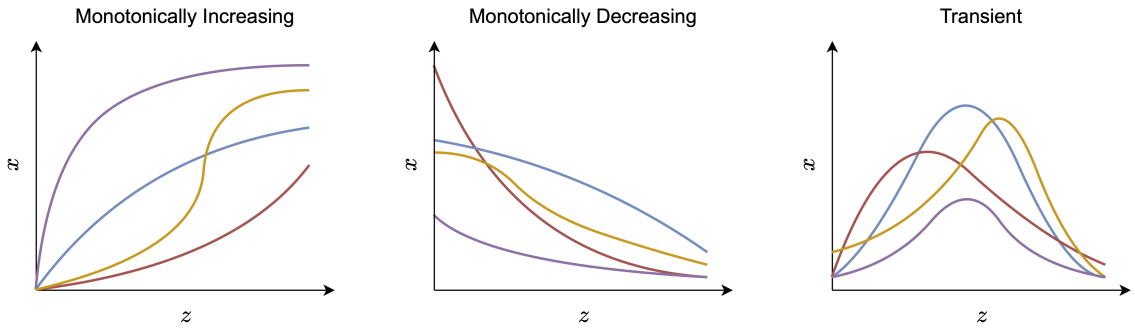

Figure 2.3: **Feature behaviours.** Schematic representations of the monotonically increasing, monotonically decreasing and transient feature behaviours often present within biological data that BasisDeVAE aims to capture.

2.2 Derivative-Based VAE (DeVAE)

The functional form of the decoder in a standard VAE is challenging to control due to its specification as a DNN and training being performed in the DNN's weight space. In our biological progression modelling applications, biomarkers often only exhibit a limited number of high-level behaviours, namely monotonic increases, monotonic decreases or a time-limited transient signal (see Figure 2.3). While a standard VAE could "learn" these behaviours given a sufficiently large number of low-noise samples, we would like to be able to robustly enforce such structures in low-sample or high-noise settings while still maintaining flexibility to model complex feature behaviours.

In order to do this, we introduce DeVAE. DeVAE specifies the decoder via its derivatives with respect to the latent variable $\mathbf{z}$. Formally, we model

$$\frac{\partial \mathbf{x}}{\partial z_p} = f_\theta^{(p)}(z_p), \tag{2.6}$$

with $f_\theta^{(p)}(z_p)$ a neural network-based function. Note that the *range* of values output by a neural network is easy to control. For example, setting the final activation of $f_\theta^{(p)}$ to be a

softplus function ensures positivity of the derivative and hence monotonicity of $\mathbf{x}(z_p)$, whilst a sigmoid-based final activation of $f_\theta^{(p)}$ limits short-scale variation of $\mathbf{x}(z_p)$.

This derivative-based specification leads to the overall decoder output being expressed via the integral

$$\mathbf{x}(\mathbf{z}) = \mathbf{x}_0 + \int_{\mathbf{0}}^{\mathbf{z}} \mathbf{f}_\theta(\mathbf{z}') \cdot \mathrm{d}\mathbf{z}', \tag{2.7}$$

where $\mathbf{x}_0 = \mathbf{x}(\mathbf{0})$ and $[\mathbf{f}_\theta(\mathbf{z})]_p = f_\theta^{(p)}(z_p)$. The form of (2.6) implies that the integral in (2.7) decomposes into a sum of $\dim(\mathbf{z})$ one-dimensional integrals of the form

$$I_\theta(z) = \int_0^z f_\theta(z')\mathrm{d}z'.$$

which can be rewritten as

$$I_\theta(z) = \frac{z}{2}\int_{-1}^{1} f_\theta\left(\frac{z}{2}(u+1)\right)\mathrm{d}u$$

and evaluated using Gauss-Legendre (GL) quadrature. The explicit calculation of $I_\theta(z)$ using GL quadrature of order n (we use $n = 15$ throughout) is

$$I_\theta(z) = \frac{z}{2}\sum_{i=1}^{n} w_i f_\theta\left(\frac{z}{2}(u_i+1)\right),$$

with u_i, w_i the order n GL quadrature nodes and weights computed via Legendre polynomials (see e.g. Press et al. (2007) for details), implying that the overall computation of the decoder's output becomes

$$\mathbf{x}(\mathbf{z}) = \mathbf{x}_0 + \sum_{i,p} \frac{z_p}{2} w_i f_\theta^{(p)}\left(\frac{z_p}{2}(u_i+1)\right). \tag{2.8}$$

The final form of the decoder computation (2.8) therefore reduces to a weighted sum of neural network evaluations. Hence, backpropagation through the decoder is straightforward and the computation is fully parallelisable over data points, p, and i during optimisation using mini-batch gradient descent. We provide a GPU-aware PyTorch (Paszke et al., 2019a) implementation of our approach at https://github.com/djdanks/BasisDeVAE and demonstrate its application to multiple settings in Section 3.5.

2.3 Basis Derivative-Based VAE (BasisDeVAE)

We next embed DeVAE within the BasisVAE framework to perform simultaneous feature-level clustering and dimensionality reduction with control over the behaviour of the feature clusters. Let $g_{\tilde{\theta}}^{jk}(\mathbf{z})$ be the DeVAE-derived behaviour of feature $x_j(\mathbf{z})$ given cluster k. The generative model of BasisDeVAE is then

$$\begin{aligned}
\mathbf{z}_i &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
\boldsymbol{\pi}|\boldsymbol{\alpha} &\sim \text{Dirichlet}(\boldsymbol{\alpha}) \\
\left(w^{j,1}, \ldots, w^{j,K}\right)|\boldsymbol{\pi} &\sim \text{Categorical}\left(\pi_1, \ldots, \pi_K\right) \\
x_i^{(j)}|\mathbf{w}^j, \mathbf{z}_i, \tilde{\theta} &\sim \mathcal{N}\left(\sum_{k=1}^{K} w^{j,k} g_{\tilde{\theta}}^{jk}(\mathbf{z}_i), \sigma_j^2\right),
\end{aligned}$$

which is obtained by starting with the BasisVAE generative model (see Section 2.1.3) and inserting $g_{\tilde{\theta}}^{jk}(\mathbf{z}_i)$ in place of $f_{\text{basis}}^{(k)}(\mathbf{z}_i + \delta_{jk})$. Intuitively, this equates to removing the notion of feature j being obtained via the translation and scaling of one of K underlying basis functions and replacing it with the idea that feature j is described by a function from one of K families, each with interpretable properties specified via their derivatives.

2.3.1 BasisDeVAE: Example

In order to illustrate our approach in the context of the running example (Figure 2.3), let $k = 1$ and $k = 2$ correspond to monotonically increasing and decreasing behaviour respectively, and let $k = 3$ correspond to Gaussian-like transient behaviour. Additionally, let $f_\theta : \mathbb{R} \mapsto \mathbb{R}^{\dim(\mathbf{x}) \times K}$ be a neural network with a softplus output layer. Then $g_{\tilde{\theta}}^{j1}(z_i)$ and $g_{\tilde{\theta}}^{j2}(z_i)$ can be modelled as

$$\begin{aligned}
g_{\tilde{\theta}}^{j1}(z_i) &= [\mathbf{x}_0]_j + \int_0^{z_i} [f_\theta(z)]_{j1} \, \mathrm{d}z \\
g_{\tilde{\theta}}^{j2}(z_i) &= [\mathbf{x}_0]_j - \int_0^{z_i} [f_\theta(z)]_{j2} \, \mathrm{d}z.
\end{aligned}$$

Note that if $g_{\tilde{\theta}}^{j1}(z_i)$ or $g_{\tilde{\theta}}^{j2}(z_i)$ as defined above are mapped through a monotonically increasing function, their monotonicity properties are retained. One can therefore constrain the output range of these decoder constituents (e.g. by passing through a scaled sigmoid function) in addition to their monotonicity within this framework.

Next, for the transient component ($k = 3$), we define a Gaussian-like transient to be any function of the form $G_{t_0}(t) = A\exp(-[h_{t_0}(t)]^2)$ with $A > 0$, $h_{t_0}(t_0) = 0$, and $h_{t_0}(t)$ a monotonically increasing (or decreasing, but we restrict attention to the former without loss of generality) function. The motivation for this comes from the fact that $h_{t_0}(t)$ (under these constraints) can be thought of as a natural generalisation of the function $(t-t_0)/(\sqrt{2}\sigma)$ which would feature within a parametric Gaussian representation of a transient, motivating our use of the term "Gaussian-like transient". More formally, G_{t_0} will only have one stationary point, namely a unique maximum point at $t = t_0$ and the function will decay away from this point.

The conditions on $h_{t_0}(t)$ are met by the derivative-based function

$$h_{t_0}(t) = (t - t_0) \times \text{softplus}\left(c + \int_0^t (\tau - t_0) f(\tau) \mathrm{d}\tau\right),$$

where $f(\tau)$ is any (locally) integrable function with positive range (see below for more details and a proof). Note that we use softplus for concreteness and due to its immediate availability within typical deep learning frameworks, but it could be replaced with any smooth monotonically increasing function $u : \mathbb{R} \mapsto \mathbb{R}_{>0}$ without invalidating the proof provided below.

We utilise this observation to define

$$h_{\tilde{\theta}}^{j3}(z_i) = (z_i - z_0)\text{softplus}\left(c + \int_0^{z_i} (z - z_0)\left[f_\theta(z)\right]_{j3} \mathrm{d}z\right) \tag{2.9}$$

and

$$g_{\tilde{\theta}}^{j3}(z_i) = A_j \exp\left(-\left[h_{\tilde{\theta}}^{j3}(z_i)\right]^2\right),$$

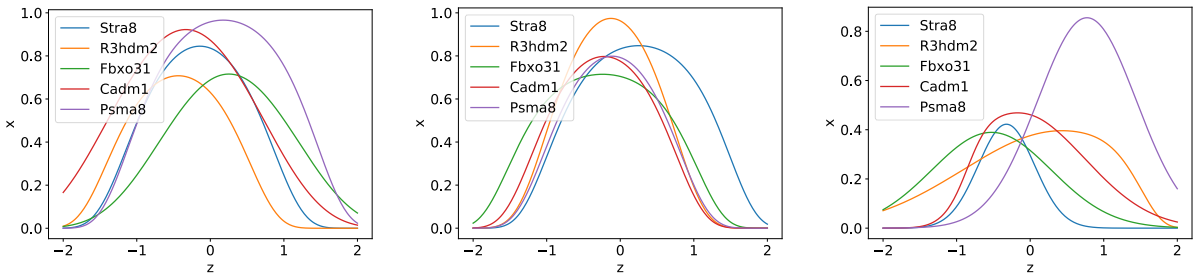

Figure 2.4: **Example $G_{t_0}(t)$ transients.** Left: $h_{t_0}(t)$ of the form (2.9) with $f(\tau)$ a random feature-dependent constant. Middle: $h_{t_0}(t)$ of the form (2.9) with $f(\tau)$ a Xavier-initialised neural network. Right: Same functional form as the middle panel, but after training on the Ernst et al. (2019) data via BasisDeVAE.

hence providing our Gaussian-like transient component. Examples of these $g_{\tilde{\theta}}^{j3}(z_i)$ transients for different $h_{\tilde{\theta}}^{j3}(z_i)$ in the context of the Ernst et al. (2019) spermatogenesis data (see Section 2.4.3) are shown in Figure 2.4, where it can be seen that this formalism allows a wide range of transient behaviours to be captured via the training of the neural network $f(\tau)$ (right panel).

Inference in the BasisDeVAE model is carried out using the collapsed variational inference scheme applied to BasisVAE, noting the removal of the necessity to learn the translation and scale parameters δ and λ.

Before concluding this section with the aforementioned $h_{t_0}(t)$ proof, we emphasise that it is the derivative-based approach of DeVAE that has allowed us to specify monotonic components without having to adopt parametric constraints (e.g. linearity, sigmoid-like models) or neural network weight constraints. It has also allowed us to specify a general form of Gaussian-like transient component. The BasisDeVAE framework has then enabled the data-driven *learning* of cluster assignments, linking each feature with an *interpretable* behaviour cluster.

Proof: $h_{t_0}(t)$ satisfies $h_{t_0}(t_0) = 0$ and $h_{t_0}(t)$ monotonically increasing

Previously, we specified

$$G_{t_0}(t) = A \exp\big(-[h_{t_0}(t)]^2\big)$$

with i) $h_{t_0}(t_0) = 0$ and ii) $h_{t_0}(t)$ a monotonically increasing function to be a Gaussian-like transient and stated that a representation of the form

$$h_{t_0}(t) = (t - t_0) \times \text{softplus}\left(c + \int_0^t (\tau - t_0) f(\tau) \mathrm{d}\tau\right),$$

with $f(\tau)$ an integrable function with positive range, satisfies i) and ii). Here we formalise and prove this result.

Let $a < 0 < b$ and $t, t_0 \in (a, b)$. We would like to show that for any positive-range real function $f : (a, b) \mapsto \mathbb{R}_{>0}$ which is locally L_1 integrable, the function

$$h_{t_0}(t) = (t - t_0) \times \text{softplus}\left(c + \int_0^t (\tau - t_0) f(\tau) \mathrm{d}\tau\right) \tag{2.10}$$

satisfies i) $h(t_0) = 0$ and ii) $h_{t_0}(t)$ is monotonically increasing.

Condition i): Evaluating $h(t_0)$ gives

$$h_{t_0}(t_0) = \underbrace{(t_0 - t_0)}_{=0} \times \text{softplus}\Bigg(\underbrace{c + \int_0^{t_0} (\tau - t_0) f(\tau) \mathrm{d}\tau}_{:= g(t_0)}\Bigg),$$

implying that condition i) is satisfied iff $g(t_0)$ is finite. As c is finite, this is equivalent to the requirement that

$$I(t_0) = \int_0^{t_0} (\tau - t_0) f(\tau) \mathrm{d}\tau$$

is finite. As $t - t_0$ is absolutely continuous and $f(\tau)$ is locally L_1 integrable, integration by parts with $u(\tau) = \tau - t_0$ and $v(\tau) = \int_0^\tau f(x) \mathrm{d}x$ can be applied to obtain

$$I(t_0) = \underbrace{u(t_0)v(t_0)}_{=0} - \underbrace{u(0)v(0)}_{=0} - \int_0^{t_0} v(\tau) \mathrm{d}\tau,$$

which is finite as $v(\tau)$ is continuous due to its integral definition. ■

Condition ii): To prove that $h_{t_0}(t)$ is monotonically increasing, we would like to show that $\frac{\mathrm{d}}{\mathrm{d}t}h_{t_0}(t) > 0 \;\forall\; t, t_0 \in (a, b)$. Differentiating (2.10) gives

$$\begin{aligned}\frac{\mathrm{d}h_{t_0}}{\mathrm{d}t} = \text{softplus}&\left(c + \int_0^t (\tau - t_0) f(\tau)\mathrm{d}\tau\right) \\ &+ (t - t_0)^2 f(t)\text{softplus}'\left(c + \int_0^t (\tau - t_0) f(\tau)\mathrm{d}\tau\right).\end{aligned}$$

Noting that $f(t), \text{softplus}$ and $\text{softplus}'$ have positive range, the result follows. ■

2.4 Experiments[1]

2.4.1 Synthetic data

We first illustrate the performance of BasisDeVAE via synthetic data experiments. We generate a dataset with $N = 500$ samples and $d = 50$ features. Thirty features are randomly translated and scaled Gaussians, and the other 20 are randomly translated and scaled monotonic functions specified using softplus functions, with 10 positive and 10 negative (see Figure 2.5, upper-left panel). Each feature is also corrupted with Gaussian noise ($\sigma = 0.1$). Ground-truth pseudotimes were drawn uniformly to produce the data but withheld from analyses. Note that both models are well-specified with respect to this data.

Figure 2.5 shows a comparative analysis of BasisVAE and BasisDeVAE fits to the data. We see that BasisDeVAE faithfully recovers both the pseudotemporal trajectories and cluster assignments. In contrast, while BasisVAE captures the overall pseudotemporal behaviour, it loses accuracy at the extremities of the latent dimension and produces erroneous clustering.

[1]All computations were performed on a Linux (Ubuntu) desktop with an Intel i7-4790K 4GHz CPU and NVIDIA GTX 980 GPU (4GB VRAM).

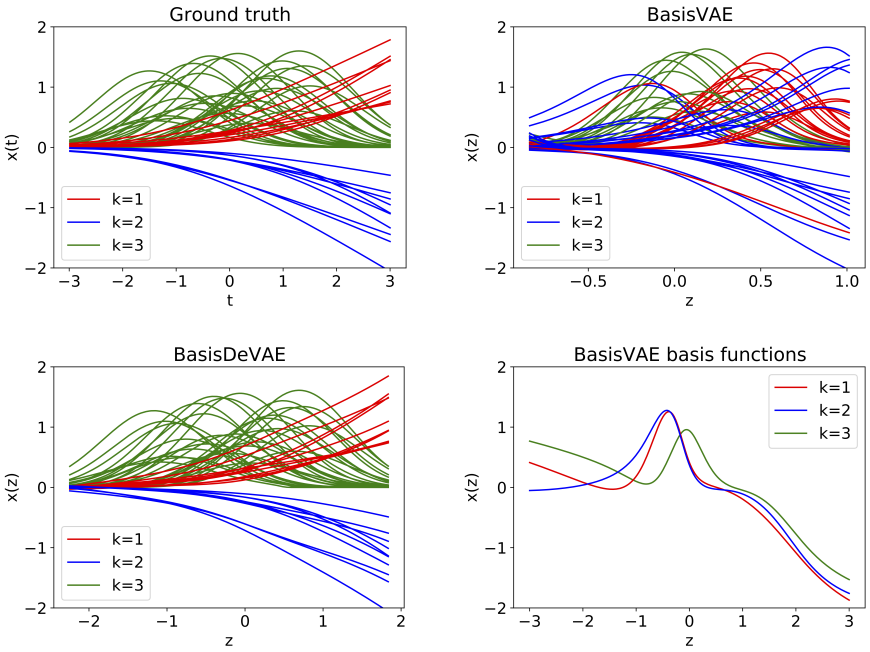

Figure 2.5: **Synthetic data.** BasisVAE and BasisDeVAE are applied to synthetic data with the ground-truth trajectories and clusterings shown in the upper-left panel. BasisDeVAE infers both feature behaviour and cluster assignments correctly (bottom left), whereas BasisVAE learns degenerate basis functions (bottom right), frustrating clustering performance (top right).

This behaviour is explained by examining the basis functions learnt by BasisVAE (Figure 2.5, bottom-right panel), which do not appear to properly distinguish the monotonic and transient characteristics, instead merging the behaviours in each of the clusters.

In real-world settings, one may not observe a set of samples which spans the entire range of pseudotime. Furthermore, the observed region may vary between experiments. It is therefore of interest to test whether clustering performance agrees across different pseudotemporal segments. As an empirical example, we segment the synthetic data into two

Table 2.1: **Synthetic data clustering performance.** Adjusted Rand Index (ARI) values associated with the clusterings learnt from the "low z" ($\mathcal{C}_{\text{l}}$) and "high z" ($\mathcal{C}_{\text{h}}$) datasets (see Section 2.4.1 text). GT represents the ground-truth clustering.

	BasisVAE	**BasisDeVAE (Ours)**
$\text{ARI}(\text{GT}, \mathcal{C}_{\text{L}})$	0.381	**0.524**
$\text{ARI}(\text{GT}, \mathcal{C}_{\text{H}})$	0.102	**0.455**
$\text{ARI}(\mathcal{C}_{\text{L}}, \mathcal{C}_{\text{H}})$	0.258	**0.280**

(overlapping) datasets. The first, referred to as "low z", contains only $\mathbf{x}$ values associated with $t < 1$ in the ground-truth dataset (plotted in the upper-left panel of Figure 2.5). The second, referred to as "high z", contains only $\mathbf{x}$ values associated with $t > -1$. Each dataset therefore totally excludes one-third of the complete trajectory. Table 2.1 shows the Adjusted Rand Index (ARI) values associated with the feature clusterings obtained by BasisVAE and BasisDeVAE on the "low z" and "high z" datasets, with GT denoting ground truth. It can be seen in Table 2.1 that the clusterings inferred by BasisDeVAE from the restricted data are more consistent with the ground truth than those of BasisVAE (rows 1 and 2), and that there is more consistency between each dataset's clusterings within the BasisDeVAE framework (row 3). These observations suggest that the presence of an underlying structured decoder model can improve extrapolation capability compared to the purely data-driven decoder approach of BasisVAE.

2.4.2 OASIS

The Open Access Series of Imaging Studies (OASIS) is a project led by the Knight Alzheimer Disease Research Center of Washington University to collect and openly release anonymised

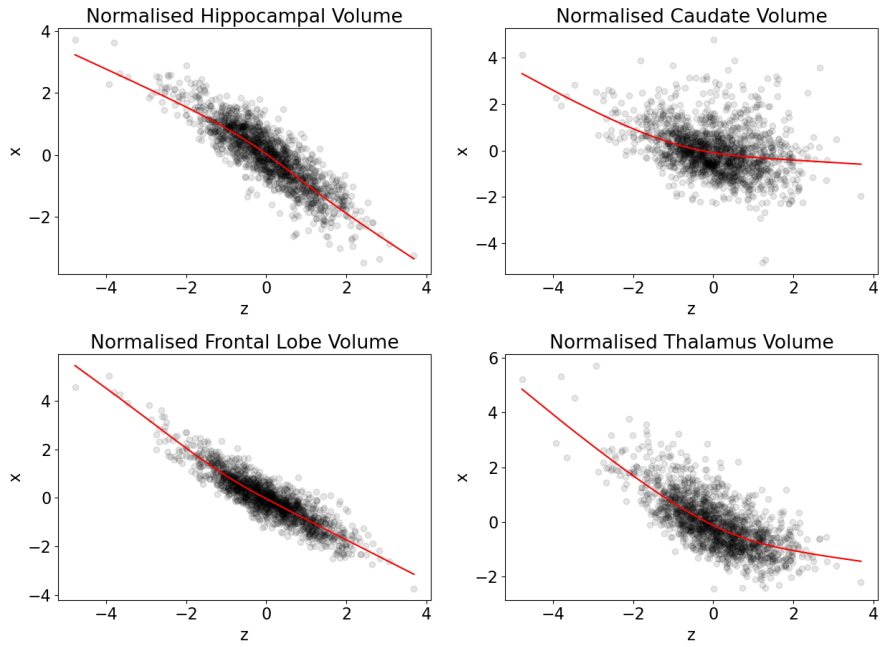

Figure 2.6: **OASIS regional brain volumes.** Inferred pseudotemporal profiles of 4 regional brain volumes. DeVAE extracts the monotonically decreasing pseudotemporal trajectories associated with cognitive decline.

patient data originating from a number of studies carried out there over the past 30 years. We use OASIS-3 (LaMontagne et al., 2019) which is the latest iteration of released data and contains entries from over 2,000 MRI sessions of patients at various stages of cognitive decline.

It is well known that cognitive decline is associated with the reduction of regional brain volumes (Fox and Schott, 2004). We would therefore like the decoder of a VAE-based pseudotime model applied to tabular regional brain volume data to be monotonically decreasing, as can be easily defined within our DeVAE framework.

To test whether the explicit specification of negative monotonicity aids performance, we extract the MRI sessions with associated FreeSurfer volume segmentations to create a tabular dataset of size $N = 2047$, $d = 13$ (see Appendix A.1 for details on the preparation of

Table 2.2: **OASIS performance metrics.** Test set predictive log-likelihood and ELBO values for the OASIS-3 data.

Method	$\log p(\mathcal{X}\|\mathcal{Z}, \theta^*)$	ELBO
Linear VAE	-4929.1	-5798.9
VAE	-4903.7	-5779.9
DeVAE (Ours)	$\mathbf{-4879.6}$	$\mathbf{-5749.6}$

the data and meaning of each feature) and train: i) a linear VAE, ii) a standard VAE and iii) a monotonically decreasing DeVAE. Each model is trained for 50 epochs using Adam (Kingma and Ba, 2015) with a 5×10^{-3} learning rate and employs a Gaussian conditional log-likelihood in the decoder. The neural networks within the VAE and DeVAE have the same architecture apart from the final layer of the DeVAE network applying a negative softplus to enforce negative monotonicity. The pseudotemporal trajectories inferred by DeVAE for features associated with atrophy in the hippocampus, caudate, frontal lobe and thalamus are shown in Figure 2.6.

We quantitatively evaluate the performance of each model by performing 10 train-evaluate iterations. Each iteration consists of i) randomly partitioning the data into an 80%/20% train/test split, ii) training on the training data and iii) evaluating two metrics on the test data. The first metric is the conditional log-likelihood $\log p(\mathcal{X}|\mathcal{Z}, \theta^*)$, where $\mathcal{X}$ is the test data, θ^* are the learnt decoder parameters and $\mathcal{Z} = \{z_i\}$ with z_i the posterior mean latent variable value associated with test point $\mathbf{x}_i$. The second metric is the ELBO, i.e. the quantity given in (1.4), evaluated over the test data, which acts as a lower bound on the marginal log-likelihood $\log p(\mathcal{X}|\theta^*)$. We report the mean value of each metric calculated across the 10 runs in Table 2.2, where it can be seen that DeVAE outperforms the linear VAE and VAE with respect to both metrics. DeVAE also had the highest metric on each of

the 10 runs.

2.4.3 Single-cell expression analysis

We next demonstrate the utility and scalability of our BasisDeVAE model by analysing a single-cell RNA sequencing (scRNA-seq) mouse spermatogenesis dataset Ernst et al. (2019) that was also used for testing BasisVAE in Märtens and Yau (2020). The data consists of the expression values of $d = 5,216$ genes measured across $N = 8,509$ cells. The analysis task on such data is to i) recreate a representation of the temporal variable via a one-dimensional latent pseudotime z and learn the gene expression profiles with respect to z, and ii) to group features with similar expression profiles into interpretable clusters (Figure 2.1). These two tasks can be performed simultaneously within the BasisVAE and BasisDeVAE frameworks.

As is common in scRNA-seq analysis, we utilise a zero-inflated negative binomial conditional likelihood model in the BasisVAE and BasisDeVAE decoder Risso et al. (2018); Lopez et al. (2018). For BasisVAE, we utilise the loss as described by (2.1)–(2.5) with $K = 3$. For BasisDeVAE, we replace the δ, λ in term (2.5) with the generalised Gaussians' z_0s and scale factors respectively but otherwise use the same loss. In both models we use $\beta = 10, \gamma = 1, \boldsymbol{\alpha} = \mathbf{0.1}$ and optimise using Adam with a 5×10^{-3} learning rate. We apply linear KL-annealing (Bowman et al., 2016b) to (β, γ) over the first 20% of 100 training epochs.

We have found that within the standard BasisVAE framework it is often necessary to place hand-tuned constraints on the values of the translation and scale parameters δ, λ in order to prevent collapse to a single basis function with regional behaviours. To demonstrate the effect of such constraints, we train two full BasisVAE models, one with the default constraint on λ specified within Märtens and Yau (2020)'s implementation, namely $\lambda_{jk} \in [0.25, 1.75]$,

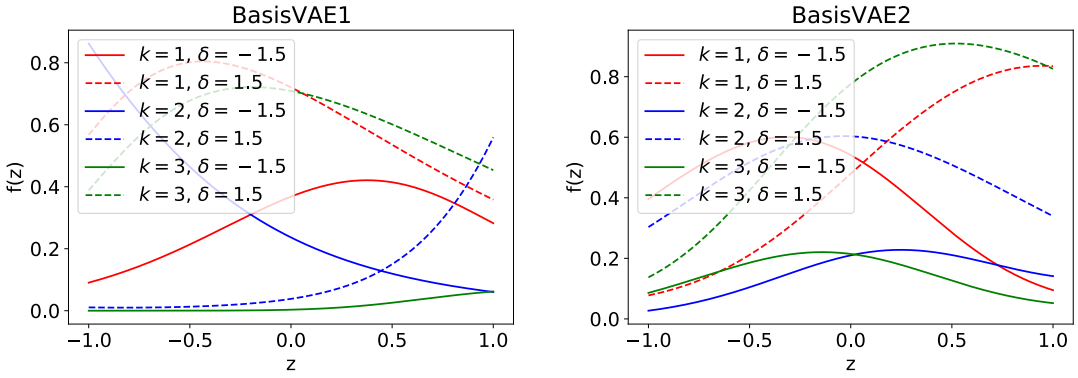

Figure 2.7: **Dependence of BasisVAE cluster predictions on translation parameter δ.** Variation of the appearance of BasisVAE1 (top) and BasisVAE2 (bottom) components with $\delta = 1.5$ (solid lines) and $\delta = -1.5$ (dashed). If the clustering were optimal, each plot would display qualitative similarity between same-colour lines and distinct behaviour between different-colour lines. This is not seen, supporting the observations made during synthetic data experiments that BasisVAE's basis functions can i) demonstrate multiple qualitatively different behaviours within a single basis function and ii) collapse such that each cluster's basis function has a very similar appearance.

which we denote BasisVAE1, and another with a looser $\lambda_{jk} \in [0.1, 3]$ condition, which we denote BasisVAE2. We also train BasisVAE with a linear network using the same hyperparameters and default constraints to serve as a baseline. We train on a randomly sampled 90% portion of the data and reserve the remaining 10% for test-set evaluation.

Using our derivative-based specification within BasisDeVAE allowed us to specify the meaning of each feature cluster, namely monotonically increasing/decreasing and Gaussian-like transient, making BasisDeVAE cluster behaviours immediately *interpretable*. Figure 2.7 shows that the interpretability of BasisVAE1 and BasisVAE2 clusterings is less clear. In particular, it shows the behaviour of inferred BasisVAE1 and BasisVAE2 components with

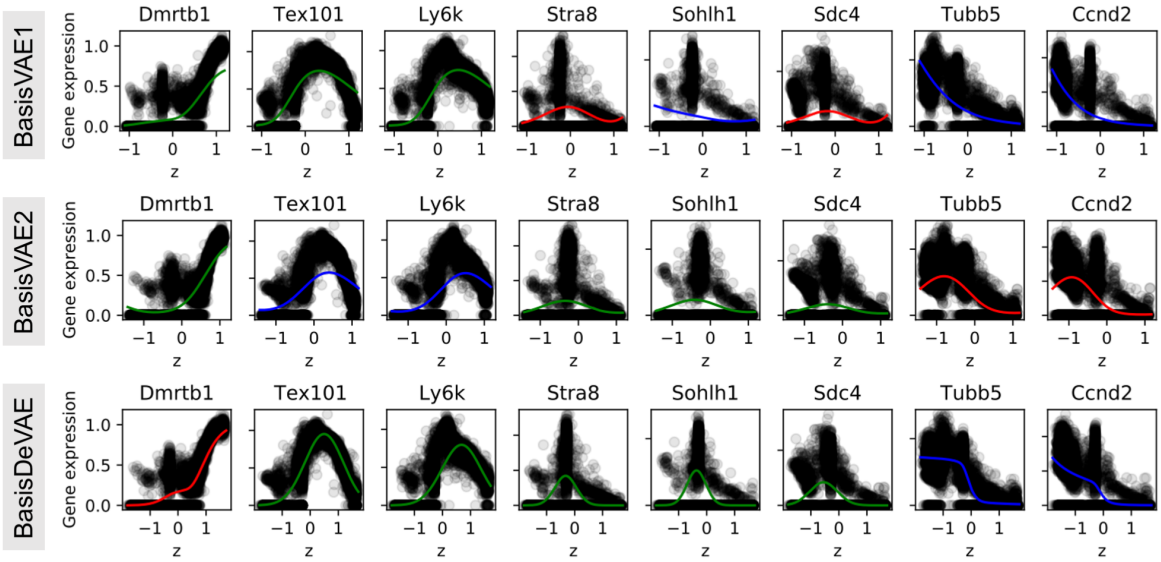

Figure 2.8: **Single-cell spermatogenesis data.** Inferred clustering and pseudotemporal gene expression trajectories of eight genes from Ernst et al. (2019) with BasisVAE1, BasisVAE2 and BasisDeVAE. BasisDeVAE provides a superior fit to observed data as well as greater cluster interpretability.

$\lambda = 1, \delta = \pm 1.5$. In the case of BasisVAE1, we see that changing δ leads to qualitatively different behaviour in components $k = 1$ and $k = 2$, meaning e.g. that both monotonic gene upregulation and monotonic gene downregulation could be attributed to component $k = 2$. Meanwhile, for BasisVAE2, we see a distinct qualitative similarity between each of the components, limiting the interpretability of a feature's cluster assignment.

Figure 2.8 visualises the pseudotemporal trajectories of 8 genes highlighted in Ernst et al. (2019) according to (from top to bottom) the BasisVAE1, BasisVAE2 and BasisDeVAE models. BasisDeVAE clearly clusters the 8 genes into the three distinct behaviours: monotonically increasing (red), transient (green) and monotonically decreasing (blue). It is less clear that the clusters identified by BasisVAE are necessarily plausible. For example, *Dmrtb1* is clustered with *Tex101* and *Ly6k* by BasisVAE1 but with *Stra8* and *Sohlh1* by BasisVAE2.

Table 2.3: **Single-cell data performance metrics.** Training time and model fit metrics for the simultaneous dimensionality reduction and feature-level clustering methods applied to the Ernst et al. (2019) scRNA-seq data.

Method	t_{train}/epoch (s)	$\log p(\mathcal{X}\|\mathcal{Z}, \theta^*)$	ELBO
Linear	7.42 ± 0.19	-2.55×10^6	-6.86×10^7
BasisVAE1	9.24 ± 0.15	-2.54×10^6	-6.82×10^7
BasisVAE2	9.32 ± 0.13	-2.54×10^6	-6.81×10^7
BasisDeVAE (Ours)	$\mathbf{4.67 \pm 0.04}$	$\mathbf{-2.51 \times 10^6}$	$\mathbf{-6.71 \times 10^7}$

In Table 2.3, we show the training time per epoch, predictive log-likelihood and ELBO for each of the VAE models. Our results indicate that BasisDeVAE is more computationally efficient than BasisVAE and is also able to achieve superior model fits in terms of log-likelihood and ELBO metrics. We also note that BasisDeVAE tends to converge more readily than BasisVAE, achieving BasisVAE-level metric scores after less than half of the 100 utilised training epochs. This is likely caused by the enforced behavioural separation of BasisDeVAE's clusters making the learning problem less difficult. The lower training time per epoch for BasisDeVAE can be explained by noting that in BasisVAE, the computation of $\lambda_{jk} f^{(k)}_{\text{basis}}(\mathbf{z}_i + \delta_{jk})$ for N datapoints, d features and K clusters requires NdK evaluations of a DNN with K outputs, compared with Nn evaluations of a dK-output DNN to compute the corresponding BasisDeVAE quantity.

2.5 Discussion

We introduced DeVAE and BasisDeVAE, two novel VAE models with decoders specified in terms of their derivatives with respect to the latent variable $\mathbf{z}$. We demonstrated that the derivative-based construction of the decoder employed in these models allows the specification of functional forms with both expressivity and interpretability, showing in particular how to specify monotonicity and transience in the context of pseudotemporal models of biological progression. We achieved state-of-the-art performance on both synthetic and real-world data examples for the problem of simultaneous dimensionality reduction and feature-level clustering.

Our work is complementary to a number of ongoing research areas. It has immediate links with other works which attempt to introduce additional structure into the VAE, such as the conditional VAE (Sohn et al., 2015), beta-VAE (Higgins et al., 2017) and functional ANOVA VAE (Märtens and Yau, 2020). Our demonstration of BasisDeVAE in the context of pseudotemporal analysis of single-cell data can be seen as a generalisation of Campbell and Yau (2018) capable of capturing any form of monotonic or Gaussian-like transient behaviour, not just sigmoidal or parametric Gaussian profiles. It also automatically assigns features to interpretable clusters without having to rely on pre-assigned genes as in Campbell and Yau (2018) or on a fully data-driven forward model as in Märtens and Yau (2020).

Our work can also be seen as another example of how the incorporation of derivative-based approaches into models can lead to improved utility and performance. The application of ODEs within machine learning has grown significantly since the widespread attention of the Neural ODE approach of Chen et al. (2018a), particularly in the contexts of irregularly sampled time-series modelling (Rubanova et al., 2019; Kidger et al., 2020b) and, more similarly to this work, in scientific machine learning Rackauckas et al. (2020). With significant volumes of related work continuing to emerge and increased attention on the development of associated

software (Rackauckas and Nie, 2017a; Bradbury et al., 2018b; Chen et al., 2018a), it is likely that this will remain an active area and may lead to further contributions related to ours in the context of VAEs.

The specification of functional constraints has typically been easier to adopt in a Gaussian Process (GP) framework due to the comparative ease with which one can specify functional properties relative to working in the (generally uninterpretable) weight space of a DNN. For example, Kazlauskaite et al. (2019) and Ustyuzhaninov et al. (2020) perform constrained-warp unsupervised learning of sequence alignments in a GP framework. However, our work here shows that by operating in the derivative space we can impose certain such functional constraints with relative ease within a DNN framework.

Natural extensions of this work include formulating derivative-based decoder specifications involving higher order derivatives, as well as the consideration of the case in which $\frac{\partial \mathbf{x}}{\partial z_p}$ is allowed to depend on features $\mathbf{x}$.

Chapter Three

Derivative-Based Neural Modelling of Cumulative Distribution Functions for Survival Analysis

This chapter presents a novel survival analysis approach which uses derivative-based modelling and neural networks to flexibly model the survival function, removing restrictions and approximations present in previous models. The presented approach (DeSurv) is also capable of modelling in the competing risks setting in a general way, something which is not widely considered in the literature. The core content of the chapter is published in the Proceedings of The 25th International Conference on Artificial Intelligence and Statistics (AISTATS 2022) — see Danks and Yau (2022).

This chapter, much like Chapter 2, is structured in the natural way to present the model and the corresponding experimental findings. It begins by covering why the model is necessary and how it is related to and improves other models (Section 3.1). It then introduces the idea of modelling Cumulative Distribution Functions (CDFs) via neural networks (DeCDF, Section 3.2), before detailing how this can be applied to the general survival analysis setting,

including competing risks (Sections 3.3 & 3.4). Various experiments are then presented which demonstrate the model's performance on synthetic and real-world data in the single and competing risks settings. The chapter is then concluded with a discussion of how the approach links with other research directions, how it is part of a growing movement to work with the exact likelihood in survival analysis and how it could be extended. We explore additional extensions in Chapter 4.

3.1 Introduction

3.1.1 Background

Survival analysis, also known as time-to-event analysis, is of fundamental importance to a wide variety of applied fields. In finance it is used to estimate default risk over time (Green and Shoven, 1986; Baesens et al., 2005; Dirick et al., 2017; Blumenstock et al., 2020), whilst in manafacturing and operational settings, estimating components' time-to-failure allows for maintenance planning and yield prediction (Ma and Krings, 2008; Susto et al., 2015; Ozturk et al., 2018). However, perhaps the most prominent application of survival analysis - and that which is the focus of this paper - is prognosis prediction in healthcare settings. In this context, the problem of interest is to predict the personalised survival distribution of a patient given their recorded covariates, in turn providing prognostic value to clinicians (Figure 3.1). Although survival analysis has long been applied to medical data, increased availability of healthcare data, in particular via personalised electronic health records (EHRs), has led to growing interest from the machine learning community.

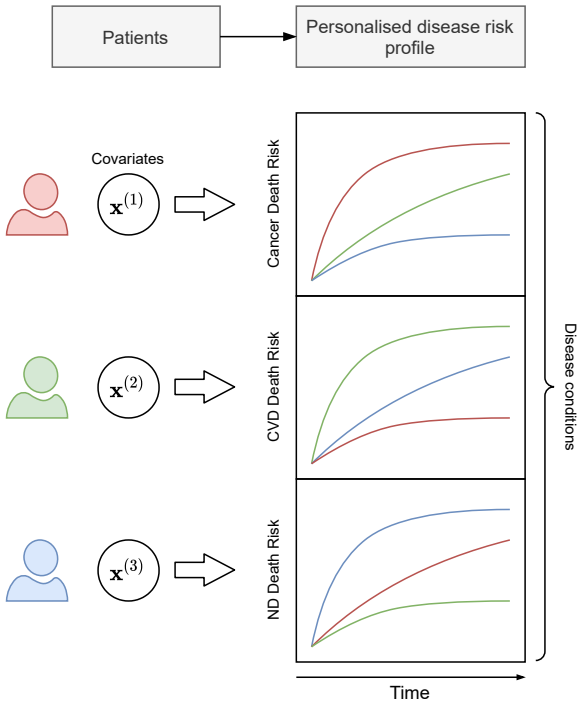

Figure 3.1: **Competing Risks Survival Analysis (CRSA).** Given a patient's covariates $\mathbf{x}^{(i)}$, CRSA provides a personalised risk profile for each disease event, for example death due to cancer, cardiovascular disease (CVD) or neurological disease (ND).

3.1.2 Motivation

In many real-world settings, patients often suffer from a number of diseases simultaneously (comorbidities) which all pose prognostic risk. To correctly capture survival behaviours in this context, these *competing risks* must be explicitly accounted for, however few survival analysis models actively address this problem. Furthermore, those that do tend to have undesirable aspects, e.g. misspecification issues (Austin et al., 2021), discrete time (Lee et al., 2018) or restrictive parameterisations (Fine and Gray, 1999).

Many of these aspects are introduced into the models due to the challenge of continu-

ously and generally modelling the survival functions (or equivalently cumulative distribution functions) involved. In this paper, we provide a method suited to this challenge by using the fact that the governing constraints on the functions of interest are naturally expressed via deep neural network-based ordinary differential equations. We then show how this in turn provides an unrestricted approach to competing risks survival analysis.

3.1.3 Contribution

Established approaches to survival analysis typically either i) do not generalise to competing-risk settings, ii) require the discretisation of time or iii) involve restrictive parameterisations. In this paper, we use the fact that survival analysis is an area in which flexible models of cumulative distribution functions (CDFs) are of critical importance in order to develop a new class of deep survival analysis model capable of dealing with competing risks in continuous time.

More specifically, our contributions are to: 1) Demonstrate how general time-to-event CDFs can be intuitively modelled via neural networks (NNs) and ordinary differential equations (ODEs) (DeCDF). (Sections 3.2.1–3.2.4). 2) Demonstrate an equivalence between DeCDF and certain normalising flow models. (Section 3.2.5). 3) Show how DeCDF can be immediately applied to allow for continuous-time, single-risk, deep learning-based survival analysis. (Section 3.3). 4) Show how DeCDF can be extended to model *cumulative incidence functions* (CIFs) and therefore perform competing-risk survival analysis (DeSurv). (Section 3.4). 5) Demonstrate our models in the single- and competing-risk settings using synthetic and real-world examples, showing strong empirical performance whilst allowing for both competing risks and continuous time within the same framework. (Section 3.5).

3.1.4 Related Work

In recent years, healthcare-based survival analysis has seen increased attention from numerous disciplines, including machine learning. Such ML-based methods have taken a variety of approaches to attempt to improve classical methods. One common continuous-time approach for single-risk data, instigated by Faraggi and Simon (1995), is to build upon the Cox proportional hazards (CPH) model (Cox, 1972). Faraggi and Simon (1995) replaced the linear log-risk function of the standard Cox model with a simple neural network but found insignificant performance benefit. Katzman et al. (2018) revisited this approach and found that by employing improved modern deep learning machinery, their similar model (DeepSurv) could outperform the linear CPH model. Kvamme et al. (2019)'s Cox-Time model retains the Cox formalism of DeepSurv (Katzman et al., 2018) but introduces non-proportionality in the hazard function by involving time in the log-risk function. Kvamme et al. (2019) also approximate the standard Cox partial likelihood loss to allow for greater compatibility with gradient descent-based learning. Yousefi et al. (2017) and Zhu et al. (2016, 2017) apply NN-based Cox methodology to genomic and imaging data respectively. Other works to apply ML methods to continuous-time single-risk survival data include piecewise-constant hazard models (Friedman, 1982; Fornili et al., 2014; Kvamme and Ørnulf Borgan, 2019), Deep Exponential Family models (Ranganath et al., 2016), Random Forest models (Breiman, 2001; Ishwaran et al., 2008; Mogensen et al., 2012; Dietrich et al., 2016).

Many survival analysis approaches discretise time into intervals, allowing the implementation convenient use of fixed and finite model outputs to compute risk at each designated time step. For example, the Logistic Hazard (Brown, 1975), PLANN (Biganzoli et al., 1998) and Nnet-Survival (Gensheimer and Narasimhan, 2019) models parameterise the conditional hazard (the probability of an event in an interval given survival up to that interval) at each time step in this manner. Meanwhile, the (N-)MTLR (Yu et al., 2011; Fotso, 2018) and

DeepHit (Lee et al., 2018) models target the probability mass function (PMF) for event occurrence directly.

Apart from DeepHit (Lee et al., 2018), the aforementioned models do not consider competing risks. Furthermore, proper treatment of competing risks is not typical in the wider survival analysis literature, and it has been noted that greater attention should be to this aspect of survival data (Koller et al., 2012). Whilst it is technically possible to apply single-risk methods to competing-risk data by considering only one event type a valid event and other event types as censoring, notable bias is introduced (Austin et al., 2016). The most prevalent approach to perform competing risks analysis is the Fine-Gray subdistribution hazard model (Fine and Gray, 1999; Austin et al., 2016), however, similarly to the standard Cox model, this involves restrictive assumptions on the hazard's functional form and its covariant dependence. Furthermore, the model has problematic aspects such as outputting potentially invalid overall time-to-failure CDFs capable of exceeding 1 (Austin et al., 2021). Some recent works have addressed competing risks from a modern ML point of view. For example, DMGPSA (Alaa and van der Schaar, 2017) expresses patient hitting times with respect to each event type via Deep Multi-Task Gaussian Processes (Damianou and Lawrence, 2013). Meanwhile, DeepHit (Lee et al., 2018) addresses competing risks by explicitly parameterising the joint PMF over failure time and event type using a large softmax-based neural network architecture.

The objective of our work (DeSurv) is to provide a natural, continuous time-based model that permits flexible modelling of survival functions whilst also allowing for both single- and competing-risk settings. It can therefore be thought of as an extension of DeepHit to incorporate continuous time, or as an extension of a method such as DeepSurv to incorporate competing risks.

3.2 Neural CDF Modelling: DeCDF

In this section, we describe how a cumulative distribution function can be naturally represented in terms of neural networks and ordinary differential equations.

3.2.1 Problem Setup

Let T be a random variable representing time to failure with support $\mathbb{R}_{\geq 0}$ and denote its probability density function (PDF) and cumulative distribution function (CDF) by $f(t)$ and $F(t)$ respectively. $F(t)$ is a valid CDF if it satisfies

1. $F(0) = 0$
2. $F(t + a) \geq F(t) \quad \forall a > 0$
3. $\lim_{t\to\infty} F(t) = 1$.

Condition 1 can be considered an initial condition, whilst condition 2 is a monotonicity condition. Both are naturally expressed in the language of ordinary differential equation (ODE) initial value problems (IVPs). More specifically, specifying $F(0) = 0$ as an initial condition and enforcing $\frac{\mathrm{d}F}{\mathrm{d}t} \geq 0$ in an IVP will ensure condition 1 and 2 are satisfied. The simultaneous satisfaction of condition 3 is a matter of practitioner choice.

3.2.2 DeCDF

Our approach, which we denote DeCDF, is to model an underlying strictly monotonic function $u(t)$ as an integral, i.e. to express its derivative in terms of t only, and then apply a suitable transformation to $u(t)$ to obtain $F(t)$. Perhaps the most immediate ML-aware example of

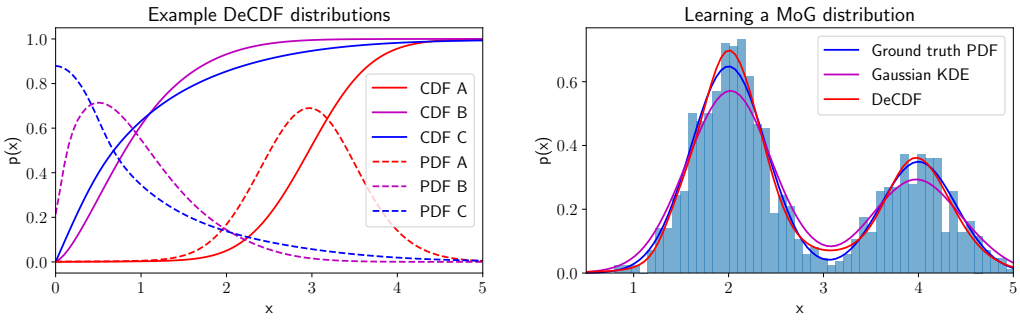

Figure 3.2: **Modelling Distributions with DeCDF.** DeCDF allows flexible modelling of general time-to-event distributions. Left: Example CDFs and corresponding PDFs modelled by DeCDF. Right: An example of DeCDF learning to model a rich distribution from data, in this case a Mixture of Gaussians (MoG) distribution.

this is to let $g(t)$ be a neural network with positive range and set $F(t) = H(u(t))$, where

$$\frac{\mathrm{d}u}{\mathrm{d}t} = g(t); \;\; H(x) = \tanh x \quad \text{and} \quad u(0) = 0. \tag{3.1}$$

Within this formalism, $\frac{\mathrm{d}u}{\mathrm{d}t}$ does not depend on u, only on t, hence $u(t)$ can be computed using Gauss-Legendre (GL) quadrature of order n (we use $n = 15$ throughout) via

$$u(t) = \frac{t}{2} \sum_{i=1}^{n} w_i g\left(\frac{t}{2}\left(u_i + 1\right)\right),$$

with u_i, w_i the order n GL quadrature nodes and weights respectively, computed via Legendre polynomials (see e.g. Press et al. (2007) for details). This computation has defined computational complexity and can be readily backpropagated and parallelised for efficient implementation in a variety of frameworks. Examples of DeCDF distributions are shown in the left panel of Figure 3.2. Details on the generation of this panel are given below in Section 3.2.4.

3.2.3 Distribution Estimation From Data

DeCDF in this form can be immediately employed to perform univariate density estimation. In this setting, DeCDF with a parameter setting ϕ takes in x and outputs a valid CDF $F_\phi(x)$. The density associated with this CDF is $f_\phi(x) = \frac{\mathrm{d}F_\phi}{\mathrm{d}x}$ which, assuming a DeCDF setup, can be calculated as $f_\phi(x) = (1 - F_\phi(x)^2)g_\phi(x)$, where $g_\phi(x)$ represents the underlying positive-output NN.

Given N independent and identically distributed samples $\mathcal{D} = \{x_i\}_{i=1}^N$ from a continuous distribution $\mathcal{P}$, the negative log-likelihood under the DeCDF model is

$$\mathcal{L}(\phi; \mathcal{D}) = -\left[\sum_{i=1}^{N} \log\left(1 - F_\phi(x_i)^2\right) + \log\left(g_\phi(x_i)\right)\right]. \tag{3.2}$$

Minimising this with respect to ϕ provides an estimate for the distribution $\mathcal{P}$.

3.2.4 CDF & PDF Estimation From Data via DeCDF: Example

To demonstrate how DeCDF can be used for distribution estimation in practice, we consider estimation of a Mixture of Gaussians distribution from sampled data (right panel of Figure 3.2).

Specifically, we draw 1000 samples from the following (Mixture of Gaussians) generative process

$$z_i \sim \text{Bernoulli}(p);\ x_i \sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i}^2),$$

with $p = 0.35, \mu_0 = 2, \mu_1 = 4$ and $\sigma_1 = \sigma_2 = 0.4$. The loss in (3.2) was then optimised for 1000 iterations using Adam (Kingma and Ba, 2014) with a 10^{-3} learning rate. The obtained DeCDF model has the density shown in the right panel of Figure 3.2, which can be seen to match the ground truth and Gaussian Kernel Density Estimate (KDE) well. The displayed Gaussian KDE is obtained via the default use of the `scipy.stats.gaussian_kde` function.

The left panel of Figure 3.2 shows additional examples of DeCDF distributions. To generate this figure, 1000 samples were drawn from each of 3 distributions, namely A) a Gaussian with $\mu = 3, \sigma = 0.6$, B) a Weibull with $k = 1.5, \lambda = 1$, and C) a Weibull with $k = 1, \lambda = 1$. The parameters of DeCDF were trained using the loss above to begin to emulate these distributions. The aim of this procedure was not to optimise for and test DeCDF's ability to emulate distributions, but rather to demonstrate illustrative example outputs which can arise from the model.

3.2.5 Normalising Flow Viewpoint

The form of (3.1) arises intuitively when considering how to model a CDF in terms of its derivatives, however it is natural to ask which functions could be used in place of tanh and whether there are any shortcomings in the generality of this specification. In fact, tanh can be replaced with any strictly increasing CDF on $\mathbb{R}_{\geq 0}$ and the method is able to model any well-behaved time-to-event distribution. This can be seen by considering the problem from the perspective of normalising flows.

Normalising flows (Rezende and Mohamed, 2016; Kobyzev et al., 2020; Papamakarios et al., 2021) model rich distributions of a random variable X by considering X to be the result of applying a learnt transformation $\mathcal{T}$ to an underlying random variable Z which follows a simple distribution. This in turn defines the density of X in terms of the density of Z and $\mathcal{T}$. When performing normalising flow analysis, one chooses to model $\mathcal{T}$ (the generative direction) or $\mathcal{T}^{-1}$ (the normalising direction). Modelling $\mathcal{T}$ allows for straightforward sampling of X, whilst modelling $\mathcal{T}^{-1}$ allows for straightforward density evaluation of X. It has been shown (see e.g. Papamakarios et al. (2021)) that normalising flows are capable of capturing arbitrarily complex X distributions using arbitrarily simple Z distributions.

To elucidate the link between normalising flows and DeCDF, first note that (3.1) implies a density on T of

$$p_t(t) = \frac{\mathrm{d}F}{\mathrm{d}t} = H'(u(t))\frac{\mathrm{d}u}{\mathrm{d}t} = H'(u(t))g(t). \tag{3.3}$$

Now let $p_z(z)$ denote the PDF of the random variable Z with support $\mathbb{R}_{\geq 0}$ and let $\mathcal{T}^{-1}(t) = u(t)$, with $u(t)$ as in (3.1). Applying change of variables provides the density on T as

$$p_t(t) = p_z(u(t))\frac{\mathrm{d}u}{\mathrm{d}t} = p_z(u(t))g(t). \tag{3.4}$$

Comparing (3.3) and (3.4), we see that $H'(u(t))$, that is the derivative of the transformation function in the DeCDF context, can be seen as the base distribution PDF in a normalising flow viewpoint. $H(x)$ can therefore be taken to be any strictly increasing CDF function on $\mathbb{R}_{\geq 0}$ and (3.1) remains valid. Meanwhile, the underlying monotonic function, $u(t)$, which is transformed by H in the DeCDF context can be seen as $\mathcal{T}^{-1}$ in normalising flow language. This means that $u(t)$ could be modelled via established normalising flow invertible functions to achieve similar functionality to our approach. We return to this point in the discussion as a recommendation for future work. Furthermore, the above relationship demonstrates the general expressivity of DeCDF and hence its suitability for flexible time-to-event distribution modelling.

3.2.6 Alternative Derivative-Based Parameterisations

Above, we present DeCDF as the natural way to employ derivative-based methodology to CDF modelling. However, there exist other derivative-based approaches which could be investigated.

One immediate suggestion, which we refer to as AutoCDF due to the autonomous nature of the governing ODE, is to let $g(F)$ be a function with range $\mathbb{R}_{>0}$ (e.g. a neural

network with appropriate output activation) and let

$$\frac{\mathrm{d}F}{\mathrm{d}t} = (1 - F)g(F) \quad \text{with} \quad F(0) = 0. \tag{3.5}$$

The main drawback of this approach compared to DeCDF is that the computation of $F(t = t^*)$ via (3.5) requires a full ODE solve (not just t-domain quadrature) from $t = 0$ to $t = t^*$. Whilst not infeasible, for $g(F)$ a neural network ODE solves can introduce implementational and computational complexity. Using adaptive ODE solvers results in a forward operation with an *a priori* unknown computational complexity which is not readily parallelisable. Furthermore, we will require gradients of this forward operation, and whilst these can be estimated using the adjoint method reintroduced by Chen et al. (2018b), this approach does not necessarily compute good approximate gradients (Gholami et al., 2019). Moreover, popular open-source implementations do not typically offer both robust neural network functionality and fully autodiff-aware ODE solvers able to reliably calculate gradients in all settings (see e.g. Rackauckas et al. (2018) for method comparisons). The primary framework at the time of writing which attempts this is Julia's SciML initiative (Rackauckas and Nie, 2017c) and is under ongoing development. These issues must be considered if modelling via (3.5) is performed.

3.3 DeCDF for Single-Risk Survival Analysis

3.3.1 Problem Setting

Survival analysis in the presence of a single risk considers data which for N subjects takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$, where $s^{(i)}$ denotes time-to-event, $\mathbf{x}^{(i)}$ denotes subject covariates and $k^{(i)}$ denotes event type. In the single-risk setting, $k^{(i)} \in \{\varnothing, 1\}$, with $k^{(i)} = 1$ implying the event of focus (e.g. death) occurred at $s^{(i)}$ and $k^{(i)} = \varnothing$ implying right censorship, that

is the subject being lost to follow up, often for an unknown reason. Given training data $\mathcal{D}_{\text{train}} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$, the goal is to learn a model which is capable of providing accurate *personalised* prognostic cumulative distribution functions of the survival time for test subjects. That is, given covariates of a test subject, $\mathbf{x}$, be able to provide an accurate estimate of $F(t|\mathbf{x}) = P(T \leq t|\mathbf{x})$.

3.3.2 Applying DeCDF

Given DeCDF's ability to model CDFs, we can directly utilise its methodology for this modelling purpose by simply adding $\mathbf{x}$ as an additional input to the model. More specifically, we let $g_\phi(\mathbf{x}, t)$ be a neural network with parameters ϕ and positive output range (enforced via softplus) and set $\hat{F}(t|\mathbf{x}) = \tanh(u(t|\mathbf{x}))$, where

$$\frac{\mathrm{d}u}{\mathrm{d}t} = g_\phi(\mathbf{x}, t) \quad \text{with} \quad u(0) = 0.$$

Note that, by the argument of Section 3.2.5, tanh could be replaced by any valid CDF on $\mathbb{R}_{\geq 0}$.

The negative log-likelihood for data $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$ and parameters ϕ is

$$\mathcal{L} = -\sum_{i=1}^{N} \Big[\mathbb{1}\left(k^{(i)} = \varnothing\right) \log\left(1 - \hat{F}(s^{(i)}|\mathbf{x}^{(i)})\right) + \mathbb{1}\left(k^{(i)} \neq \varnothing\right) \log\left(\hat{F}'(s^{(i)}|\mathbf{x}^{(i)})\right) \Big].$$

We use this as our training loss and apply minibatch gradient descent via Adam (Kingma and Ba, 2014) to optimise the neural network parameters ϕ.

3.4 Extending to Competing Risks: DeSurv

3.4.1 Problem Setting

In the competing risks setting, survival data again takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$, but now $k^{(i)} \in \{\varnothing, 1, 2, \ldots, K\}$ for K event types of interest. Typically, these event types code for "death due to cause k". For example, in a study of cancer patients, some participants may suffer from comorbidities such as cardiovascular disease (CVD) and may die due to these *competing risks* rather than cancer itself.

Whilst the overall survival time CDF, $F(t|\mathbf{x})$, is of interest in the competing risks context, the quantity of greatest interest is the cumulative incidence function (CIF), $F_k(t|\mathbf{x})$, associated with each event type. This is defined as $F_k(t|\mathbf{x}) = P(T \leq t, k|\mathbf{x})$, and therefore encodes the probability of experiencing event k before time t *accounting for the presence of competing risks*. Marginalising out the event type provides the overall survival time CDF as $F(t|\mathbf{x}) = \sum_{k=1}^{K} F_k(t|\mathbf{x})$.

3.4.2 DeSurv

DeCDF can be adapted to the competing risks setting by noting that each cumulative incidence function is a scaled CDF as $F_k(t|\mathbf{x}) = P(T \leq t, k|\mathbf{x}) = P(T \leq t|k, \mathbf{x})P(k|\mathbf{x})$ where the conditional CDF, $P(T \leq t|k, \mathbf{x})$, which we denote $\tilde{F}_k(t|\mathbf{x})$, is a valid CDF and $P(k|\mathbf{x})$, which we denote $\pi_k(\mathbf{x})$, satisfies $\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1$. The CIF can therefore be captured by simultaneously modelling $\{\tilde{F}_k\}_{k=1}^{K}$ via a DeCDF approach and $\{\pi_k(\mathbf{x})\}_{k=1}^{K}$ via a neural network with softmax activation. We refer to this model as DeSurv. The architecture for DeSurv is shown in Figure 3.3.

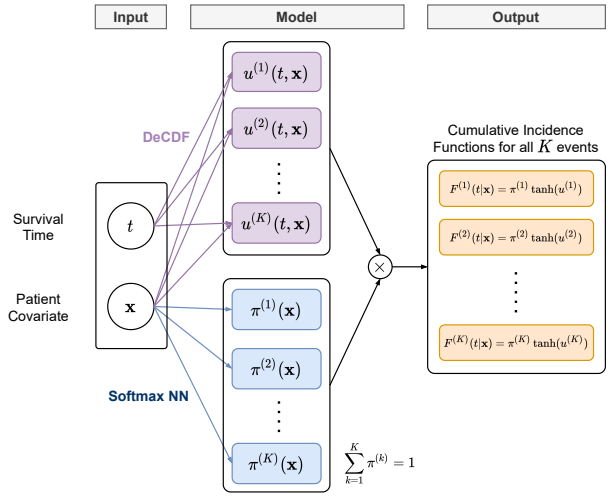

Figure 3.3: **DeSurv Model Architecture.** DeSurv flexibly models the cumulative incidence function of each event by performing elementwise multiplication between the outputs of a DeCDF block and a softmax neural network block. In the absence of competing risks, only the DeCDF block is required.

The negative log-likelihood for data $\mathcal{D} = \{s^{(i)}, \mathbf{x}^{(i)}, k^{(i)}\}_{i=1}^{N}$ and parameters ϕ is

$$\mathcal{L} = -\sum_{i=1}^{N} \left[\mathbb{1}\left(k^{(i)} = \varnothing\right) \log\left(1 - \sum_{k=1}^{K} F_k(s^{(i)}|\mathbf{x}^{(i)})\right) + \mathbb{1}\left(k^{(i)} \neq \varnothing\right) \log\left(F'_{k^{(i)}}(s^{(i)}|\mathbf{x}^{(i)})\right) \right].$$

As in the single risk case, we take this to be our loss and optimise via Adam (Kingma and Ba, 2014) to learn the parameters ϕ of the neural networks describing the $\{\tilde{F}_k\}_{k=1}^{K}$ and $\{\pi_k\}_{k=1}^{K}$ mappings.

3.5 Experiments

3.5.1 Reproducibility

In this section we demonstrate the behaviour of DeSurv and other related approaches on real and synthetic datasets from single-risk and competing-risk settings. For fair comparison, we utilise the same neural network architectures for each method as far as possible and provide all hyperparameter settings in the supplementary material. We attach a complete PyTorch-based (Paszke et al., 2019b) implementation of our DeSurv approach along with the code used to assess our model in the context of related methods. We also include data and licenses in the supplementary material where it is permitted. Train/Validation/Test splits are 64%/16%/20% in all cases. Table entries are provided as mean $\pm$ std and are calculated by re-running experiments (including data partitioning) in their entirety 10 times, each time with a new random seed drawn from a seeded RNG. The computational requirements of all involved algorithms were relatively low, with all experiments able to run within hours on a 2.7 GHz Quad-Core i7 processor.

3.5.2 Single-Risk Survival Analysis

We evaluate our model in the single-risk setting alongside various single-risk benchmarks discussed in section 3.1.4 on two anonymised real-world medical datasets: METABRIC and SUPPORT.

Table 3.1: **Survival Datasets.** Details of the datasets used to demonstrate model performance.

Dataset		Observed	Censored	Features	Event time		Censoring time	
					Mean	Max	Mean	Max
METABRIC		1103 (58%)	801 (42%)	9	100	355	160	337
SUPPORT		6036 (68%)	2837 (32%)	14	6.85	64.8	35.3	67.6
Synth	Cause 1	5699 (19%)	15000 (50%)	12	0.080	0.988	0.043	1.28
	Cause 2	9301 (31%)			0.086	1.11		
SEER	BC	116170 (19%)	442685 (74%)	23	57.8	311	112.1	311
	CVD	39639 (7%)			95.4	311		

Datasets

The METABRIC dataset contains gene expression profiles and clinical covariates of breast cancer (BC) patients for the purposes of BC subgroup identification. We follow the practice of Katzman et al. (2018), and utilise 9 features, namely 4 gene indicators (*MKI67*, *EGFR*, *PGR*, and *ERBB2*) and 5 clinical covariates (hormone treatment indicator, radiotherapy indicator, chemotherapy indicator, ER-positive indicator and age at diagnosis). SUPPORT (Knaus et al., 1995) is a larger dataset derived from a prognostic study of seriously ill patients and contains 14 covariates. We prepare the dataset as in Katzman et al. (2018). Further dataset details are provided in Table 3.1.

Table 3.2: **METABRIC Performance Metrics.**

Method	$\mathbf{C_{td}}$	**IBS**	**INBLL**	**MAE**	**RMSE**
Cox	0.621 ± 0.035	0.176 ± 0.009	0.523 ± 0.021	75.60 ± 4.86	87.76 ± 4.86
Cox-CC	0.634 ± 0.012	0.177 ± 0.007	0.522 ± 0.017	74.15 ± 4.54	86.99 ± 4.10
DeepSurv	0.634 ± 0.019	0.175 ± 0.006	0.519 ± 0.014	74.45 ± 4.65	87.36 ± 4.98
Cox-Time	0.664 ± 0.019	0.175 ± 0.005	0.518 ± 0.016	66.59 ± 2.90	78.60 ± 3.07
DeepHit	$\mathbf{0.673 \pm 0.013}$	0.188 ± 0.006	0.549 ± 0.013	88.3 ± 5.05	100.11 ± 4.90
DeSurv (Ours)	0.660 ± 0.022	$\mathbf{0.172 \pm 0.008}$	$\mathbf{0.509 \pm 0.020}$	$\mathbf{54.76 \pm 2.14}$	$\mathbf{65.5 \pm 2.44}$

Metrics

We evaluate the models on each test set using five metrics: i) Time-dependent Concordance Index (Antolini et al., 2005) (C_{td}), ii) Integrated Brier Score (IBS), iii) Integrated Negative Binomial Log-Likelihood (INBLL), iv) Mean Absolute Error (MAE) and v) Root Mean Square Error (RMSE). Full mathematical definitions and detailed explanations of these metrics are provided in Section 1.3.5 of Chapter 1. We evaluate the metrics using the `pycox` library (Kvamme et al., 2019).

3.5.3 Competing Risks Survival Analysis

We evaluate DeSurv in the competing risks setting using the methodology of Section 3.4 on real and synthetic data alongside the state-of-the-art deep learning-based competing risks model, DeepHit (Lee et al., 2018), which provides discretised time output.

Table 3.3: **SUPPORT Performance Metrics.**

Method	$\mathbf{C}_{\text{td}}$	**IBS**	**INBLL**	**MAE**	**RMSE**
Cox	0.562 ± 0.005	0.211 ± 0.003	0.609 ± 0.006	8.05 ± 0.17	10.86 ± 0.31
Cox-CC	0.600 ± 0.007	0.200 ± 0.004	0.588 ± 0.013	8.11 ± 0.16	10.85 ± 0.29
DeepSurv	0.603 ± 0.006	0.199 ± 0.004	0.585 ± 0.013	8.09 ± 0.17	10.85 ± 0.29
Cox-Time	0.612 ± 0.008	0.200 ± 0.004	0.586 ± 0.010	7.94 ± 0.21	10.84 ± 0.32
DeepHit	0.619 ± 0.010	0.216 ± 0.002	0.622 ± 0.004	20.37 ± 1.08	21.73 ± 1.14
DeSurv (Ours)	$\mathbf{0.622 \pm 0.010}$	$\mathbf{0.199 \pm 0.003}$	$\mathbf{0.583 \pm 0.010}$	$\mathbf{7.86 \pm 0.17}$	$\mathbf{10.75 \pm 0.22}$

Datasets

The Surveillance, Epidemiology, and End Results Program (SEER) dataset contains anonymised survival data from cancer patients in the United States. We extract breast cancer (BC) patients from the database and select those patients that either i) were right censored, ii) died from BC or iii) died from cardiovascular disease (CVD) (see Appendix B.1 for further details), yielding data with censorship and two competing risks (see Table 3.1).

We also generate a synthetic dataset of the style of that in Lee et al. (2018) with $N = 30,000$ examples drawn from the generative process

$$\begin{aligned}
&\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \mathbf{x}_3^{(i)} \sim U[\mathbf{0}, \mathbf{1}] \\
&T_k^{(i)} \sim \text{Exp}\left(\left(\boldsymbol{\gamma}_3^T \mathbf{x}_3^{(i)}\right)^2 + \boldsymbol{\gamma}_k^T \mathbf{x}_k^{(i)}\right) \equiv \text{Exp}\left(\lambda_k^{(i)}\right),
\end{aligned}$$

for $k = 1, 2$ with $(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \boldsymbol{\gamma}_3) = (\mathbf{0.5}, \mathbf{2}, \mathbf{1})$ and applied random censoring to 50% of examples (see Table 3.1).

Table 3.4: **Competing Risks Performance Metrics.**

	SEER		**Synthetic**	
Method	$\mathbf{C_{td}}^{(BC)}$	$\mathbf{C_{td}}^{(CVD)}$	$\mathbf{C_{td}}^{(1)}$	$\mathbf{C_{td}}^{(2)}$
DeepHit	$\mathbf{0.844 \pm 0.001}$	$\mathbf{0.885 \pm 0.002}$	0.611 ± 0.009	0.570 ± 0.004
DeSurv	0.826 ± 0.002	0.879 ± 0.002	$\mathbf{0.629 \pm 0.009}$	$\mathbf{0.587 \pm 0.005}$

Table 3.5: **Additional Competing Risks Metrics.**

Dataset	Method	$\text{RMSE}^{(1)}$	$\text{RMSE}^{(2)}$	BCE	Acc.
SEER	DeepHit	97.86 ± 6.28	103.65 ± 5.77	1.330 ± 0.004	75.01 ± 0.83
	DeSurv	$\mathbf{47.46 \pm 0.53}$	$\mathbf{76.82 \pm 1.17}$	$\mathbf{1.150 \pm 0.006}$	$\mathbf{79.73 \pm 0.33}$
Synthetic	DeepHit	0.0956 ± 0.0030	0.1032 ± 0.0013	1.396 ± 0.001	$\mathbf{62.02 \pm 1.04}$
	DeSurv	$\mathbf{0.0944 \pm 0.0022}$	$\mathbf{0.1013 \pm 0.0016}$	$\mathbf{1.389 \pm 0.003}$	61.78 ± 0.91

Metrics

We evaluate the models using Time-dependent Concordance Index ($\mathbf{C_{td}}$), Integrated Brier Score (IBS), Integrated Negative Binomial Log-Likelihood (INBLL), Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), all as defined in Section 1.3.5. In the competing risks case we also consider the binary cross entropy (BCE) and accuracy (Acc.) values associated with the task of predicting which event type occurred for those that experienced an event.

3.5.4 Results Discussion

Tables 3.2 & 3.3 show that DeSurv outperforms alternative single-risk methods overall on both the METABRIC and SUPPORT datasets. DeepHit does demonstrate a higher concordance than DeSurv on METABRIC and a comparable concordance on SUPPORT, however its performance is notably poorer with respect to other metrics, suggesting poor calibration. Visual inspection of the survival functions associated with DeepHit relative to those associated with the performant Cox-Time and DeSurv models confirms this (see Figure 3.4). In particular, it can be seen that whilst the models generally agree on the ranking of patient risk (reflected in concordance values), the survival functions associated with DeepHit exhibit a lack of patient-level variation relative to those of DeSurv and Cox-Time, leading to poor calibration and the large MAE and RMSE metrics observed in Table 3.2. This highlights the importance of looking beyond solely concordance values during model evaluation and comparison.

DeSurv also extends successfully to the competing-risk setting, outperforming DeepHit on the synthetic dataset and performing competitively with DeepHit with respect to concordance on SEER. In Appendix B.2, we also demonstrate that increasing the density of DeepHit's discretisation does not significantly improve its performance on the synthetic data. Visualisations of SEER patient CIFs suggest that DeepHit's tendency to lack trajectory separation is also present in the competing risk case, whereas DeSurv captures a variety of risk patterns (Figure 3.5). Additional visual comparisons of survival functions and CIFs across the models and datasets are provided in Appendix B.3. Further insight can be gained via the additional evaluation metrics shown in Table 3.5, namely the RMSE for each of the competing risks as well as the binary cross entropy (BCE) and accuracy (Acc.) values associated with the task of predicting which event type occurred for those that experienced an event. DeSurv again demonstrates a strong calibration performance relative to DeepHit,

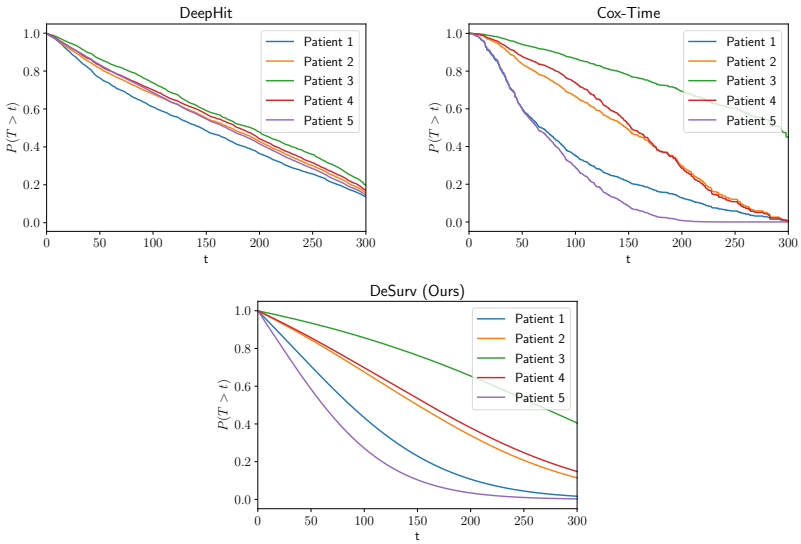

Figure 3.4: **METABRIC Survival Functions.** 25-year survival trajectories for five random test patients according to three of the considered models. Note that whilst the models agree on patient risk ranking, DeepHit has limited trajectory separation.

particularly on SEER, despite demonstrating similar concordance performance, again highlighting the importance of considering a variety of metrics when evaluating overall survival model performance.

3.6 Discussion

We have shown how general time-to-event CDFs can be modelled naturally in the language of derivative-based models, yielding a general model for such functions, DeCDF. We have also demonstrated how DeCDF can be applied immediately to single-risk survival analysis and developed a novel DeCDF-based survival analysis method, DeSurv, capable of flexibly modelling competing-risk survival data in continuous time. DeSurv can be seen as an extension of DeepHit (Lee et al., 2018) to continuous time or an extension of work such as

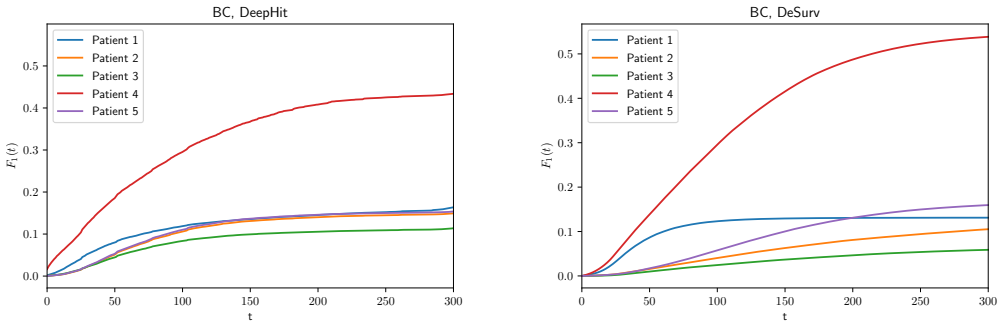

Figure 3.5: **SEER CIFs.** 25-year cumulative incidence functions for five random test patients according to DeepHit and DeSurv. As in the single-risk case (Figure 3.4), whilst the models agree on patient risk ranking DeepHit has limited trajectory separation.

DeepSurv (Katzman et al., 2018) and Cox-Time (Kvamme et al., 2019) to the competing-risk setting.

Our work also highlights a link between DeCDF and normalising flows, namely that the ODE solution and transformation function in DeCDF can be thought of as a normalisation direction transformation and base CDF respectively in normalising flow language (see Section 3.2.5). In this paper, we drew upon this equivalence to reinforce the expressivity of DeCDF. A natural extension of our work would be to explore this link further, for example by investigating the effect of flow-based transformation parameterisation and base distribution on performance. Another possible avenue for further work would be to investigate alternative ODE-based CDF parameterisations, an example of which is provided in Section 3.2.6. It would also be of interest to apply DeCDF to CDF-based problems in other application settings.

We also note that whilst we and others stress the importance of predictive performance of survival analysis models in prognostic settings, it is also of interest to understand how the personalised survival functions output by our model and others such as DeepHit (Lee et al.,

2018) depend on covariates and hence which covariates are most associated with disease prognosis. The required sensitivity analysis for this work can be readily carried out on DeSurv and other deep learning models via automatic differentiation and would also represent a valuable contribution.

Limitations and potential negative societal consequences: A key potential use case for deep survival analysis models such as DeSurv is to provide prognostic information to clinicians making critical decisions. Applying poorly performing models in this context would clearly adversely affect patient care and would represent a particularly negative social consequence. Hence, significant validation testing *including human-computer interaction elements* would have to be performed on any model hoping to deploy in a clinical setting. Additionally, whilst a significant benefit of models such as DeSurv is their ability to learn complex dependencies between patient covariates and event risk profiles from data, this can also act as a limitation. In particular, if trained blindly on well-curated retrospective data, these models will inherit any *biases* which may be present, meaning they may not generalise well to the everyday clinical setting. When training deployment-stage models, training should be carried out in a bias-aware way in order to avoid systemic algorithmic inequality such as the racial bias recently shown to exist in a U.S. healthcare management algorithm (Obermeyer et al., 2019).

Chapter Four

Neural ODE-Based Longitudinal Survival Analysis

This chapter considers the problem of longitudinal survival analysis, that is the modelling of the time-to-event given multiple observations of each patient over time, from a machine learning perspective. In doing so, it introduces three novel approaches to the problem which introduce various benefits over typically employed methods. The methods presented in this chapter were inspired by working within the *OPTIMising therapies, discovering therapeutic targets and AI assisted clinical management for patients Living with complex multimorbidity* study (OPTIMAL) project, led by Prof. Krishnarajah Nirantharakumar (University of Birmingham) and are currently being used within the project. During the PhD, work was also carried out to apply standard joint longitudinal-survival models to prostate cancer data and was published as Stavrinides et al. (2021). This is not included in this chapter due to its fully applied rather than methodological focus.

We begin the chapter by introducing the context for our work and the contributions it provides (Section 4.1). We then present the challenges associated with hazard-based modelling in the longitudinal setting (Section 4.2). Following this, in Section 4.3, we introduce three

ways in which a traditional longitudinal survival model can be extended: In Section 4.3.1, we show how DeSurv can be included within the joint longitudinal-survival model formalism to increase the flexibility of the survival component; In Section 4.3.2, we show how latent neural ODEs can be utilised to avoid the scaling limitations of the standard joint model formalism; In Section 4.3.3, we show how discriminative modelling of the survival time can be performed using recurrent architecures and DeSurv, in turn simplifying inference relative to random effects-based models. We then present experiments involving these models on synthetic and real-world data in Section 4.4 before concluding with a discussion of the key contributions of the work and how they can be and are being extended (Section 4.5).

4.1 Introduction

4.1.1 Motivation

In much of the survival analysis literature, the most canonical form of survival analysis is considered, within which it is assumed that each patient provides a measurement at baseline and predictions are made given this information. This is a useful form of survival analysis — for example, it may be useful to estimate a prognosis for a patient given baseline information upon admission to hospital. It is also highly applicable in the context of clinical trials, where treatment protocols are assigned at baseline and recorded alongside clinical measurements — it is indeed common for such methods to be used to statistically argue for or against the efficacy of a given drug.

However, in modern health settings it is common to regularly record patient measurements over time with the aim of complementing or updating a patient's prognosis assessed upon admission or entry into a study. This setting, referred to as longitudinal survival analy-

sis (see Figure 4.1), is becoming increasingly important as healthcare becomes increasingly data-driven but remains underexplored by the machine learning community. In this work, we approach longitudinal survival analysis from a machine learning perspective and utilise the links between generative latent variable modelling, as often considered in machine learning settings, and longitudinal survival models to introduce novel approaches capable of extending the models currently considered in the literature.

4.1.2 Background

In the longitudinal survival analysis setting, data takes the form $\mathcal{D} = \{s^{(i)}, \mathbf{x}_0^{(i)}, \mathbf{X}(\mathbf{t})^{(i)}, k^{(i)}\}_{i=1}^N$, where $s^{(i)}$ denotes time-to-event, $\mathbf{x}_0^{(i)}$ denotes subject covariates at baseline, $\mathbf{X}(\mathbf{t})^{(i)} \in \mathbb{R}^{d \times N_i}$ denotes subject covariate measurements $\mathbf{x}(t)$ at times $\mathbf{t} = [t_0, \ldots, t_{N_i}]$, and $k^{(i)} \in \{\varnothing, 1\}$ denotes event type (assuming a single-risk setting). In this work, we consider the primary target variable of interest to be the event time s, i.e. we would like to use covariates $\mathbf{x}_0, \mathbf{X}(\mathbf{t})$ to model survival time.

The canonical approach to performing survival analysis of data of the form $\mathcal{D}$ is the joint longitudinal-survival model. This form of model targets the joint distribution over longitudinal and survival variables, typically by choosing a longitudinal model for $\mathbf{x}(t)$ and a survival model for s, each of which may involve parameters $\boldsymbol{\theta}$, and then coupling these by assuming shared dependence on a set of random effects $\mathbf{b}$ (see Section 1.3.4 for more details). A linear mixed model of the form

$$x_i(t) = \boldsymbol{\beta}^{\mathrm{T}} \mathbf{u}_i(t) + \mathbf{b}_i^{\mathrm{T}} \mathbf{z}_i(t) + \varepsilon_i(t)$$

is often assumed for the longitudinal part (with one-dimensional x for simplicity), whilst a Cox model of the form

$$h_i(t) = h_0(t) \exp\left\{\boldsymbol{\gamma}^{\mathrm{T}} \mathbf{x}_0 + \alpha^{\mathrm{T}} m_i(t)\right\}, \tag{4.1}$$

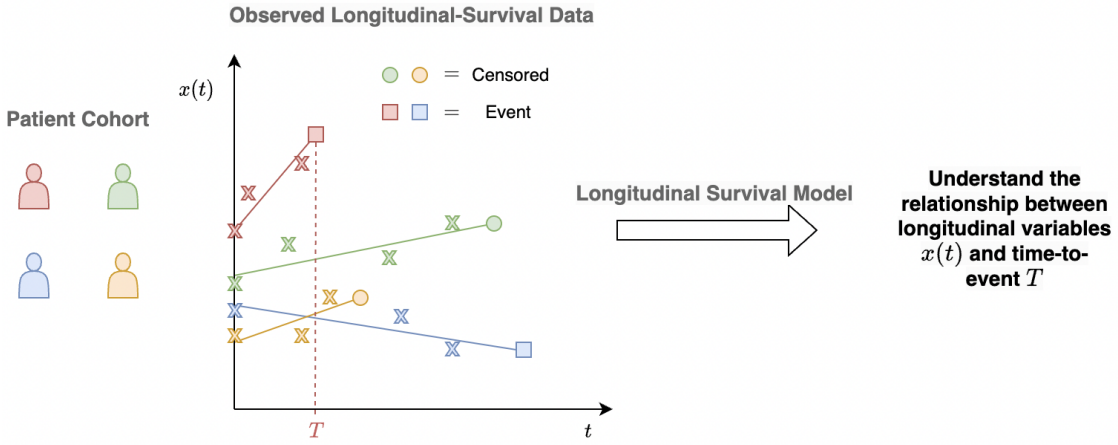

Figure 4.1: **Longitudinal Survival Analysis.** Longitudinal survival analysis aims to understand the relationship between time-to-event and time-varying covariates $x(t)$ in the presence of repeated time-labelled measurements.

is assumed for the survival part, where $m_i(t)$ denotes the mean of $x_i(t)$, i.e.

$$m_i(t) \coloneqq \boldsymbol{\beta}^{\mathrm{T}}\mathbf{u}_i(t) + \mathbf{b}_i^{\mathrm{T}}\mathbf{z}_i(t). \tag{4.2}$$

4.1.3 Contributions

The standard joint model considered above is useful in many settings but can be extended in various ways to more flexibly model a wider range of data. In this work, we present various ways in which insights from generative machine learning models and recent work in the survival analysis setting Danks and Yau (2022) can be used to provide more general models in the longitudinal survival setting. More specifically, we:

1. Present how adopting a hazard-based approach can be problematic in the longitudinal survival setting (Section 4.2).

2. Show how DeSurv can be embedded within the joint longitudinal-survival formalism to

allow for increased flexibility in the survival component (Section 4.3.1).

3. Demonstrate how Latent Neural ODEs can be used to flexibly model both the longitudinal and survival components of joint longitudinal-survival models and how they improve scaling with respect to data dimension and trajectory complexity (Section 4.3.2).

4. Show how survival time can be modelled directly in the longitudinal setting, allowing some of the complexities inherent to inference in random effects-based models to be circumvented (Section 4.3.3).

5. Demonstrate the properties of these models on synthetic and real-world examples (Section 4.4).

4.2 Hazard-Based Modelling in the Longitudinal Setting

The popularity of the Cox Proportional Hazards model in the canonical baseline, single-risk setting, leads many to consider it natural to develop models centred around the notion of the hazard function in the longitudinal setting. Indeed, as a natural extension of the Cox model to the longitudinal setting, a model of the form (4.1) is often employed when performing joint longitudinal-survival analysis. However, it should be noted that hazard-based models of this form can be problematic, particularly in the longitudinal setting, further motivating the consideration of models that utilise DeSurv's notion of directly targeting the survival function.

The problems associated with hazard-based models in the longitudinal setting are best seen by expressing the survival function via the cumulative hazard function, i.e.

$$S(t) = \exp\{-\Lambda(t)\},$$

with

$$\Lambda(t) = \int_0^t \lambda(s)\,\mathrm{d}s \tag{4.3}$$

and λ the hazard function. Note that for $S(t)$ to be a valid survival function, it must satisfy $\lim_{t\to\infty} S(t) = 0$, in turn implying that the associated cumulative hazard $\Lambda(t)$, i.e. the integral (4.3), must diverge for $t = \infty$. In the non-longitudinal, standard Cox setting, i.e.

$$\lambda(t; \mathbf{x}_0) = \lambda_0(t) \exp\left\{\beta^{\mathrm{T}} \mathbf{x}_0\right\}.$$

issues are not typically encountered as the baseline hazard is either not considered due to the use of partial log-likelihood and interest in hazard ratios or a valid baseline hazard λ_0 is chosen (e.g. Weibull) and given a suitable λ_0, the Cox hazard will be valid, as seen by e.g.

$$S(t; \mathbf{x}_0) = S_0(t)^{\exp\left\{\beta^{\mathrm{T}} \mathbf{x}_0\right\}}$$

in the Cox setting.

However, in the longitudinal setting, where the canonical joint model takes the form (4.1), i.e.

$$\lambda_i(t) = \lambda_0(t) \exp\left\{\boldsymbol{\gamma}^{\mathrm{T}} \mathbf{x}_0^{(i)} + \alpha^{\mathrm{T}} m_i(t)\right\},$$

the computation of the log-likelihood requires the baseline hazard $\lambda_0(t)$, hence its form must be specified. Furthermore, λ_0 being a valid hazard is no longer sufficient to guarantee that $\lambda_i(t)$ is valid. To illustrate, consider a Weibull hazard for $\lambda_0(t)$ with scale and shape parameters l and k respectively and a linear mixed effects model $m_i(t)$. In this setting, the hazard is

$$\lambda(t) = kl^{-k} t^{k-1} \exp\left\{\boldsymbol{\gamma}^{\mathrm{T}} \mathbf{x}_0 + \alpha_0 \left(\beta_0 + b_i^0\right) + \alpha_1 \left(\beta_1 + b_i^1\right) t\right\}.$$

which does not necessarily diverge upon integration as required. For example, if $\alpha_1 \left(\beta_1 + b_i^1\right) = -1$, then

$$\lambda(t) = Ct^{k-1} \exp\{-t\}$$

where C is a time-independent constant. This implies that

$$\lim_{t\to\infty} \Lambda(t) = C \lim_{t\to\infty} \int_0^t s^{k-1} \exp\{-s\}\, \mathrm{d}s\,,$$

which is by definition $C\Gamma(k)$, where Γ is the Gamma function, which is finite for finite $k > 0$. Issues such as this arise because typical hazard-based approaches do not typically constrain the behaviour of the hazard even though it must be divergent under integration to describe a valid survival function. Similar issues are seen in the context of competing-risks analysis when using the Fine-Gray subdistribution hazards model (Fine and Gray, 1999), where it is possible for the Cumulative Distribution Functions associated with event occurence to exceed 1 (Austin et al., 2021). These issues can be avoided by instead targeting the survival function in a constraint-aware way, as is done with the DeSurv-derived models presented in the next section.

4.3 Extending the Canonical Joint Longitudinal-Survival Model

4.3.1 Derivative-Based Neural Joint Longitudinal-Survival Modelling

In the context of joint longitudinal-survival modelling, the assumption of proportional hazards is, as in the case of baseline survival analysis, the dominant dogma. However, one need not make this assumption or necessarily use hazard-based models which, as discussed in the previous section, can be problematic in the longitudinal setting. One can instead often gain from direct modelling of the survival function via DeSurv (Danks and Yau, 2022).

The most immediate way to utilise DeSurv in the longitudinal setting is to recall that

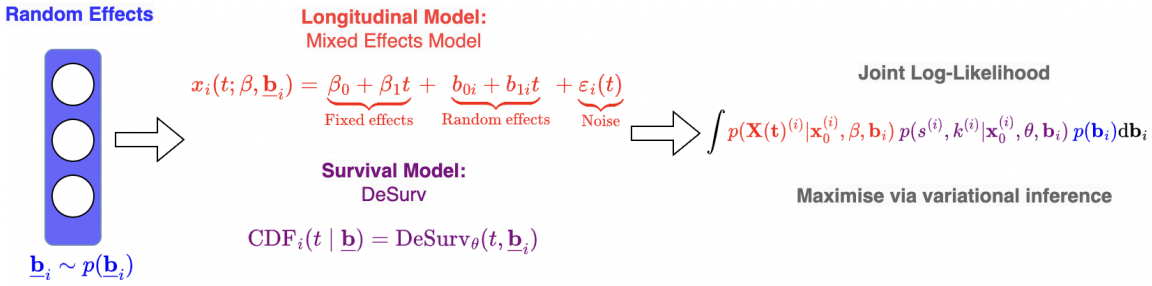

Figure 4.2: **Joint DeSurv Model Diagram.** A pictorial representation of the model we refer to as Joint DeSurv (Section 4.3.1).

the full log-likelihood of a joint longitudinal-survival model resembles

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) = \sum_{i=1}^{N} \log p(s^{(i)}, k^{(i)}, \mathbf{X}(\mathbf{t})^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}),$$

with

$$p(s^{(i)}, k^{(i)}, \mathbf{X}(\mathbf{t})^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}) = \int \left\{ p(\mathbf{X}(\mathbf{t})^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i) p(s^{(i)}, k^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i) p(\mathbf{b}_i) \right\} \mathrm{d}\mathbf{b}_i \,.$$

In the proportional hazards case, the survival portion of the model $p(s^{(i)}, k^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i)$ is derived from the hazard as in (4.1). Replacing this survival portion with the corresponding DeSurv expression, i.e.

$$\log p(s^{(i)}, k^{(i)}|\mathbf{x}_0^{(i)}, \boldsymbol{\theta}, \mathbf{b}_i) = \mathbb{1}\left(k^{(i)} = \varnothing\right) \log\left(\hat{S}_{\boldsymbol{\theta}}(s^{(i)}|\mathbf{x}_0^{(i)}, \mathbf{b}_i)\right) + \mathbb{1}\left(k^{(i)} \neq \varnothing\right) \log\left(\hat{f}_{\boldsymbol{\theta}}(s^{(i)}|\mathbf{x}_0^{(i)}, \mathbf{b}_i)\right),$$

where $\hat{S}_{\boldsymbol{\theta}}(s^{(i)}|\mathbf{x}_0^{(i)}, \mathbf{b}_i)$ and $\hat{f}_{\boldsymbol{\theta}}(s^{(i)}|\mathbf{x}_0^{(i)}, \mathbf{b}_i)$ are the survival function and density respectively associated with a DeSurv model with parameters (contained within) $\boldsymbol{\theta}$ and inputs $\left(\mathbf{x}_0^{(i)}, \mathbf{b}_i\right)$ provides the model class we refer to as Joint DeSurv. A pictorial representation of this model is shown in Figure 4.2.

Note that the longitudinal part of a joint model, i.e. (4.2), can also be non-linearised in t. For example, to move from a straight line to a cubic polynomial, one may replace the standard linear form of

$$m_i(t) = \left(\beta_0 + b_i^0\right) + \left(\beta_1 + b_i^1\right) t$$

with

$$m_i(t) = \left(\beta_0 + b_i^0\right) + \left(\beta_1 + b_i^1\right) t + \left(\beta_2 + b_i^2\right) t^2 + \left(\beta_3 + b_i^3\right) t^3,$$

and proceed as normal. In Section 4.4, we refer to joint DeSurv models as e.g. Joint DeSurv (Linear) and Joint DeSurv (Non-linear, Cubic Polynomial) to clarify the form of longitudinal model paired with the DeSurv component. Similarly, a standard Cox joint model is referred to as e.g. JM (Linear) and JM (Non-linear, Cubic Polynomial).

4.3.2 Flexible Latent Neural ODE-based Joint Longitudinal-Survival Modelling

The standard way to attempt more flexible longitudinal modelling within the random effects framework is, as was described in the previous section, to let $\mathbf{x}(t)$ be described by increasingly complex, higher order models. However, the number of random effects, $\dim(\mathbf{b}_i)$, increases with this complexity. For example, in the case of increasing polynomial order in the fixed and random effects, $\dim(\mathbf{b}_i)$ increases linearly with polynomial order. Note also that for $\mathbf{x}(t) \in \mathbb{R}^d$, $\dim(\mathbf{b}_i)$ also increases linearly with d. Such an approach therefore scales poorly when attempting to flexibly model high-dimensional data. Furthermore, polynomial bases of this form do not tend to fit general longitudinal datasets well in general. Rather than choose a high-order polynomial or spline, with a large associated $\dim(\mathbf{b}_i)$, it would be preferable to be able to use a model to define $\dim(\mathbf{b}_i)$ a priori and proceed to model the joint data given this constraint. Here, we present an approach which allows this by using latent neural ordinary differential equations (see Section 1.2.3), in turn introducing a model we refer to as NODESurv.

At the heart of this idea is the observation that the latent neural ODE generative

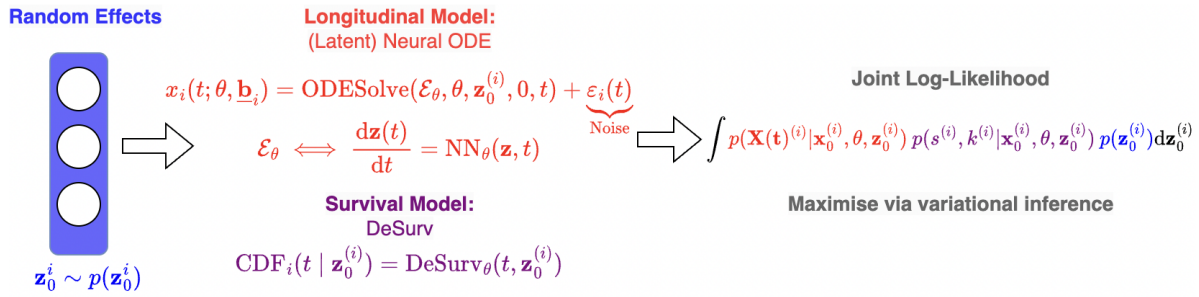

Figure 4.3: **NODESurv Model Diagram.** A pictorial representation of the model we refer to as NODESurv (Section 4.3.2).

model, i.e.

$$\begin{aligned}
\mathbf{z}_0 &\sim p(\mathbf{z}_0) \\
\mathbf{z}_n &= \text{ODESolve}(\mathcal{E}_{\boldsymbol{\theta}}, \mathbf{z}_0, t_0, t_n) \\
\mathbf{x}_n &\sim p_\lambda(\mathbf{x}_n|\mathbf{z_n}),
\end{aligned}$$

where $\mathcal{E}_{\boldsymbol{\theta}}$ represents the equation

$$\frac{d\mathbf{z}}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{z}, t),$$

with $\mathbf{f}_{\boldsymbol{\theta}}$ a neural network with parameters $\boldsymbol{\theta}$ and λ representing NN parameters mapping $\mathbf{z}$ to $\mathbf{x}$, can be thought of as generating longitudinal trajectories from a fixed-dimensional latent $\mathbf{z}_0$, much like a standard mixed effects model can be thought of as generating trajectories from the latent vector $\mathbf{b}$. The key difference in the context of latent neural ODEs however is that the complexity of the forward mapping and the dimension of the output $\mathbf{x}(t)$ can grow without necessarily increasing the dimension of $\mathbf{z}_0$. We extend the form of the prior to account for potential baseline information so that $p_\xi(\mathbf{z}_0|\mathbf{x}_0)$ rather than $p(\mathbf{z}_0)$. We then pair this longitudinal portion with a DeSurv survival model which takes as input $(\mathbf{z}_0, \mathbf{x}_0, t)$ and outputs an estimate of CDF $F(t|\mathbf{z_0}, \mathbf{x}_0)$. Let the parameters of this model be denoted by α. In this work we use the same feedforward NN-based approach as in Danks and Yau (2022), however one could model the derivative $(g(t))$ functions within DeSurv in a similar way to $\mathbf{x}(t)$ in the Latent Neural ODE if desired. A pictorial representation of the full NODESurv

model is shown in Figure 4.3.

Associated with the NODESurv model is a likelihood combining the longitudinal and survival aspects. Firstly, we have the likelihood associated with patient i's measurements $\mathcal{X}_i \coloneqq \mathbf{X}(\mathbf{t})^{(i)}$ being generated from the latent neural ODE. Secondly, we have the likelihood associated with a patient's survival data $\mathcal{Y}_i \coloneqq (s^{(i)}, k^{(i)})$ under DeSurv given $\mathbf{z}_0, \mathbf{x}_0$. Assuming that both the longitudinal and survival processes are defined completely by $\mathbf{z}_0$, i.e. they are conditionally independent given $\mathbf{z}_0$ (as assumed in a random effects-based joint model), we can derive an ELBO loss of the form $\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i$, with

$$\mathcal{L}_i(\tilde{\theta}, \phi) = \mathop{\mathbb{E}}_{\mathbf{z}_0^{(i)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}_0^{(i)})} \left[-\log p_{\tilde{\theta}}(\mathcal{X}_i, \mathcal{Y}_i | \mathbf{z}_0^{(i)}) \right] + \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z}_0^{(i)}) \| p(\mathbf{z}_0^{(i)}) \right],$$

where $\tilde{\theta} = (\alpha, \theta, \xi, \lambda)$ and

$$\log p_{\tilde{\theta}}(\mathcal{X}_i, \mathcal{Y}_i | \mathbf{z}_0^{(i)}) = \log p_{\tilde{\theta}}(\mathcal{X}_i | \mathbf{z}_0^{(i)}) + \log p_{\tilde{\theta}}(\mathcal{Y}_i | \mathbf{z}_0^{(i)}) \tag{4.4}$$

due to conditional independence.

For consistency with the inference process typically used in standard joint longitudinal-survival models (and, in turn, Joint DeSurv), we utilise non-amortised variational inference to optimise this objective, that is we parameterise

$$q_\phi(\mathcal{Z}_0) = q_\phi(\mathbf{z}_0^{(1)}, \mathbf{z}_0^{(2)}, \ldots, \mathbf{z}_0^{(N)}) = q_\phi(\mathbf{z}_0^{(1)}) q_\phi(\mathbf{z}_0^{(2)}) \ldots q_\phi(\mathbf{z}_0^{(N)})$$

via N pairs of Gaussian parameters $(\boldsymbol{\mu}, D)$ and iteratively update these. This removes the complexity of designing and tuning an encoder in return for additional computation at test time, namely performing posterior inference for new test points.

4.3.3 Discriminative Longitudinal Survival Analysis

In the previous section, we saw how the use of latent neural ODEs can allow for the use of flexible longitudinal and survival modelling without the need to use increasingly complex

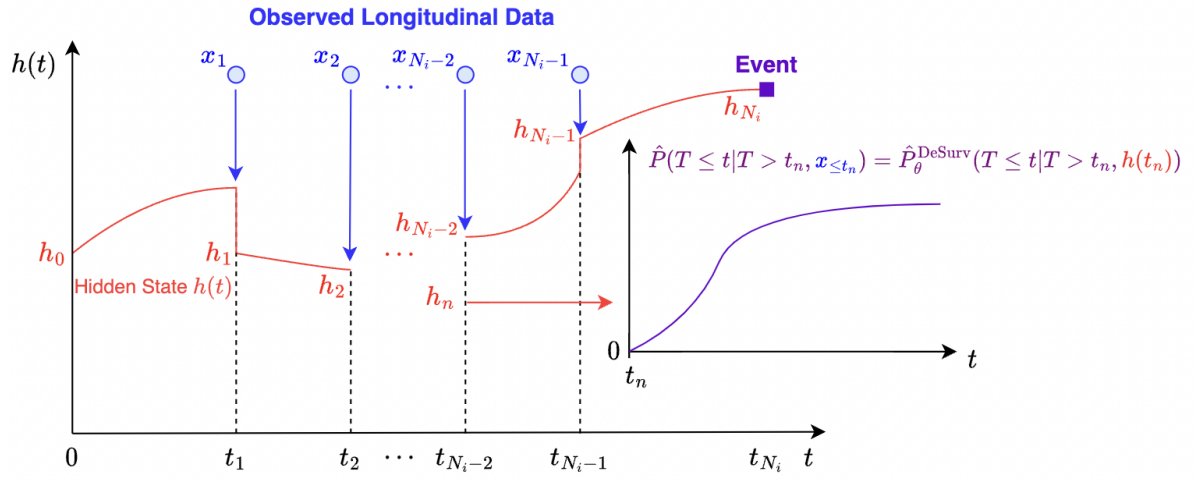

Figure 4.4: **D-NODESurv Model Diagram.** A pictorial representation of the model we refer to as D-NODESurv (Section 4.3.3).

basis functions with accompanying growth in $\dim(\mathbf{b}_i)$. However, the latent nature of $\mathbf{b}$, or $\mathbf{z}_0$ in the case of NODESurv, in random effects-style joint longitudinal-survival models introduces complexity into the inference process relative to a more discriminative approach of directly modelling the survival distribution given data. Additionally, modelling continuous and categorical covariates simultaneously in generative models of the style considered in random effect-style joint models can be challenging due to the different likelihood scales involved. Furthermore, in certain settings (e.g. high-dimensional, high-frequency sampling), it may be required to balance the survival and longitudinal aspects of the likelihood beyond the original quantity derived. Here we present RNN-based methodology, which we refer to as D-NODESurv (Discriminative-NODESurv), which performs discriminative longitudinal survival analysis, hence avoiding such difficulties.

At the heart of this methodology is the idea that an RNN's hidden state, $h(t)$, can be thought of as a representation of a patient's state at time t given their history and can therefore act as an appropriate, covariate-style input to a DeSurv model, much like the random effects in Joint DeSurv and initial condition $\mathbf{z}_0$ in NODESurv. Mathematically, this

translates to the argument that the CDF given observations up to t_n, i.e.

$$P(T \leq t | T > t_n, \mathcal{X}_{i,t \leq t_n}),$$

can be modelled as

$$\hat{P}(T \leq t | T > t_n, \mathcal{X}_{i,t \leq t_n}) = \hat{P}_{\boldsymbol{\theta}}^{\text{DeSurv}}(T \leq t | T > t_n, h(t_n)) \tag{4.5}$$

One is free to choose the form of RNN employed to generate the hidden state $h(t)$. One option is to specify an ODERNN (see Section 1.2.3) with hidden state $h(t)$ obeying updates of the form

$$h(t) = \text{ODESolve}\left(\mathbf{f}_{\boldsymbol{\theta}}, h_{i-1}, t_{i-1}, t\right)$$

between observations, i.e. for $t \in [t_{i-1}, t_i)$, and

$$h_i = \text{RNNCell}\left(h_i', \mathbf{x}_i\right),$$

with

$$h_i' = \text{ODESolve}\left(\mathbf{f}_{\boldsymbol{\theta}}, h_{i-1}, t_{i-1}, t_i\right)$$

for observed t_i. A pictorial representation of this architecture is shown in Figure 4.4. In the OPTIMAL data of relevance to this model, interval sampling is used, where patient measurements are listed as being uniform in time. There is therefore limited benefit to introducing ODERNN evolution between observations, and we rather use a standard LSTM arrangement, where RNNCell in the above is an LSTM cell and the hidden state is considered constant between observations.

Under this formalism (4.5), modelling a dataset $\mathcal{D}$ can be considered a set of $\sum_{i=1}^{N} N_i$ single-risk baseline survival subproblems, each of which has a DeSurv negative log-likelihood loss term associated with it. The overall training loss can be obtained by summing over these problems such that

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{j=1}^{N_i} \mathcal{L}_{ij}^{\text{DeSurv}}(t_{ij}, h_{ij}, k^{(i)})$$

and optimised using standard gradient descent techniques.

4.4 Experiments

We now investigate the behaviour of the aforementioned models in a variety of data settings. We begin by introducing the datasets under consideration in Section 4.4.1. We then present experimental findings associated with joint longitudinal-survival methods and discriminative longitudinal-survival methods in Sections 4.4.2 & 4.4.3 respectively.

4.4.1 Datasets

Due to the lack of reliable metrics to evaluate performance on general unseen test data (see e.g. Rindt et al. (2021)), synthetic data plays a key role when investigating behaviour of proposed survival models, particularly in the longitudinal setting, as it is possible to evaluate reliable performance metrics by comparing inferred survival functions with the corresponding ground truth. Furthermore, datasets can be generated which explicitly do or do not satisfy assumptions such as proportional hazards, allowing the performance of models with such assumptions to be evaluated on data which is formally consistent or inconsistent with them. Of course, it is also important to observe the utility of proposed models on real-world data. Whilst this is again challenging due to imperfect metrics and often high levels of censoring, insights into performance can still be gained via visualisation and informed metric quantification.

For our experiments, we utilise 4 synthetic datasets (Synthetic datasets A–D) and one real-world dataset (CPRD data). Synthetic datasets A–C are designed for and evaluated on the joint longitudinal survival models discussed in Sections 4.1.2, 4.3.1 & 4.3.2, i.e. JM, Joint DeSurv and NODESurv. In particular, they generate data with and without a Cox-based hazard function and with linear and non-linear longitudinal parts in order to probe how a model with Cox-based assumptions performs relative to the proposed Joint DeSurv and

NODESurv models in both Cox and non-Cox settings. Meanwhile, we use synthetic dataset D and the CPRD data to probe the behaviour of the discriminative model of Section 4.3.3, D-NODESurv. Full details of each of the datasets are provided under the relevant subheading below.

Synthetic Dataset A (Linear + Cox)

Synthetic dataset A generates data according to a Joint Cox model with a linear longitudinal part. This represents what is considered by many to be the simplest, most natural form of joint longitudinal survival model and is commonly used in the medical statistics literature. Explicitly, we have a generative model of the form

$$\begin{aligned} b_0^j,\ b_1^j &\sim \mathcal{N}(0, I) \quad \text{(Random Effects)} \\ \tilde{\beta}_0^j,\ \tilde{\beta}_1^j &= \beta_0^j + b_0^j,\ \beta_1^j + b_1^j \\ m_j(t) &= \tilde{\beta}_0^j + \tilde{\beta}_1^j t \\ \varepsilon_j(t) &\sim \mathcal{N}(0, \sigma_j^2) \\ x_j(t) &\sim m_j(t) + \varepsilon_j(t) \end{aligned}$$

for longitudinal covariates $x_j(t)$ where $j = 1, \ldots, D$ denotes covariate dimension. The survival component of the model is defined by the Cox hazard relation

$$\lambda(t) = \lambda_0(t) \exp\left\{ \sum_j \alpha_j m_j(t) \right\},$$

for some baseline hazard $\lambda_0(t)$. In this setting, we choose $\lambda_0(t)$ to be a Weibull hazard such that $\lambda_0(t) = kl^{-k}t^{k-1}$ for shape parameter k and scale parameter l. We set $k = 5, l = 1.5$, implying a mean and variance of approximately 1.4 and 0.1 respectively under the baseline hazard.

It should be noted that in its unconstrained form, this model is problematic, as discussed in Section 4.2, as it may be possible for $\lambda(t)$ to decay exponentially quickly to zero,

resulting in finite cumulative hazard for large t. In order to combat this and ensure a valid survival function for all samples, we set $\alpha_j > 0, \beta_j > 0$ and set $b_1^j = -b_1^j$ for subjects with $\sum_j \alpha_j b_1^j < 0$.

Time points $t_1, \ldots, t_{N_i}$ are generated by first sampling an event time T_i^{event} under $\lambda(t)$, then choosing $t_{N_i} = T_i^{\text{event}}$ with probability $1 - p_{\text{cens}}$ and $t_{N_i} = t_{\text{cens}} \sim \text{Unif}(0, T_i^{\text{event}})$ with probability p_{cens}. $t_1, \ldots, t_{N_i - 1}$ are then equally spaced across $[0, t_{N_i}]$ with $N_i = \min(n + \lfloor t_{N_i}/\tau \rfloor, n_{\max})$ for some timescale τ. In this setting, we set $p_{\text{cens}} = 0.2$, $n = 3$, $n_{\max} = 7$, $\tau = 0.5$, $D = 2$ and set the number of patients, N, to 500.

Synthetic Dataset B (Linear + Non-Cox)

For this dataset, we retain the notion of a linear longitudinal component, but do not generate the data using the canonical Cox proportional hazards assumption. Instead we assume that survival times are drawn from a Weibull distribution with mixed effect-dependent shape and scale parameters.

In particular, we describe the shape parameter k via a linear transformation of the overall mixed parameters $\tilde{\boldsymbol{\beta}} = \{\tilde{\beta}_0^j, \tilde{\beta}_1^j\}_{j=1}^D$ and set l to retain a fixed variance of 0.1 and manage the difficulty of the prediction problem. The hazard is therefore of the form

$$\lambda(t; \tilde{\boldsymbol{\beta}}) = k(\tilde{\boldsymbol{\beta}}) l(k(\tilde{\boldsymbol{\beta}}))^{-k(\tilde{\boldsymbol{\beta}})} t^{k(\tilde{\boldsymbol{\beta}})-1},$$

with

$$k(\tilde{\boldsymbol{\beta}}) = 2 + \boldsymbol{\theta}_k^{\mathrm{T}} \tilde{\boldsymbol{\beta}}$$
$$l(k(\tilde{\boldsymbol{\beta}})) = \sqrt{\frac{\text{variance}}{\Gamma\left(1 + 2/k\right) - \Gamma\left(1 + 1/k\right)^2}}.$$

We draw the elements of $\boldsymbol{\theta}_k$ from Unif$[0.1, 0.6]$, leading larger $\boldsymbol{\beta}$ to correspond to longer lifetimes. We keep the model for $x_j(t)$ unchanged relative to synthetic dataset A to isolate

the effect of the survival process change.

Synthetic Dataset C (Non-Linear + Non-Cox)

In this dataset, we again use the idea from synthetic dataset B that the survival process follows a Weibull distribution such that

$$\lambda(t; \tilde{\boldsymbol{\beta}}) = k(\tilde{\boldsymbol{\beta}}) l(k(\tilde{\boldsymbol{\beta}}))^{-k(\tilde{\boldsymbol{\beta}})} t^{k(\tilde{\boldsymbol{\beta}})-1},$$

with the same parameter settings as before. However, we now edit the form of the longitudinal process so that it is no longer a linear function, but is rather a cubic polynomial, i.e.

$$m_j(t) = \tilde{\beta}_0^j + \tilde{\beta}_1^j t + \tilde{\beta}_2^j t^2 + \tilde{\beta}_3^j t^3.$$

Note that this increases the number of random effects from $2D$ to $4D$ due to the addition of the quadratic and cubic components.

Synthetic Dataset D (Linear + Non-Cox)

This dataset is generated with the discriminative model setting in mind. It follows the same overall generative model format as synthetic dataset B, but edits the observation time sampling procedure to be such that patients are sampled at defined times, in our case every 0.1 time units. This mirrors the format of the CPRD data, where the data is provided in interval format, where measurements made during any 4-month interval are recorded at the end of said interval.

CPRD Data

The CPRD data under consideration was extracted from EHR records of patients from the Birmingham and Black Country regions using the Data Extractor for Epidemiological

Research (DExtER) tool (Gokhale et al., 2021). We consider data originating from $N = 132121$ individuals registered at GP practices in Birmingham & The Black Country with a type 2 diabetes diagnosis. Baseline and longitudinal measurements (from repeated visits to primary care) are recorded. Baseline characteristics used are Age, Sex, Smoking Status and BMI. Longitudinal variables used are Age, Body Mass Index (BMI), HbA1c, Systolic Blood Pressure (BP) and Serum Creatinine. Based on clinical motivation, the outcome of interest is a "cardiovascular event", which for the purposes of this study is considered to be any occurrence of stroke, transient ischemic attack, ischemic heart disease, myocardial infarction or heart failure. Baseline missing data is imputed via mean and mode imputation. Missing longitudinal entries are imputed via within-subject piecewise-constant interpolation.

4.4.2 Joint Longitudinal-Survival Model Experiments

We begin the experiments by probing the behaviour of the joint longitudinal-survival models considered in previous subsections, i.e. a standard joint model (JM), Joint DeSurv and NODESurv. We first train each of the models on each of the synthetic datasets A, B and C, which represent Cox + Linear, Non-Cox + Linear and Non-Cox + Non-Linear respectively. We then evaluate each of the models on 5 test data sets, each with $N = 100$, drawn from the same generative process and report in Table 4.1 the mean and standard deviation of 3 metrics calculated for each dataset, namely Longitudinal RMSE, Survival Log-Likelihood, and Survival MISE. In this context, Longitudinal RMSE refers to the Root Mean Square Error between observed longitudinal observations $\{\mathbf{X}(\mathbf{t})^{(i)}\}_{i=1}^{N}$ and the predictions for such points under the longitudinal portion of the model, with the mean divisor taken to be the total number of observations, i.e. $\sum_{i=1}^{N} N_i$. Survival Log-Likelihood refers to the survival portion of the log-likelihood within the ELBO, i.e. the second term in (4.4). Survival MISE refers to the Mean Integrated Squared Error between the ground truth CDF and that inferred

Table 4.1: **Joint Model Synthetic Data Performance.** Metric values representing the performance of each of the joint longitudinal-survival models considered in Section 4.3 on synthetic datasets A, B & C as defined in Section 4.4.1. The metrics considered are defined in Section 4.4.2.

Dataset	Method	Longit. RMSE	Surv. Log Likelihood	Surv. MISE	$\dim(\mathbf{b}_i)$
A	JM	0.1485 ± 0.0027	$\mathbf{100.62 \pm 7.74}$	$\mathbf{0.0329 \pm 0.0056}$	4
	Joint DeSurv	0.1496 ± 0.0017	90.49 ± 6.43	0.1037 ± 0.0081	4
	NODESurv	$\mathbf{0.1466 \pm 0.0018}$	90.37 ± 5.13	0.0879 ± 0.0092	4
B	JM	0.1444 ± 0.0025	$\mathbf{-36.88 \pm 6.11}$	0.5973 ± 0.2688	4
	Joint DeSurv	0.1453 ± 0.0015	-37.22 ± 6.84	$\mathbf{0.5202 \pm 0.2276}$	4
	NODESurv	$\mathbf{0.1442 \pm 0.0019}$	-36.91 ± 6.30	0.5538 ± 0.2053	4
C	JM	$\mathbf{0.2409 \pm 0.0063}$	-39.44 ± 4.62	0.8081 ± 0.2181	8
	Joint DeSurv	0.2873 ± 0.0065	$\mathbf{-28.02 \pm 4.21}$	$\mathbf{0.4446 \pm 0.0962}$	8
	NODESurv	0.3905 ± 0.0143	-33.57 ± 1.33	0.5272 ± 0.0990	4

by the relevant model. This can therefore be seen as a gold-standard metric of a survival model's fit to the ground-truth survival function. Table 4.1 also reports $\dim(\mathbf{b}_i)$, that is the dimension of the random effects, for each of the models ($\dim(\mathbf{z}_0)$ is entered in the case of NODESurv).

Inspecting Table 4.1, we see that on dataset A, that is Linear + Cox, all three models fit the data very well, with small Longitudinal RMSE and Survival MISEs present for all three models, however the standard joint Cox model (JM) performs best. This is unsurprising due to the generative process completely matching the JM model. On dataset B, that is

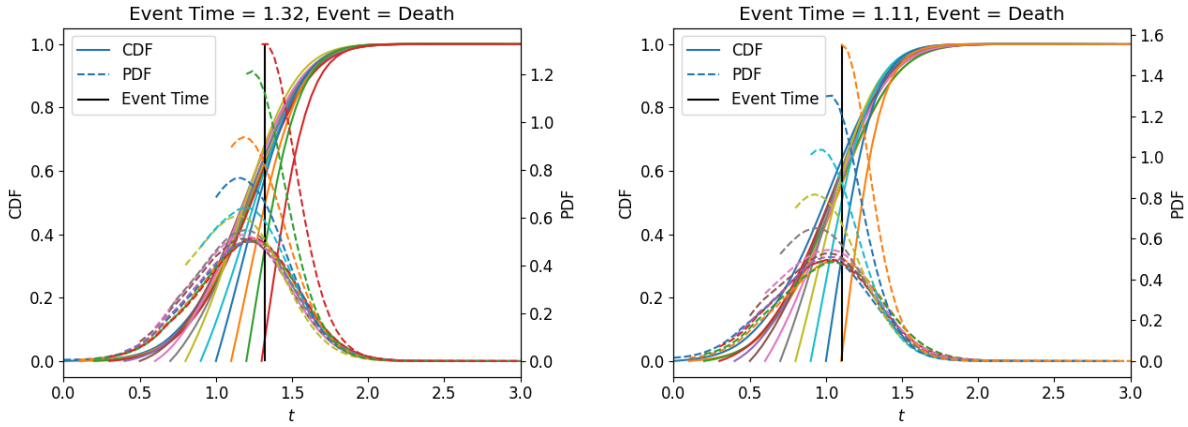

Figure 4.5: **D-NODESurv Predictions**. D-NODESurv predictions for two test datapoints from Synthetic Dataset D. D-NODESurv provides an estimate of the conditional CDF $P(T \leq t|T > t_n, \mathcal{X}_{i,t\leq t_n})$ and associated PDF for each observation time t_n.

Linear + Non-Cox, we see that JM now has the highest value of Survival MISE of the three models, suggesting it struggles to capture the non-Cox survival process more than Joint DeSurv and NODESurv. Each model demonstrates essentially equivalent performance on the longitudinal data, with all fitting the linear behaviour well. Finally, on dataset C, we see that again each model fits the longitudinal data well, with small Longitudinal RMSEs given the presence of some large values of $x_j(t)$, however Joint DeSurv and NODESurv are seen to obtain significantly lower Survival MISEs than JM. Note also that this performance is obtained even with our choice to set $\dim(\mathbf{z}_0) = 4$ for NODESurv in the cubic case.

4.4.3 Discriminative Model Experiments

We now consider the behaviour of the discriminative model, D-NODESurv, introduced in Section 4.3.3, using synthetic dataset D and the CPRD data described in Section 4.4.1.

Due to the bespoke nature of the D-NODESurv model, it is natural to wonder about

the form of its predictions. To clarify this, we begin by training D-NODESurv on synthetic dataset D and visualise the output of the model on two random test patients (Figure 4.5). In this figure it can be seen that at each observation time t_n, the model provides its estimate of the conditional CDF $P(T \leq t|T > t_n, \mathcal{X}_{i,t\leq t_n})$ (plotted as a solid line in Figure 4.5). The PDF associated with each of these CDFs can also be calculated and is represented by a dotted line of the same color. In a successful model, the PDFs should typically be concentrated around the observed event time, as seen in these cases.

We now move on to consider the quantitative performance of D-NODESurv on synthetic dataset D and the real-world CPRD OPTIMAL data. Of particular interest in this setting is the question of whether introducing a time-dependent hidden state $h(t)$, in this case within an RNN, frustrates or improves performance relative to that offered by what we refer to as "pooled" models, which treat the dataset as a collection of $\sum_i N_i$ independent survival problems. To address this question, we train pooled Cox and DeSurv models alongside the D-NODESurv model and compute the time-dependent concordance ($\mathbf{C}_{\text{td}}$), Integrated Brier Score (IBS) and Integrated Negative Binomial Log-Likelihood values (see Section 1.3.5 for definitions) on a held-out 10% test portion of the data. The values obtained are presented in Table 4.2, with uncertainties obtained by running 5 full train-evaluate runs, each with a different training/validation/test dataset but retaining a 80%/10%/10% split.

Upon inspection of Table 4.2, it is immediately clear that the synthetic data can be modelled almost equally well by all three methods, with all three approaches demonstrating strong performance. At first glance, it may seem surprising that D-NODESurv does not convincingly outperform the pooled approaches here, especially as the sampling frequency of synthetic dataset D is high (0.1 intervals, averaging > 10 observations per patient). One explanation is that as synthetic dataset D has a low-noise linear longitudinal process, when a pooled model is provided with t and $x(t)$ at time t, it is implicitly fed with information on the $\hat{\beta}$ values associated with its trajectory and underpinning the survival process.

Table 4.2: **Discriminative Model Performance on Synthetic and Real-World Data.**

Dataset	Method	$\mathbf{C}_{\text{td}}$	IBS	INBLL
Synthetic D	Pooled Cox	0.769 ± 0.012	0.0887 ± 0.0032	0.277 ± 0.015
	Pooled DeSurv	0.777 ± 0.021	$\mathbf{0.0839 \pm 0.0054}$	$\mathbf{0.266 \pm 0.017}$
	D-NODESurv	$\mathbf{0.779 \pm 0.015}$	0.0869 ± 0.0076	0.278 ± 0.021
CPRD	Pooled Cox	0.513 ± 0.15	0.172 ± 0.011	0.495 ± 0.023
	Pooled DeSurv	0.563 ± 0.005	$\mathbf{0.154 \pm 0.001}$	$\mathbf{0.474 \pm 0.001}$
	D-NODESurv	$\mathbf{0.568 \pm 0.005}$	0.156 ± 0.001	0.478 ± 0.001

The CPRD data represents a greater challenge for all three of the models, with Pooled Cox in particular struggling to capture any information, with a concordance value ≈ 0.5, i.e. that of random noise. As in the previous synthetic setting, there appears little difference between Pooled DeSurv and D-NODESurv performance, with both performing well relative to Cox. The fact that the use of a hidden state does not appear to improve modelling in this setting could be interpreted negatively, in the sense that it questions the need for a complete model such as D-NODESurv when a pooled approach performs equivalently. However, the results could also be interpreted the other way — the use of a hidden state does not appear to frustrate modelling relative to a standard pooled approach and is capable of capturing dependencies beyond those of a pooled approach if the data supports them. It should be noted that the form of the CPRD data is such that it is arguably expected that no particular benefit arises from the use of a hidden state due to high levels of longitudinal missingness and the relatively non-informative nature and sampling rate of the biomarkers measured (blood pressure, HbA1c, BMI, creatinine at minimum of 4-month intervals). A

valuable continuation of this work would be to apply the D-NODESurv and pooled models to data with high-frequency sampling of informative biomarkers such as that recorded in ICU units.

4.5 Discussion

The problem of performing time-to-event modelling in the presence of longitudinal data, referred to as longitudinal survival analysis, is becoming increasingly important as healthcare becomes increasingly data driven. Despite this, the area is not well-explored from a machine learning-aware perspective. This chapter presented the problem such a perspective, highlighting the similarity of the joint longitudinal-survival model with machine learning-style generative latent variable models and showing how the direct modelling of the survival function demonstrated in the baseline setting in Chapter 3 can be extended to the longitudinal setting. There are numerous natural directions in which to continue the work of this chapter, many of which are being carried out as part of the OPTIMAL project. For example, additional empirical evaluation of the introduced models could be performed to convey further the attributes of each in different settings. Linked to this is the explicit consideration of general-purpose evaluation metrics for survival models in general beyond those (non-proper) metrics (Rindt et al., 2021) currently employed when non likelihood-based models are utilised. Additionally, the extension of the approaches introduced in this chapter to the competing risk setting is a natural consideration.

Chapter Five

Conclusion

5.1 Summary

This thesis has presented a series of machine learning methods capable of modelling a wide range of tasks. Whilst the context of application and many of the notions used are varied, one theme continued throughout, namely that constraints, particularly those involving derivatives and beyond what are normally considered, can be valuable introductions to a machine learning model. We also saw that the theme of latent variables and their associated inference techniques appear in seemingly disjoint settings, for example both in the context of VAE-based modelling of scRNA-seq and in the joint longitudinal-survival analysis setting.

In Chapter 1, we introduced the background to our work via three at first glance seemingly unrelated areas, namely latent variable models, ordinary differential equations, and survival analysis, though it is seen in Chapter 4 that concepts from all three of these can be unified in a single model to perform longitudinal survival analysis.

We then, in Chapter 2, presented a novel method for simultaneous dimensionality reduction and feature clustering (BasisDeVAE) which allowed prior knowledge to be incor-

porated. In the process we also introduced a generalised derivative-based functional form for Gaussian-like transients which is useful in its own right for those modelling transient dynamics.

In Chapter 3, we introduced DeSurv as a new approach to survival analysis which is capable of modelling single-risk and competing-risk settings without the traditional assumptions and/or restrictions of traditional survival analysis approaches. We note that this is commensurate with other recent work in the literature focusing on likelihood-based survival modelling, e.g. Rindt et al. (2021).

Chapter 4 then considered the problem of longitudinal survival modelling from a modern perspective in light of DeSurv's development. We showed that the similarity between a traditional joint longitudinal-survival model and ML-based latent variable models is significant and can be utilised to build models for longitudinal survival analysis via modern methods including neural ODEs and DeSurv.

5.2 Future Directions

The methodological nature of the work presented in this thesis allows for numerous future directions of research, both in the form of further methodological development and in the form of additional data-driven testing and application of the models.

The work presented in Chapter 2 (BasisDeVAE) was primarily developed with the degenerative disease and scRNA-seq setting in mind. It would be valuable future work for those interested in other potential application settings to apply the work in that context. Another natural development of this work would be to consider settings within which one had access to multimodal 'omics data, referred to as the multi-omics setting, or where multiple

similar datasets exist and knowledge should be shared between them, often referred to as transfer learning. This may allow the inference of cross-modal latent variables and those that may be shared across biological processes respectively. Meanwhile, more incremental though immediate methodological extensions could consider the introduction of higher-order derivatives or autonomous ODEs into the decoder generative process.

Chapter 3 naturally lends itself to application to additional datasets for further evaluation and calibration checks, as well as the examination of its performance with respect to clinical expectations. Work is ongoing in this direction within the Yau Group, in particular by Christopher Yau and Natalia Hong (PhD Student, University of Oxford), to evaluate the model on data from the MuM-PreDiCT study. Results emerging from this work are encouraging and show that DeSurv can extract clinically consistent conclusions from the data.

The development of the methodology presented in Chapter 4 was inspired by the problem setting of the *OPTIMising therapies, discovering therapeutic targets and AI assisted clinical management for patients Living with complex multimorbidity* study (OPTIMAL). This is an ongoing large multi-institutional NIHR-funded project to investigate outcomes for patients with multimorbidity. At the heart of this study is the problem of predicting time-to-event given longitudinal data, hence the developments of Chapter 4. The methodology presented in Chapter 4 is currently adopted within the project and will be applied to a wider set of data and quite probably within the MuM-PreDiCT study as well. Future work may also consider methodological extensions to the methods presented, with the extension to the competing risks setting particularly natural and useful.

More generally, it should be noted that the evaluation of survival models is in general challenging and suboptimal, for example Rindt et al. (2021) highlight how standard, widely adopted metrics are not proper. Furthermore, the evaluation of more robust metrics, in

particular test log-likelihood is not possible in many models or is frustrated by different log-likelihoods not being directly comparable. This leads to studies either reporting with suboptimal metrics or running synthetic experiments in order to be able to evaluate a suitable ground-truth comparison for robust reporting. An important open problem therefore remains in the form of formulating a metric capable of quantifying survival model performance in a proper, reliable way without access to ground-truth and in the presence of censoring, particularly when competing risks are also present.

Appendix One

Appendix to Chapter 2

A.1 OASIS features

In the main text, we presented the results of experiments using data $\mathbf{X} \in \mathbb{R}^{2047\times 13}$ from the OASIS-3 dataset (LaMontagne et al., 2019). Here we describe the meaning of each feature and how it is obtained from the raw OASIS-3 data.

We began by extracting MRI sessions with associated FreeSurfer segmentations and downloading these segmentations. By default, the FreeSurfer software provides highly granular cortical and subcortical segmentation. We apply the groupings described in the appendix of Klein and Tourville (2012) and in the FreeSurfer wiki to obtain the volumes of the following 13 interpretable regions associated with cognitive decline (also used in e.g. Young et al. (2018)): frontal lobe, temporal lobe, parietal lobe, occipital lobe, cingulate, insula, accumbens, amygdala, caudate, hippocampus, pallidum, putamen, thalamus. We then divide each regional measurement by the corresponding patient's total intracranial volume to obtain normalised volumes (Whitwell et al., 2001). Finally, we standardise each feature of the data by subtracting its mean and dividing by its standard deviation.

Appendix Two

Appendix to Chapter 3

B.1 Experimental Details

B.1.1 Dataset Preprocessing

As noted in the main paper, no additional preprocessing was performed on the METABRIC and SUPPORT datasets beyond that described in Katzman et al. (2018).

SEER

We extract data from the SEER database corresponding to “Female malignant breast cancers". From this data, we extract patients with time-to-event measurements who either i) are alive as of the last measurement, ii) experienced death attributable to their cancer diagnosis, or iii) died due to “Diseases of heart", which we refer to as cardiovascular disease (CVD). We utilise all present informative variables, providing 23 (as noted in other works using (previous verions of) SEER, such as DeepHit (Lee et al., 2018)) variables including age, race, grade, laterality, histology codes, treatment information and tumour size (see e.g. this SEER dictionary for

example SEER features). Missing values were mode imputed.

B.1.2 Model Hyperparameters

All neural networks had two 32-unit hidden layers with ReLU activations. The Adam optimiser (Kingma and Ba, 2014) was used with $\beta_1 = 0.9, \beta_2 = 0.999$, $\alpha \in \{10^{-2}, 10^{-3}\}$ and batch size was 32 in all cases. As a discrete-time method, DeepHit requires a time discretisation scheme. In each case, we discretised the time frame into 300 equal intervals, leading to the output layer of DeepHit having $300K$ output nodes for K competing risks. In the loss of DeepHit, we set a relative weight of 4:1 between the likelihood and ranking components respectively and set $\sigma = 0.1$ following Lee et al. (2018).

B.2 Additional Synthetic Data Experiment

In the main text, DeepHit demonstrated limited discriminative performance on synthetic data. As the synthetic data contains many subjects with small hitting times, one may expect that performance should improve as the number of discretisation points is increased. To investigate this, we repeat the experiment associated with Table 3.4 with $n = 1000$ and $n = 10000$ discretisation points (Table B.1). Here it can be seen that increasing the number of discretisation points improves performance. However, performance appears to saturate for large n, and with 10000 discretisation points DeepHit still exhibits a lower concordance than DeSurv (for comparison we reevaluate DeSurv here using 10000 evaluation points when computing the CIFs for concordance). Note that whilst DeSurv's neural network output dimensions are always K ($= 2$ here), DeepHit's are nK, hence for $n = 10000$ in Table B.1, 20000 output dimensions are required.

Table B.1: **Effect of Increased Number of Discretisation Points on Synthetic Data DeepHit Performance.**

Method	$\mathbf{C_{td}}^{(1)}$	$\mathbf{C_{td}}^{(2)}$
DeepHit (n=300)	0.611 ± 0.009	0.570 ± 0.004
DeepHit (n=1000)	0.627 ± 0.007	0.583 ± 0.004
DeepHit (n=10000)	0.626 ± 0.009	0.584 ± 0.006
DeSurv (10000)	$\mathbf{0.637 \pm 0.010}$	$\mathbf{0.592 \pm 0.006}$

B.3 Example Survival Functions and CDFs

To gain insight into model behaviour, we visualise inferred survival functions and CIFs on each dataset.

Figures B.1 and B.2 show the survival functions of five random test patients in the METABRIC and SUPPORT datasets respectively according to the trained Cox, Cox-Time, DeepHit and DeSurv models. Note in each case that the models generally agree on the relative order of patient risk, however the trajectories are less separated by DeepHit than other models. Note also the similarity between DeSurv's trajectories and those found via Cox-Time, arguably the prevailing modern continuous time single-risk survival analysis approach at the time of writing.

Figures B.3 and B.4 show the cumulative incidence functions (CIFs) for five random test patients in the synthetic and SEER datasets respectively. As in the single-risk case, the models generally agree on the risk orderings of the patients, but DeSurv demonstrates increased patient-dependent trajectory separation.

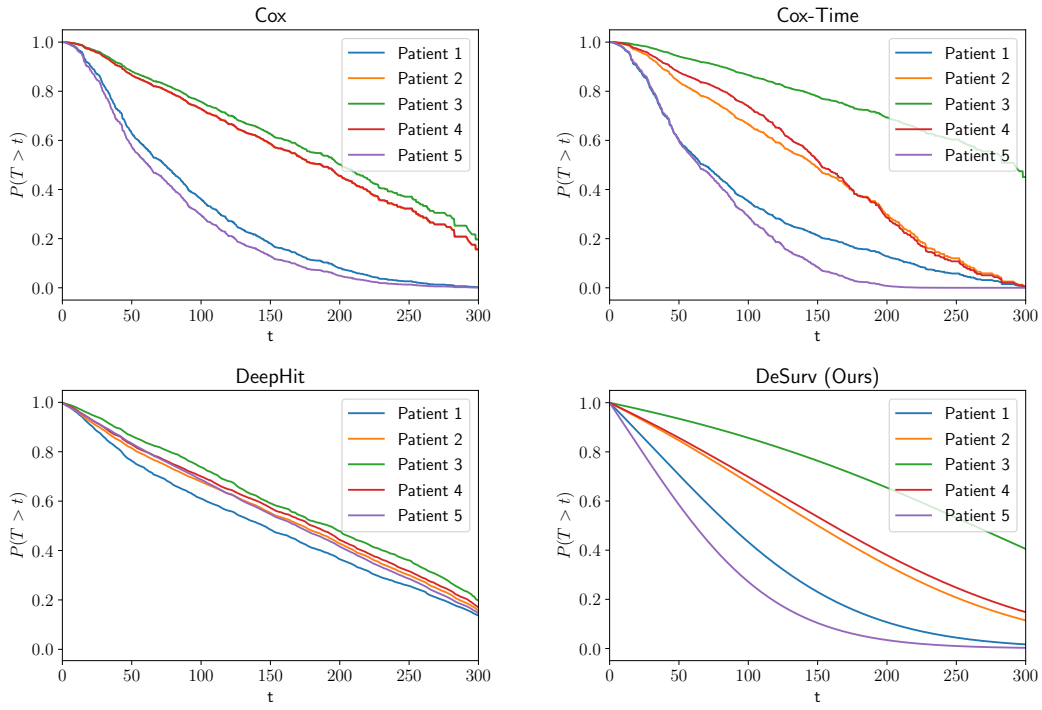

Figure B.1: **METABRIC Survival Functions.** 25-year survival trajectories for five random test patients according to four of the considered models. (Appendix B.3)

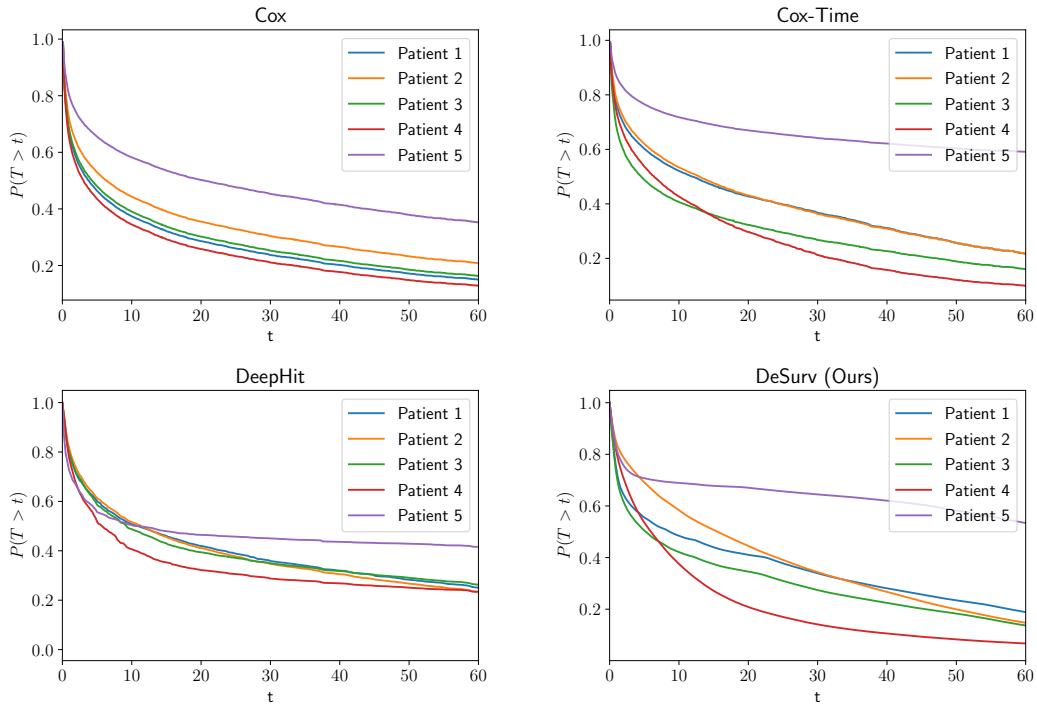

Figure B.2: **SUPPORT Survival Functions.** Five-year survival trajectories for five random test patients according to four of the considered models. (Appendix B.3)

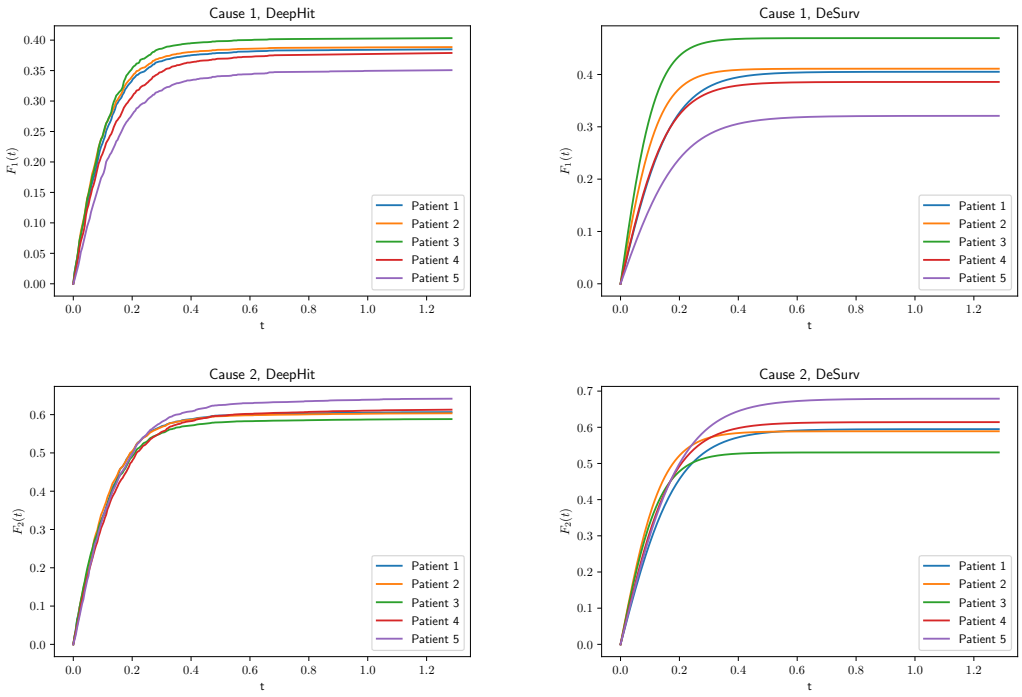

Figure B.3: **Synthetic Cumulative Incidence Functions.** Cumulative incidence functions for five random test subjects according to DeepHit and DeSurv. (Appendix B.3)

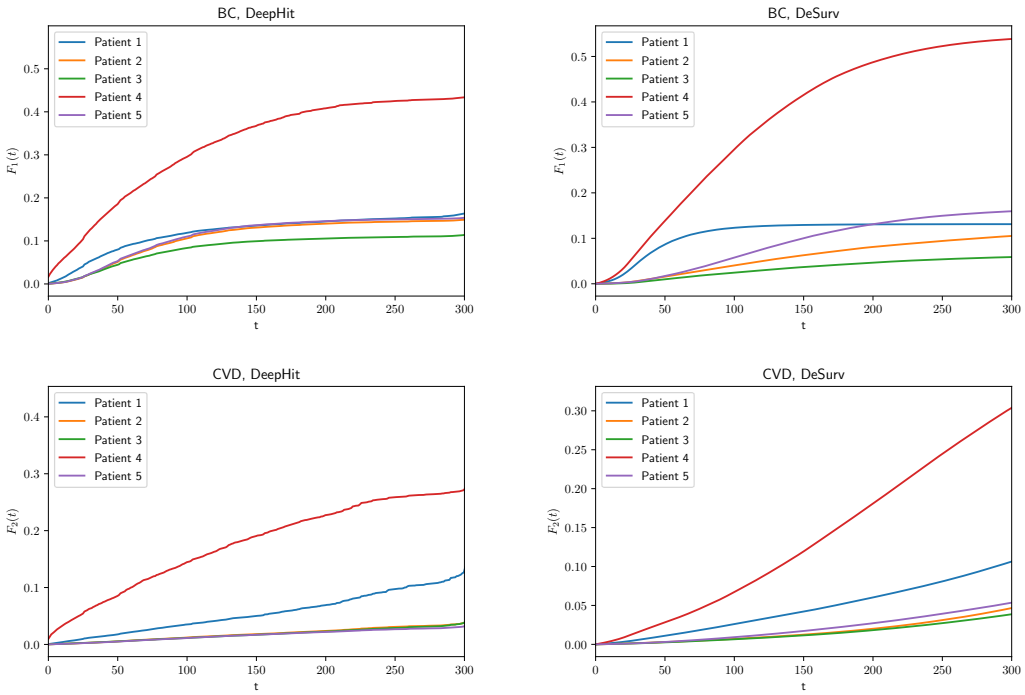

Figure B.4: **SEER Cumulative Incidence Functions.** 25-year cumulative incidence functions for five random test patients according to DeepHit and DeSurv. (Appendix B.3)

Bibliography

Ahmed M. Alaa and M. van der Schaar. 2017. Deep multi-task gaussian processes for survival analysis with competing risks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Laura Antolini, Patrizia Boracchi, and Elia Biganzoli. 2005. A time-dependent discrimination index for survival data. *Statistics in Medicine*, 24(24):3927–3944.

Peter C. Austin, Douglas S. Lee, and Jason P. Fine. 2016. Introduction to the analysis of survival data in the presence of competing risks. *Circulation*, 133(6):601–609.

Peter C. Austin, Ewout W. Steyerberg, and Hein Putter. 2021. Fine-gray subdistribution hazard models to simultaneously estimate the absolute risk of different event types: Cumulative total failure probability may exceed 1. *Statistics in Medicine*.

B Baesens, T Van Gestel, M Stepanova, D Van den Poel, and J Vanthienen. 2005. Neural network survival analysis for personal loan data. *Journal of the Operational Research Society*, 56(9):1089–1098.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98.

Elia Biganzoli, Patrizia Boracchi, Luigi Mariani, and Ettore Marubini. 1998. Feed forward

neural networks for the analysis of censored survival data: a partial logistic regression approach. *Statistics in Medicine*, 17(10):1169–1186.

Gabriel Blumenstock, Stefan Lessmann, and Hsin-Vonn Seow. 2020. Deep learning for survival and competing risk modelling. *Journal of the Operational Research Society*, 0(0):1–13.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016a. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany. Association for Computational Linguistics.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. 2016b. Generating sentences from a continuous space.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018a. JAX: composable transformations of Python+NumPy programs.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018b. JAX: composable transformations of Python+NumPy programs.

Leo Breiman. 2001. Random forests. *Machine Learning*, 45(1):5–32.

Charles C. Brown. 1975. On the use of indicator variables for studying the time-dependence of parameters in a response-time model. *Biometrics*, 31(4):863–872.

John Charles Butcher. 2016. *Numerical methods for ordinary differential equations*. John Wiley & Sons.

Kieran R Campbell and Christopher Yau. 2018. A descriptive marker gene approach to single-cell pseudotime inference. *Bioinformatics*, 35(1):28–35.

Ricky T. Q. Chen. 2018. torchdiffeq.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018a. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, pages 6571–6583. Curran Associates, Inc.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018b. Neural ordinary differential equations. *CoRR*, abs/1806.07366.

Michael R. Chernick, Wenceslao González-Manteiga, Rosa M. Crujeiras, and Erniel B. Barrios. 2011. Bootstrap methods. In Miodrag Lovric, editor, *International Encyclopedia of Statistical Science*, pages 169–174. Springer Berlin Heidelberg, Berlin, Heidelberg.

D. R. Cox. 1972. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220.

Andreas Damianou and Neil D. Lawrence. 2013. Deep Gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 207–215, Scottsdale, Arizona, USA. PMLR.

Dominic Danks and Christopher Yau. 2021. Basisdevae: Interpretable simultaneous dimensionality reduction and feature-level clustering with derivative-based variational autoencoders. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2410–2420. PMLR.

Dominic Danks and Christopher Yau. 2022. Derivative-based neural modelling of cumulative distribution functions for survival analysis. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 7240–7256. PMLR.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

Robert P. Dickinson and Robert J. Gelinas. 1976. Sensitivity analysis of ordinary differential equation systems—a direct method. *Journal of Computational Physics*, 21(2):123 – 143.

Stefan Dietrich, Anna Floegel, Martina Troll, Tilman Kühn, Wolfgang Rathmann, Anette Peters, Disorn Sookthai, Martin von Bergen, Rudolf Kaaks, Jerzy Adamski, Cornelia Prehn, Heiner Boeing, Matthias B Schulze, Thomas Illig, Tobias Pischon, Sven Knüppel, Rui Wang-Sattler, and Dagmar Drogan. 2016. Random survival forest in practice: a method for modelling complex metabolomics data in time to event analysis. *International Journal of Epidemiology*, 45(5):1406–1420.

Lore Dirick, Gerda Claeskens, and Bart Baesens. 2017. Time to default in credit scoring using survival analysis: a benchmark study. *Journal of the Operational Research Society*, 68(6):652–665.

Christina Ernst, Nils Eling, Celia P. Martinez-Jimenez, John C. Marioni, and Duncan T. Odom. 2019. Staged developmental mapping and x chromosome transcriptional dynamics during mouse spermatogenesis. *Nature Communications*, 10(1):1251.

David Faraggi and Richard Simon. 1995. A neural network model for survival data. *Statistics in Medicine*, 14(1):73–82.

Jason P. Fine and Robert J. Gray. 1999. A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association*, 94(446):496–509.

Marco Fornili, Federico Ambrogi, Patrizia Boracchi, and Elia Biganzoli. 2014. Piecewise exponential artificial neural networks (peann) for modeling hazard function with right

censored data. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*, pages 125–136, Cham. Springer International Publishing.

Stephane Fotso. 2018. Deep neural networks for survival analysis based on a multi-task framework.

Nick C. Fox and Jonathan M. Schott. 2004. Imaging cerebral atrophy: normal ageing to alzheimer's disease. *The Lancet*, 363(9406):392–394.

Michael Friedman. 1982. Piecewise Exponential Models for Survival Data with Covariates. *The Annals of Statistics*, 10(1):101 – 113.

Michael F. Gensheimer and Balasubramanian Narasimhan. 2019. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257.

Amir Gholami, Kurt Keutzer, and George Biros. 2019. Anode: Unconditionally accurate memory-efficient gradients for neural odes.

Krishna Margadhamane Gokhale, Joht Singh Chandan, Konstantinos Toulis, Georgios Gkoutos, Peter Tino, and Krishnarajah Nirantharakumar. 2021. Data extraction for epidemiological research (DExtER): a novel tool for automated clinical epidemiology studies. *Eur. J. Epidemiol.*, 36(2):165–178.

Erika Graf, Claudia Schmoor, Willi Sauerbrei, and Martin Schumacher. 1999. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545.

Jerry Green and John B. Shoven. 1986. The effects of interest rates on mortgage prepayments. *Journal of Money, Credit and Banking*, 18(1):41–59.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. 2015. Draw: A recurrent neural network for image generation. In *Proceedings of the 32nd International*

Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 1462–1471, Lille, France. PMLR.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

J. Hensman, M. Rattray, and N. D. Lawrence. 2015. Fast nonparametric clustering of structured time-series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):383–393.

James Hensman, Magnus Rattray, and Neil Lawrence. 2012. Fast variational inference in the conjugate exponential family. In *Advances in Neural Information Processing Systems*, volume 25, pages 2888–2896. Curran Associates, Inc.

I. Higgins, Loïc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. 2017. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*.

Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. 2005. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6e.

Zhiyuan Hu, Mara Artibani, Abdulkhaliq Alsaadi, Nina Wietek, Matteo Morotti, Tingyan Shi, Zhe Zhong, Laura Santana Gonzalez, Salma El-Sahhar, Mohammad KaramiNejadRanjbar, Garry Mallett, Yun Feng, Kenta Masuda, Yiyan Zheng, Kay Chong, Stephen Damato, Sunanda Dhar, Leticia Campo, Riccardo Garruto Campanile, Hooman Soleymani majd, Vikram Rai, David Maldonado-Perez, Stephanie Jones, Vincenzo Cerundolo, Tatjana

Sauka-Spengler, Christopher Yau, and Ahmed Ashour Ahmed. 2020. The repertoire of serous ovarian cancer non-genetic heterogeneity revealed by single-cell sequencing of normal fallopian tube epithelial cells. *Cancer Cell*, 37(2):226–242.e7.

Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. 2008. Random survival forests. *The Annals of Applied Statistics*, 2(3):841 – 860.

Jared L. Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. 2018. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1):24.

Ieva Kazlauskaite, Carl Henrik Ek, and Neill Campbell. 2019. Gaussian process latent variable alignment learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 748–757. PMLR.

P. Kidger. 2021a. torchcde.

P. Kidger. 2021b. torchsde.

P. Kidger. 2022. diffrax.

Patrick Kidger, James Foster, Xuechen Li, Harald Oberhauser, and Terry Lyons. 2021. Neural sdes as infinite-dimensional gans.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. 2020a. Neural controlled differential equations for irregular time series.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. 2020b. Neural controlled differential equations for irregular time series.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, volume 27, pages 3581–3589. Curran Associates, Inc.

Arno Klein and Jason Tourville. 2012. 101 labeled brain images and a consistent human cortical labeling protocol. *Frontiers in Neuroscience*, 6:171.

William A. Knaus, Frank E. Harrell, Joanne Lynn, Lee Goldman, Russell S. Phillips, Alfred F. Connors, Neal V. Dawson, William J. Fulkerson, Robert M. Califf, Norman Desbiens, Peter Layde, Robert K. Oye, Paul E. Bellamy, Rosemarie B. Hakim, and Douglas P. Wagner. 1995. The support prognostic model: Objective estimates of survival for seriously ill hospitalized adults. *Annals of Internal Medicine*, 122(3):191–203.

Ivan Kobyzev, Simon Prince, and Marcus Brubaker. 2020. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1.

Michael T. Koller, Heike Raatz, Ewout W. Steyerberg, and Marcel Wolbers. 2012. Competing risks and the clinical community: irrelevance or ignorance? *Statistics in Medicine*, 31(11-12):1089–1097.

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. 2018. Variational Inference

of Disentangled Latent Concepts From Unlabeled Observations. In *International Conference on Learning Representations*.

Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. 2019. Time-to-event prediction with neural networks and cox regression. *Journal of Machine Learning Research*, 20(129):1–30.

Håvard Kvamme and Ørnulf Borgan. 2019. Continuous and discrete-time survival prediction with neural networks.

Pamela J. LaMontagne, Tammie LS. Benzinger, John C. Morris, Sarah Keefe, Russ Hornbeck, Chengjie Xiong, Elizabeth Grant, Jason Hassenstab, Krista Moulder, Andrei G. Vlassenko, Marcus E. Raichle, Carlos Cruchaga, and Daniel Marcus. 2019. Oasis-3: Longitudinal neuroimaging, clinical, and cognitive dataset for normal aging and alzheimer disease. *medRxiv*.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. 2016. Fractalnet: Ultra-deep neural networks without residuals.

Changhee Lee, William Zame, Jinsung Yoon, and Mihaela van der Schaar. 2018. Deephit: A deep learning approach to survival analysis with competing risks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. 2020. Scalable gradients for stochastic differential equations.

Romain Lopez, Jeffrey Regier, Michael B. Cole, Michael I. Jordan, and Nir Yosef. 2018. Deep generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12):1053–1058.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. 2017. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations.

Zhanshan Ma and Axel W. Krings. 2008. Survival analysis approach to reliability, survivability and prognostics and health management (phm). In *2008 IEEE Aerospace Conference*, pages 1–20.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. 2016. Auxiliary deep generative models. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1445–1453, New York, New York, USA. PMLR.

Kaspar Märtens and Christopher Yau. 2020. Neural decomposition: Functional anova with variational autoencoders. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2917–2927. PMLR.

Ulla B. Mogensen, Hemant Ishwaran, and Thomas A. Gerds. 2012. Evaluating random forests for survival analysis using prediction error curves. *Journal of Statistical Software, Articles*, 50(11):1–23.

Kaspar Märtens and Christopher Yau. 2020. BasisVAE: Translation-invariant feature-level clustering with Variational Autoencoders. *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. 2019. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453.

Samet Ozturk, Vasilis Fthenakis, and Stefan Faulstich. 2018. Assessing the factors impacting on the reliability of wind turbines via survival analysis—a case study. *Energies*, 11(11).

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. Normalizing flows for probabilistic modeling and inference.

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. 2019a. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

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. 2019b. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 edition. Cambridge University Press, USA.

Yeping Lina Qiu, Hong Zheng, and Olivier Gevaert. 2020. Genomic data imputation with variational auto-encoders. *GigaScience*, 9(8). Giaa082.

Christopher Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. 2019. Diffeqflux.jl - A julia library for neural differential equations. *CoRR*, abs/1902.02376.

Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. 2018. A comparison of automatic differentiation

and continuous sensitivity analysis for derivatives of differential equation solutions. *CoRR*, abs/1812.01892.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. 2020. Universal differential equations for scientific machine learning.

Christopher Rackauckas and Qing Nie. 2017a. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1).

Christopher Rackauckas and Qing Nie. 2017b. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1). Exported from https://app.dimensions.ai on 2019/05/05.

Christopher Rackauckas and Qing Nie. 2017c. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1).

Rajesh Ranganath, Adler Perotte, Noémie Elhadad, and David Blei. 2016. Deep survival analysis.

Danilo Jimenez Rezende and Shakir Mohamed. 2016. Variational inference with normalizing flows.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1278–1286, Bejing, China. PMLR.

David Rindt, Robert Hu, David Steinsaltz, and Dino Sejdinovic. 2021. Survival regression with proper scoring rules and monotonic neural networks.

Davide Risso, Fanny Perraudeau, Svetlana Gribkova, Sandrine Dudoit, and Jean-Philippe Vert. 2018. A general and flexible method for signal extraction from single-cell rna-seq data. *Nature Communications*, 9(1):284.

Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. 2019. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, volume 32, pages 5320–5330. Curran Associates, Inc.

Lawrence F Shampine. 2018. *Numerical solution of ordinary differential equations*. Routledge.

Nikola Simidjievski, Cristian Bodnar, Ifrah Tariq, Paul Scherer, Helena Andres Terre, Zohreh Shams, Mateja Jamnik, and Pietro Liò. 2019. Variational autoencoders for cancer data integration: Design principles and computational practice. *Frontiers in Genetics*, 10:1205.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. 2015. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, pages 3483–3491. Curran Associates, Inc.

C. Spearman. 1904. "general intelligence," objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292.

Vasilis Stavrinides, Georgios Papageorgiou, Dominic Danks, Francesco Giganti, Nora Pashayan, Bruce Trock, Alex Freeman, Yipeng Hu, Hayley Whitaker, Clare Allen, Alex Kirkham, Shonit Punwani, Geoffrey Sonn, Dean Barratt, Mark Emberton, and Caroline M. Moore. 2021. Mapping psa density to outcome of mri-based active surveillance for prostate cancer through joint longitudinal-survival models. *Prostate Cancer and Prostatic Diseases*, 24(4):1028–1031.

Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. 2015. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820.

Michael E. Tipping and Christopher M. Bishop. 1999. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society Series B*, 61(3):611–622.

Belinda Tzen and Maxim Raginsky. 2019. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit.

Ivan Ustyuzhaninov, Ieva Kazlauskaite, Carl Henrik Ek, and Neill Campbell. 2020. Monotonic gaussian process flows. In *International Conference on Artificial Intelligence and Statistics*, pages 3057–3067. PMLR.

Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. 2009. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM J. Sci. Comput.*, 31(4):2549–2567.

Jennifer L. Whitwell, William R. Crum, Hilary C. Watt, and Nick C. Fox. 2001. Normalization of cerebral volumes by use of intracranial volume: Implications for longitudinal quantitative mr imaging. *American Journal of Neuroradiology*, 22(8):1483–1489.

Peng Xu, Jackie Chi Kit Cheung, and Yanshuai Cao. 2020. On variational learning of controllable representations for text without supervision. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10534–10543. PMLR.

Alexandra L. Young et al. 2018. Uncovering the heterogeneity and temporal complexity of neurodegenerative diseases with subtype and stage inference. *Nature Communications*, 9(1):4273.

Safoora Yousefi, Fatemeh Amrollahi, Mohamed Amgad, Chengliang Dong, Joshua E. Lewis, Congzheng Song, David A. Gutman, Sameer H. Halani, Jose Enrique Velazquez Vega, Daniel J. Brat, and Lee A. D. Cooper. 2017. Predicting clinical outcomes from large scale cancer genomic profiles with deep survival models. *Scientific Reports*, 7(1):11707.

Chun-Nam Yu, Russell Greiner, Hsiu-Chin Lin, and Vickie Baracos. 2011. Learning patient-specific cancer survival distributions as a sequence of dependent regressors. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Hong Zhang and Adrian Sandu. 2014. Fatode: a library for forward, adjoint, and tangent linear integration of odes. *SIAM Journal on Scientific Computing*, 36(5):C504–C523.

Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. 2016. Polynet: A pursuit of structural diversity in very deep networks.

Xinliang Zhu, Jiawen Yao, and Junzhou Huang. 2016. Deep convolutional neural network for survival analysis with pathological images. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 544–547.

Xinliang Zhu, Jiawen Yao, Feiyun Zhu, and Junzhou Huang. 2017. Wsisa: Making survival prediction from whole slide histopathological images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6855–6863.