Neural Networks for Functional and Survival Data

by

Sidi Wu

M.Sc., The George Washington University, 2016
B.A., University of International Relations, 2014

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
Department of Statistics and Actuarial Science
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2024

Declaration of Committee

Name: **Sidi Wu**

Degree: **Doctor of Philosophy**

Thesis title: **Neural Networks for Functional and Survival Data**

Committee:

Chair: Harsha Perera
Lecturer, Statistics and Actuarial Science

Jiguo Cao
Co-supervisor
Professor, Statistics and Actuarial Science

Cédric Beaulac
Co-supervisor
Professor, Mathematics
Université du Québec à Montréal

Haolun Shi
Committee Member
Assistant Professor, Statistics and Actuarial Science

Liangliang Wang
Examiner
Associate Professor, Statistics and Actuarial Science

Chongzhi Di
External Examiner
Professor
Biostatistics Program, Public Health Sciences Division
Fred Hutchinson Cancer Center

Abstract

The integration of advanced statistical methods with cutting-edge machine learning techniques has attracted substantial attention. Within this convergence, functional data analysis (FDA) and survival analysis are two pivotal areas where traditional statistical tools often encounter limitations in capturing the intricacies of dynamic processes and time-to-event outcomes. FDA is a statistical discipline that analyzes curves, surfaces and any random variables defined across infinite-dimensional spaces for various statistical tasks, including functional regression and functional data representation. We take advantage of neural networks, proposing novel models to tackle the scarce exploration of nonlinear regression analysis with scalar predictors and a functional response. In addition, a neural network-based approach is developed to address nonlinear representation learning and direct curve smoothing of discrete functional data concurrently. Time-to-event prediction, the task of predicting the time until the occurrence of a particular event of interest based on the characteristics of individuals, is a fundamental challenge in survival analysis and finds applications across diverse fields. We propose a simplified strategy to analyze right-censored survival outcomes using neural networks, enhancing estimation accuracy and computational efficiency in model discrimination in comparison to several existing survival neural networks.

Keywords: Neural networks; Functional data analysis; Functional regression; Representation learning; Survival analysis; Time-to-event prediction

Dedication

To my parents Biying Lin and Xuechao Wu.

Acknowledgements

I would first like to express my sincere gratitude to my supervisors, Dr. Jiguo Cao and Dr. Cédric Beaulac, for their unwavering support, continuous encouragement and invaluable guidance throughout the course of my Ph.D. Their expertise, patience, and mentorship have been instrumental in shaping my research direction and academic growth.

I am also grateful to the members of my thesis committee, Dr. Haolun Shi, Dr. Liangliang Wang, and Dr. Chongzhi Di, for their valuable feedback and constructive criticism, which greatly contributed to the improvement of this work.

Additionally, I would like to acknowledge the contributions of my collaborators, Dr. Shu Jiang, Dr. Farouk S. Nathoo, Dr. Michelle F. Miranda, Dr. Mirza Faisal Beg, Dr. Eugene Opoku, Dr. Yunlong Nie, Erin Gibson and Jie Wang, for their guidance, collaboration, expertise, and shared insights that have enhanced the scope and depth of my research and broadened my knowledge in statistics.

My heartfelt appreciation is extended to all faculty members and staff in the Department of Statistics and Actuarial Science at Simon Fraser University. Thank you for being there whenever I needed help. Many thanks to my fellow graduate students in the department for their camaraderie, encouragement, and insightful discussions. The delightful and challenging times we spent together are precious memories that I will never forget.

Last but not least, I am deeply indebted to my family and friends who supported me throughout my academic journey. Deepest thanks to my parents, whose endless love, encouragement, patience, and understanding have been the cornerstone of my academic pursuits. Special thanks to my cousin, Xiaoqin Wu, whose care and love made Vancouver a home to me. Warmest thanks to my best friend, Shuwei Yan, whose friendship has brought me immense joy over the years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The intersection of machine learning and statistical analysis has witnessed remarkable advancements, with neural networks emerging as powerful tools for analyzing different types of data. This thesis primarily focuses on the development of statistical models leveraging neural networks in the domains of functional data analysis (FDA) and survival analysis. Functional data, which consist of curves or functions defined over a continuum, are typically observed in a discrete manner. In this work, we seek novel approaches relying on neural networks to address challenges related to nonlinearity in regression and representation problems regarding discrete functional data. In addition, survival analysis refers to the study of time-to-event data, where the main emphasis lies in understanding the time until the occurrence of an event of interest. This work investigates a simplified framework to tackle time-to-event prediction through neural networks, providing a new perspective to model survival data subject to right censoring.

Functional regression is an active and trendy topic of research in FDA and has received wide attention in real applications and methodological developments. Among different types of functional regression problems, function-on-scalar regression (regression of a functional response on scalar predictors) is always a challenging task, especially when the relation between the response and the predictors is nonlinear. In Chapter 2, we develop a general framework of neural networks tailored for functional response, which aims at predicting smooth curves using the scalar inputs through a typical feed-forward neural network. To achieve the direct usage of functional data collected in a discrete form for network training, we slightly modify the objective function of the neural network, thereby ensuring the application of backpropagation, a gradient estimation method for neural network optimization. Numerical experiments demonstrate the superiority of the proposed models in relationship recovery and curve prediction under various nonlinear scenarios.

Representation learning of functional data is another fundamental but critical problem of interest in FDA. A standard pipeline in FDA is to first convert the discretely observed data to smooth functions, and then represent smooth functional data by some finite-dimensional vector of scalar values summarizing the information carried by the functions.

Basis expansion and functional principal component analysis are the two most widely recognized and commonly employed approaches for dimensional reduction of functional data. Both methods concentrate on learning the linear representation of functional data, yet in existing literature, limited attention has been paid to exploring the nonlinear mappings from the data space to the representation space, especially with deep learning techniques. To bridge this divide, we propose a solution to the nonlinear representation learning using a novel functional autoencoder (FAE) based on densely feed-forward neural networks in Chapter 3. The architecture of the proposed FAE is partially based on the neural network for functional output introduced in Chapter 2, integrating a newly designed input layer to directly input discrete functional data. Distinguished from the majority of the existing representation learning methods that require smooth functional inputs, our approach is dedicated to performing unsupervised representation learning and direct curve smoothing for discrete functional observations simultaneously, eliminating the conduct of curve smoothing assuming any particular form in advance. The advantages of the proposed FAE in reconstructing discrete data as smooth curves and extracting informative representation for classification are illustrated in different simulation designs.

In Chapter 4, we divert our attention to integrate neural networks into time-to-event prediction. With the expansion of modern data sets, the conventional approaches modelling biomedical covariates and a time-to-event outcome, e.g., Cox regression (Cox, 1972), become computationally challenging, prompting the emergence of survival neural networks as an alternative approach. Existing survival neural networks mainly focus on constructing neural networks directly using time-to-event outcome, each with its customized loss function and distinct training procedure to address right censoring. These customizations, however, introduce arbitrariness and bias into the estimation and increase the computational burden in network training. We instead propose a simplified survival neural network in the form of a three-step pipeline that does not require any modification of the loss function or training procedure to tackle time-to-event prediction. Numerical studies demonstrate that our simplified survival neural networks is computationally efficient and statistically competitive in model discrimination comparing to several existing survival neural networks.

Chapter 2

Neural Networks for Scalar Input and Functional Output

The regression of a functional response on a set of scalar predictors can be a challenging task, especially if there is a large number of predictors, or the relationship between those predictors and the response is nonlinear. In this chapter, we propose a solution to this problem: a feed-forward neural network (NN) designed to predict a functional response using scalar inputs. First, we transform the functional response to a finite-dimensional representation and construct an NN that outputs this representation. Then, we propose to modify the output of an NN via the objective function and introduce different objective functions for network training. The proposed models are suited for both regularly and irregularly spaced data, and a roughness penalty can be further applied to control the smoothness of the predicted curve. In our experiments, we demonstrate that our models outperform the conventional function-on-scalar regression model in multiple scenarios while computationally scaling better with the dimension of the predictors. See Wu et al. (2023) for the published version of this chapter.

2.1 Introduction

Functional data analysis (FDA) is a rapidly developing branch of statistics which targets at the theory and analysis of functional variables. As the atom of FDA, functional variables or functional data refer to curves, surfaces and any random variables taking values in an infinite dimensional space, such as time and spatial space (Ramsay and Silverman, 2005; Ferraty and Vieu, 2006). A basic and commonly recognized framework in FDA is to regard the functional data as realizations of an underlying stochastic process (Wang et al., 2016a), and indeed, a large fraction of data coming from different fields can be characterized as functional data. Figure 2.1, for example, illustrates the fertility rates over the age of females for 92 countries. Each curve is treated as a smooth function of age and can be viewed as functional data. The fertility curves were measured at several time points, each of which represents an age group

in increasing order. This age-specific fertility rate (ASFR) data was collected from the United Nations Gender Information (UNGEN) database by Mehrotra and Maity (Suchit Mehrotra, 2022). They estimated the observations based on the surveys conducted between 2000 and 2005. The data set is publicly available at `https://github.com/suchitm/fosr_clust`.

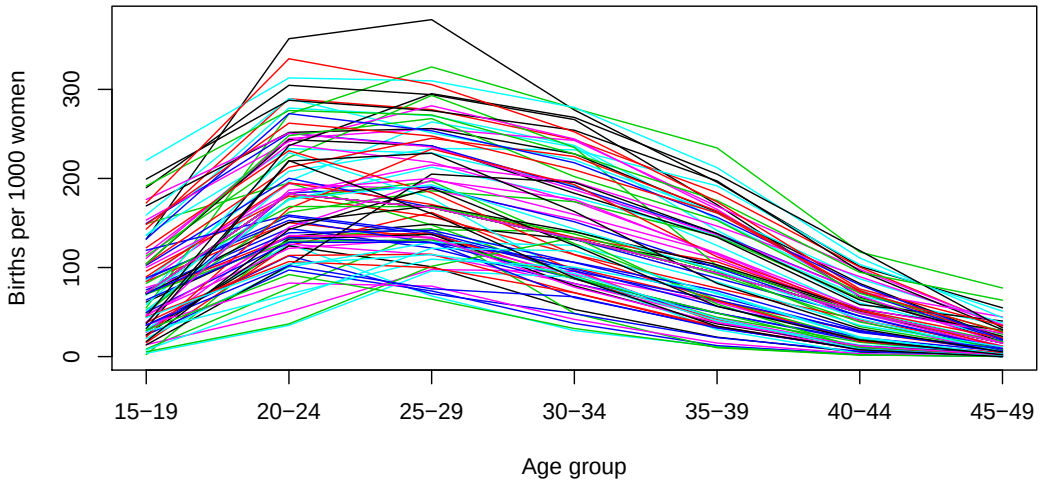

Figure 2.1: Age-specific fertility rates for 92 countries around the world.

Among different aspects of FDA, functional regression has received the most attention in applications and methodological development (Morris, 2015). Functional regression, in general, can be divided into three types: (1) scalar-on-function regression (regression analysis of scalar responses on functional predictors)(Lin et al., 2017), (2) function-on-function regression (predictors and responses of the regression are both functional) (Cai et al., 2021b,a), and (3) function-on-scalar (FoS) regression (regression of functional responses on scalar predictors, also named as the varying-coefficient model)(Cai et al., 2022). While plenty of work has been done for the first two scenarios, studies discussing functional response regression remain scarce. Functional linear models (FLM), a series of extensions of the classic linear models, are the most conventional and widely applied methods in dealing with regression problems. A general form of FLM for FoS regression can be written as

$$Y(t) = \boldsymbol{X}\boldsymbol{\beta}(t) + \epsilon(t), \tag{2.1}$$

where $Y(t)$ is the functional response, and $\boldsymbol{X}$ is a vector of scalar covariates. The unknown vector $\boldsymbol{\beta}(t)$ consists of parameter functions varying over t, and $\epsilon(t)$ stands for the random error term. In FDA, the consensual approach is to represent functional data with a linear combination of a finite number of known basis functions. Ramsay & Silverman (Ramsay and Silverman, 2005) introduced projecting functional response with basis functions like B-splines

for fitting the FoS regression model. Chiou et al. (Chiou et al., 2004, 2003) proposed to use functional principal component analysis (FPCA) for dimensional reduction and represent the functional response with eigenfunctions obtained via spectral decomposition of the covariance function of $Y(t)$. Both approaches summarize the information carried by the functional variable $Y(t)$ to a finite-dimensional vector of scalar representations (B-spline basis coefficients or functional principal component scores (FPC scores)). In solving linear functional regression problems, taking FoS regression as an example, the linear relation between the scalar predictors and the functional response is indeed captured by fitting a linear relation between the predictors and the scalar representation.

Some nonlinear approaches have been developed to handle more complicated regression settings. Zhang and Wang (Zhang and Wang, 2014) combined the FoS with the additive models to gain the varying-coefficient additive model for functional data which does not require the linearity assumption between the scalar predictors and the functional response. Afterwards, an extension to this varying-coefficient additive model, named as functional additive mixed (FAM) model, was established by Scheipl et al. (Fabian Scheipl and Greven, 2015) with a general form

$$Y(t) = \sum_{r=1}^{R} f_r\left(\boldsymbol{X}_r, t\right) + \epsilon(t), \tag{2.2}$$

for functional response $Y(t)$. Each term in the additive predictor $f_r(\boldsymbol{X}_r, t)$ is a function of t and a subset $\boldsymbol{X}_r$ of the complete predictor set $\boldsymbol{X}$ including scalar and functional covariates. This extensive framework can consist of both linear and nonlinear effects of functional and scalar covariates that may vary smoothly over the index of the functional response. Naturally, this model was further extended, by Scheipl et al (Fabian Scheipl and Greven, 2016), to the generalized functional additive mixed model in order to take care of non-Gaussian functional response. Although several nonlinear attempts have been made, the existing regression methods for FDA have been predominantly linear (Wang et al., 2016a). Considering the emerging trend of complicatedly structured functional data, there is an increasing demand to develop more nonlinear approaches to FDA.

In this work, we propose a solution to the FoS problem which is able to handle a large number of predictors and the nonlinear relation between the scalar predictors and the functional response. Our solution borrows from the machine learning literature, which follows a trend in adapting machine learning techniques to known statistical problems such as survival analysis (Ishwaran et al., 2008; Katzman et al., 2018; Beaulac et al., 2018) or causal inference (Schölkopf et al., 2021; Lecca, 2021). We adapt the NN architecture for functional data.

In some existing works, efforts have been made to combine deep NNs to the field of functional data analysis. For instance, Rossi et al.(Rossi et al., 2002; Rossi and Conan-Guez, 2005) firstly explored the idea of applying NNs to functional data by constructing

a functional neural network (FNN) with functional neurons in the first hidden layer for functional inputs. FNN was then extended by Thind et al. (Thind et al., 2023, 2020) to feed both functional and scalar coviarates as inputs and outputs a scalar response. Rao et al. (Rao et al., 2020) equipped FNN with geographically weighted regression and spatial autoregressive technique to handle regression problems with spatially correlated functional data. Meanwhile, Wang et al.(Wang et al., 2020) proposed a nonlinear function-on-function model using a fully connected NN. Previously, Yao et al. (Yao et al., 2021) developed an NN with a new basis layer whose hidden neurons are micro NNs, to perform a parsimonious dimension reduction for functional inputs using information relevant to the scalar target. (Wang and Cao, 2023a) introduced a functional nonlinear learning approach to adequately represent multivariate functional data within a reduced-dimensional feature space. (Wang and Cao, 2023b) proposed a nonlinear prediction method for functional time series.

Most of the mentioned works are focused on building NNs with functional inputs and scalar outputs. In this work, we consider the other side of the coin. We design an NN meant to predict a functional response and to the best of our knowledge, this study is the first attempt to solve the FoS regression problem using artificial NNs. Because the standard machine learning techniques are designed for finite-dimensional feature vectors, we propose to encode the information contained in the intrinsically infinite-dimensional functional response to a finite-dimensional vector of scalar representations. Different from other nonlinear approaches targeting at FoS regression problems, our methods, in particular, focus on studying the nonlinear association between the predictors and the scalar representations of the functional response, to further reveal the relation between the scalar predictors and the functional response. In this way, we maintain the interpretability of the relation between the scalar predictors and the functional response, despite of the usage of NNs.

Challenged by the ASFR data introduced previously, in this work, we are interested in conducting a FoS analysis to accurately predict the functional curves using scalar predictors and also reveal any potential nonlinear relation between the predictors and the response. For each of the 92 observations (countries), the functional curve of fertility rates is associated with 15 demographic and socioeconomic variables averaging over the information available on Gapminder in a country-level manner from 2000 to 2005 (Suchit Mehrotra, 2022). These 15 scalar covariates consist of age at first marriage, under-5 mortality, maternity deaths per 1000 women, cervical cancer deaths per 100,000 women, female labor force participation, male to female ratio (women aged from 15-49), contraception prevalence (women aged from 15-49), life expectancy, mean years of school (women % men, women aged from 15-34), female's body mass index (BMI), the number of births attended by trained birth staff (% of total), gross domestic product (GDP) per capita, the proportion of dollar billionaires per 1 million people, health expenditures (% of GDP) and the amount of alcohol its populace consumes. All the covariates downloaded from the listed data source have been previously standardized, and many predictor pairs among them are found highly correlated.

Due to the difficulty in directly determining whether there is a nonlinear relation between a scalar variable and a random function, we pay more attention to the association between the scalar covariates and the scalar representations summarizing the information of the functional response. Regarding the scalar representations, we choose to use the basis coefficients which are estimated by approximating the response function with a linear expansion of the most common B-spline basis functions. Besides the initial choice of the basis coefficients, we also attempt to implant the FPC scores as the scalar representations in the real application.

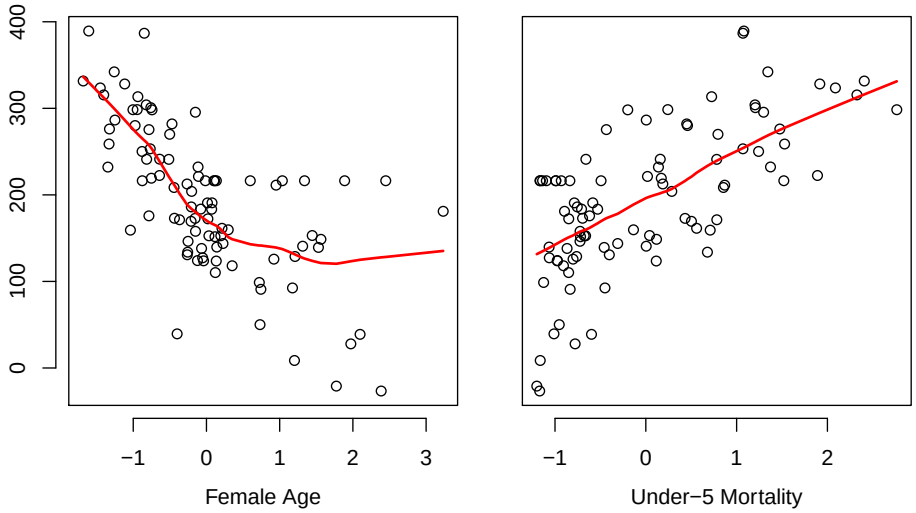

Figure 2.2: The second B-spline basis coefficient (estimated using the functional response $Y(t)$) versus two scalar covariates, including female age and under-5 mortality, separately. The red lines are the smoothing curves estimated using `R` function `lowess()`.

Accordingly, with the ASFR data set, instead of picturing the relation between the infinite-dimensional response curve and the 15 scalar covariates, we obtain visualizations of one B-spline basis coefficient against one predictor for all possible combinations. The representation process is purely based on the fertility trajectories with no contribution from the 15 predictors. Six B-spline basis functions are employed, which is recommended by the cross-validation we performed. Figure 2.2 displays the in-pair associations for some selected basis coefficient-predictor pairs. We incorporate a locally-weighted smoothing curve in red to help visualizing if the association is linear or not. The existence of nonlinear associations between some coefficient-predictor pairs is endorsed, as shown with the female age predictor in Figure 2.2. On the other hand, the association pattern of the second basis coefficient to under-5 mortality is basically linear as depicted in Figure 2.2. As both linear and nonlinear patterns are detected among all possible basis coefficient-predictor sets, we would like to set up a procedure with the ability to include every available covariate to precisely predict

the response trajectory, while simultaneously capturing the true relationships between the predictors and the scalar representations of the functional response.

Three main contributions are made in this work. First, we construct a general framework of NN for functional response which aims at predicting curves using the input through a feed-forward NN architecture by slightly modifying the output of the NN via the objective function. Such modifications are readily applicable to other types of networks such as convolutional neural networks (CNN) (Zhu et al., 2016), and long short-term memory (LSTM) model (Hochreiter and Schmidhuber, 1997; Greff et al., 2017; Gers et al., 2000). Second, we pay attention to the behavior of the functional output during the network training process and propose to hold their natural smoothness property by adding a roughness penalty in the objective function for training NN. Lastly, we make numerous comparisons under various scenarios to analyze and conclude the conditions of use for different versions of our methods.

The biggest challenge to the implementation of the aforementioned contributions is to ensure the NN models can be trained using backpropagation as a standard NN (Rumelhart et al., 1986). Theoretically, we need to define the flow of information from the input to the objective function as a sequence of operations differentiable with respect to the weights of the NN. However, in practice, because we rely on an auto-differentiation package, such as **Keras** (Allaire and Chollet, 2020), to train the NN, we need to define the objective function as a sequence of operations provided by that package. The available differential operations are incapable of dealing with derivatives of the integral of the infinite-dimensional functional response with respect to NN weights. Besides, some common features when fitting functional data, such as roughness penalties, were obviously not designed with that consideration in mind. For example, it is common to penalize the integrated second-order derivative of a functional curve for smoothing in FDA. But a roughness penalty cannot be simply integrated into the objective function when training the NN. Overcoming that challenge is a central part of these contributions mentioned above and to do so, we needed to be creative and adapt the objective functions in various ways detailed later in this manuscript.

The remainder of this chapter is organized as follows. In Section 2.2, we detail the proposed NN with a functional response, with an additional discussion on how to control the smoothness of the predicted curves. A brief discussion about the computational costs of NNs and the FLM with functional response is provided in Section 2.3. Section 2.4 contains the results of prediction in the real application using different methods. In section 2.5, we conduct simulation studies, under linear and nonlinear scenarios, and compare the proposed models with the existing FoS model in both the predictive accuracy for discretely observed time points and the ability to reconstruct the true response trajectories over a continuum. Lastly, some concluding remarks are given in Section 2.6.

2.2 Methodology

Suppose we have N subjects and for the i-th subject, the input is a set of numerical variables $\boldsymbol{X}_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, and the output is a functional variable $Y_i(t), t \in \mathcal{T}$ in the $L^2(t)$ space. Functional data is often assumed to lie in the L^2 space because it ensures that the data is square-intergrable, allowing the optimization of the statistical models with respect to the squared-loss criterion in the FoS regression framework. In reality, the functional response $Y_i(t)$ is usually measured in a discrete manner, for instance, at m_i time points or locations. Therefore, instead of observing the full trajectory of $Y_i(t)$, for the i-th subject, we obtain m_i pairs of observations $\{t_{ij}, Z_i(t_{ij})\}$, $j = 1, 2, ...m_i$ and $Z_i(t_{ij}) = Y_i(t_{ij}) + \epsilon_i(t_{ij})$, where $\epsilon_i(t_{ij})$ is the i.i.d. observation error. To simplify the situation, in the following discussion, we assume that for all subjects, the functional term $Y_i(t)$'s are observed at the same m densely and equally spaced time points. In other words, we assume that $m_1 = m_2 = \cdots = m_N = m$ and $t_{1j} = t_{2j} = \cdots = t_{Nj} = t_j$ for $j = 1, 2, ..., m$. A summary of notations used throughout the manuscript is provided in Table A.1 in Appendix A.

2.2.1 Neural Network with Functional Response: Mapping to Basis Coefficients (NNBB)

To overcome the difficulty in modelling infinite-dimensional data, a common pipeline in FDA is to summarize the information of functions $\{Y_i(t)\}_{i=1}^N$ into a set of finite-dimensional vectors of coefficients using some basis representation method, and consequently,

$$Y_i(t) = \sum_{k=1}^{K_b} c_{ik}\theta_k(t) = \boldsymbol{\theta}'\boldsymbol{C}_i, \tag{2.3}$$

where K_b is the number of basis functions. It is common practice to select K_b by cross-validation (Ramsay and Silverman, 2005). The vector $\boldsymbol{\theta}$ contains the basis functions $\theta_1(t), ..., \theta_{K_b}(t)$ from a selected basis system, such as the Fourier basis system or B-spline system (in our study, we mainly used the latter), and $\theta_k(t)$ also belongs to L^2 space. $\boldsymbol{C_i}$ is a K_b-dimensional vector of basis coefficients $\{c_{ik}\}_{k=1}^{K_b}$ which lies in $\mathbb{R}^{K_b}$ and needs to be determined. These basis coefficients serve as parameters of this model. Without the smooth underlying functions $Y_i(t)$'s, the discrete observations $Z_i(t_{ij})$'s are used to estimate $\{c_{ik}\}_{k=1}^{K_b}$ by fitting $Z_i(t_{ij}) = \sum_{k=1}^{K_b} c_{ik}\theta_k(t_{ij}) + \epsilon_i(t_{ij})$. The least square estimator of c_{ik}, denoted by c_{ik}°, is obtained by minimizing the sum of squared error (SSE) criterion $\text{SSE}(\boldsymbol{Z}|\boldsymbol{C}) = \sum_{i=1}^N \sum_{j=1}^m \left(Z_i(t_{ij}) - \sum_{k=1}^{K_b} c_{ik}\theta_k(t_{ij})\right)^2$.

Eq.(2.3) implies that $Y(t)$ can be approximated by a linear combination of the basis functions $\theta(t)$'s, which carry the fixed modes of variations over $\mathcal{T}$ with real-valued basis coefficients. Learning how the predictors X's regress on the variation of $Y(t)$, as a result, can be naturally replaced with learning how the predictors X's regress on the set of unknown basis coefficients c's. Therefore we propose to set the basis coefficients c's as a function of

X's. Let $F(\cdot)$ be a mapping function from $\mathbb{R}^P$ to $\mathbb{R}^{K_b}$ so that $\boldsymbol{X}$ can be mapped to the basis coefficients as

$$\boldsymbol{C}_i = F(\boldsymbol{X}_i), \tag{2.4}$$

which in turn can be used to map $\boldsymbol{X}$ to the functional response $Y(t)$ with $Y_i(t) = \boldsymbol{\theta}' F(\boldsymbol{X}_i)$. Consequently, we can learn the nonlinear relation between $\boldsymbol{X}_i$ and $Y_i(t)$ by setting $F(\cdot)$ to be a nonlinear function. Here we propose a densely feed-forward NN as the mapping function $F(\cdot)$, where the basis coefficients $\{c_{i1}, c_{i2}, ..., c_{iK_b}\}$ are the outputs of the NN, then the model can be expressed as

$$\boldsymbol{C}_i = \mathrm{NN}_\eta(\boldsymbol{X}_i) = g_{L+1}\left(\cdots g_1\left(\sum_{p=1}^{P} w_{1p}X_{ip} + b_1\right)\right), \tag{2.5}$$

where $g_1, ..., g_{L+1}$ are the activation functions at each layer with L being the number of hidden layers, and η denotes the NN parameter set consisting of weights $\{w_{\ell p}\}_{\ell=1}^{L+1}$ and bias $\{b_\ell\}_{\ell=1}^{L+1}$ of all layers. The NN is optimized by minimizing the standard objective function calculating the mean squared error (MSE) as $L_{\boldsymbol{C}}(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^{K_b} (\hat{c}_{ik} - c_{ik})^2$, where n_{train} is the number of samples in the training set, and in application the *true* basis coefficients c_{ik}'s are replaced with ${c^\circ}_{ik}$'s.

We named this model NNBB as it is an NN, with B-spline coefficient output (B) and trained by minimizing the MSE of those B-spline coefficients (B). With the exception of the output layer, the NNBB uses a conventional NN architecture and thus, typical approaches for hyperparameter tuning and training can be applied. We summarize the training process of NNBB in Algorithm 1. The optimized NN is used for prediction where it takes the new scalar inputs in the test set and then outputs the predicted basis coefficients $\hat{\boldsymbol{C}}_{\text{new}}$. The predicted functional response $\hat{\boldsymbol{Y}}(t)$ is further constructed as $\hat{\boldsymbol{Y}}(t) = \boldsymbol{\theta}' \hat{\boldsymbol{C}}_{\text{new}}$.

2.2.2 Neural Network with Functional Response: Mapping to FPC Scores (NNSS)

Apart from basis expansion, the other popular approach for dimension reduction is FPCA. To be specific, let $\mu(t)$ and $K(t, t') = \mathrm{cov}(Y(t), Y(t'))$ be the mean and covariance functions of the underlying function $Y(t)$, and accordingly, the spectral decomposition of the covariance function is $K(t, t') = \sum_{k=1}^{\infty} \gamma_k \phi_k(t) \phi_k(t')$, where $\{\gamma_k, k \geq 1\}$ are the eigenvalues in decreasing order with $\sum_k \gamma_k < \infty$ and ϕ_k's are corresponding eigenfunctions, also named functional principle components (FPCs), with restriction of $\int \phi_k^2(t)dt = 1$ for all k. Following the Karhunen-Loève expansion, the i-th observed functional response $Y_i(t)$ can be represented as $Y_i(t) = \mu(t) + \sum_{k=1}^{\infty} \xi_{ik} \phi_k(t)$. Denote $\tilde{Y}_i(t) = Y_i(t) - \mu(t)$ as the centered functional response

Algorithm 1: Training NNBB

Input: $\boldsymbol{X}_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, $\{Z_i(t_{ij})\}_{j=1}^{m}$ in the training set n_{train}

Output: NN with the optimized parameter set $\hat{\eta}$

set up hyperparameters, including:
- for smoothing $Z_i(t_{ij})$: basis function vector $\boldsymbol{\theta} = [\theta_1(t), ..., \theta_{K_b}(t)]$, number of basis functions K_b
- for training NN: number of hidden layers (L), number of neurons per hidden layer, activation functions ($g_1, ..., g_{L+1}$), number of epochs (E), batch size, NN optimizer (with a learning rate ϱ), etc.

forall $i \in n_{\text{train}}$ **do** obtain $\{c_{ik}^{\circ}\}_{k=1}^{K_b}$ by projecting $\{Z_i(t_{ij})\}_{j=1}^{m}$ to the selected set of basis functions $\boldsymbol{\theta}$

randomly initialize NN parameter set and get $\eta = \eta_{\text{initial}}$

train a fully-connected NN with input $\boldsymbol{X}_i$ and output $\{c_{ik}^{\circ}\}_{k=1}^{K_b}$

for $e = 1$ **to** E **do**

I. *forward propagation*

(i) pass $\boldsymbol{X}_i$ through the NN and get $\{\hat{c}_{ik}\}_{k=1}^{K_b}$ for all $i \in n_{\text{train}}$

(ii) calculate $L(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^{K_b} (\hat{c}_{ik} - c_{ik}^{\circ})^2$

II. *backward propagation*

(i) update NN parameter set as $\eta^* = \eta - \varrho \frac{\partial L(\eta)}{\partial \eta}$

III. **if** $e < E$ **then** set $\eta = \eta^*$, and then repeat I. & II.

else return η^*

end

return NN with the estimated parameter set $\hat{\eta} = \eta^*$

for the i-th subject, we can express the expansion as

$$\tilde{Y}_i(t) = \sum_{k=1}^{\infty} \xi_{ik} \phi_k(t), \tag{2.6}$$

where $\xi_{ik} = \int \{Y_i(t) - \mu(t)\} \phi_k(t) dt$ is the k-th FPC score for $Y_i(t)$ with zero mean and $\text{cov}(\xi_{ik}, \xi_{il}) = \gamma_k \cdot \mathbb{1}(k = l)$. Representing functions with eigenfucntions eigen-decomposed from $\text{cov}(Y(t), Y(t'))$ and the corresponding FPC scores can be considered as a special case of basis expansion, where eigenfunctions play the role of basis functions which, however, are unknown and need to be numerically calculated using the observed functional data. Given a desired proportion of variance explained (τ) by the FPCs, $\tilde{Y}_i(t)$ can be approximated arbitrarily well by a finite number of the leading FPCs, where

$$\tilde{Y}_i(t) \approx \sum_{k=1}^{K_\tau} \xi_{ik} \phi_k(t), \tag{2.7}$$

and K_τ is a valued parameter truncating the FPCs and reducing the dimension of $Y(t)$. K_τ is determined by τ as the smallest integer satisfying $\sum_{k=1}^{K_\tau}\gamma_k \geq \tau$. Because of the fast decay rate of γ_k, K_τ is usually an small integer. In pratice, FPCA can be easily performed in `R` with the help of some FDA-related packages, e.g., **fda** (Ramsay et al., 2020) and **fdapace** (Carroll et al., 2020). The **fda** package requires the users to firstly smooth the discrete observations $Z_i(t_{ij})$'s and then apply FPCA on the smoothed functional data for $\{\xi_{ik}^{\circ}\}_{k=1}^{K_\tau}$ and $\{\hat{\phi}_k(t)\}_{k=1}^{K_\tau}$, an estimator of $\{\xi_{ik}\}_{k=1}^{K_\tau}$ and $\{\phi_k(t)\}_{k=1}^{K_\tau}$, respectively, while the **fdapace** package is able to estimate $\{\xi_{ik}\}_{k=1}^{K_\tau}$ and $\{\phi_k(t)\}_{k=1}^{K_\tau}$ with observed data $Z_i(t_{ij})$'s directly.

It is clearly shown in Eq.(2.7) that $\tilde{Y}_i(t)$ can be effectively approximated by a linear combination of the top eigenfunctions $\{\phi_k(t)\}_{k=1}^{K_\tau}$, with the major modes of variations among $\tilde{Y}_i(t)$ captured. Following the idea in Wang & al. (Wang et al., 2020), under the regression setting, in order to learn the impact of different values of $\boldsymbol{X}_i$ on the variation of $\tilde{Y}_i(t)$, we propose to set the coefficients $\boldsymbol{\xi}_i = \{\xi_{i1}, \xi_{i2}, ..., \xi_{iK_\tau}\}$ to be a function of $\boldsymbol{X}_i$, by formulation, $\boldsymbol{\xi}_i = F(\boldsymbol{X}_i)$ and consequently we have

$$\tilde{Y}_i(t) = \boldsymbol{\phi}' F(\boldsymbol{X}_i), \tag{2.8}$$

where $\boldsymbol{\phi}$ is a K_τ-dimensional vector cataloging the K_τ leading FPCs. To model the nonlinear relation, similarly, we proposed to set the mapping function $F(\cdot)$ as a densely feed-forward NN, and then we can write

$$\boldsymbol{\xi}_i = \mathrm{NN}_\eta(\boldsymbol{X}_i) = g_{L+1}\left(\cdots g_1\left(\sum_{p=1}^{P} w_{1p}X_{ip} + b_1\right)\right), \tag{2.9}$$

where $g_1, ..., g_{L+1}$ are the activation functions at each layer with L being the number of hidden layers, and η denotes the NN parameter set consisting of weights $\{w_{\ell p}\}_{\ell=1}^{L+1}$ and bias $\{b_\ell\}_{\ell=1}^{L+1}$ of all layers. Likewise, $\mathrm{NN}_\eta(\boldsymbol{X}_i)$ is trained by minimizing the MSE loss function $L_{\boldsymbol{\xi}}(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^{K_\tau} (\hat{\xi}_{ik} - \xi_{ik})^2$, where n_{train} stands for the number of training set subjects, and the *true* FPC scores ξ_{ik}'s are replaced by ξ_{ik}°'s.

Following the same naming convention, we refer to this model has NNSS because it is an NN model build with output being the FPC scores (S) trained on the scores (S) themselves.The complete training process of NNSS is summarized in Algorithm 2. Again, the hyperparameter tuning and model training of NNSS are the same as of a conventional NN. Meanwhile, the returned NN with optimized parameters set $\hat{\eta}$ is used for prediction where it takes the new scalar inputs in the test set and then outputs the predicted FPC scores $\hat{\boldsymbol{\xi}}_{\text{new}}$. The predicted functional response $\hat{\boldsymbol{Y}}(t)$ is further recovered as $\hat{\boldsymbol{Y}}(t) = \hat{\mu}(t) + \hat{\boldsymbol{\phi}}'\hat{\boldsymbol{\xi}}_{\text{new}} = \hat{\mu}(t) + \hat{\boldsymbol{\phi}}'\mathrm{NN}(\boldsymbol{X}_{\text{new}}|\hat{\eta})$, where $\hat{\boldsymbol{\phi}}$ is the vector consisting of the estimated FPCs $\hat{\phi}_1(t), ..., \hat{\phi}_{K_\tau}(t)$.

Algorithm 2: Training NNSS

Input: $\boldsymbol{X}_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, $\{Z_i(t_{ij})\}_{j=1}^{m}$ in the training set n_{train}

Output: NN with the optimized parameter set $\hat{\eta}$

set up hyperparameters, including:
- for performing FPCA: the desired proportion of variance explained τ for determining K_τ
- for training NN: number of hidden layers (L), number of neurons per hidden layer, activation functions ($g_1, ..., g_{L+1}$), number of epochs (E), batch size, NN optimizer (with a learning rate τ), etc.

forall $i \in n_{\text{train}}$ **do**

  estimate the mean function $\mu(t)$ and the covariance function $K(t, t')$ using $\{Z_i(t_{ij})\}_{j=1}^{m}$

  perform eigen-decomposition on $\hat{K}(t, t')$ and get $\{\hat{\phi}_k(t)\}_{k=1}^{K_\tau}$ and the corresponding FPC scores $\{\xi_{ik}^{\circ}\}_{k=1}^{K_\tau}$

end

randomly initialize NN parameter set and get $\eta = \eta_{\text{initial}}$

train a fully-connected NN with input $\boldsymbol{X}_i$ and output $\{\xi_{ik}^{\circ}\}_{k=1}^{K_\tau}$

for $e = 1$ **to** E **do**

  I. *forward propagation*

    (i) pass $\boldsymbol{X}_i$ through the NN and get $\{\hat{\xi}_{ik}\}_{k=1}^{K_\tau}$ for all $i \in n_{\text{train}}$

  (ii) calculate $L(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{k=1}^{K_\tau} (\hat{\xi}_{ik} - \xi_{ik}^{\circ})^2$

  II. *backward propagation*

  (i) update NN parameter set as $\eta^* = \eta - \varrho \frac{\partial L(\eta)}{\partial \eta}$

  III. **if** $e < E$ **then** set $\eta = \eta^*$, and then repeat I. & II.

    else return η^*

end

return NN with the estimated parameter set $\hat{\eta} = \eta^*$

2.2.3 Modification to the Objective Function (NNBR and NNSR)

In the two previous sections, we define two NNs outputting basis coefficients or FPC scores, and those outputs are further used to construct the predicted response variable. Let us discuss a way to build an objective function that directly uses the response variable in order to estimate the parameters. For simplicity, we only discuss the B-spline model of Section 2.2.1 but know that a similar concept can also be applied to the FPCA model described in Section 2.2.2. In brief, in Section 2.2.1, we first fit a B-spline model on the observed functional response to estimate a set of basis coefficients for the response in the training set and then we train an NN to predict these basis coefficients using the predictors.

In this section, we propose to modify the objective function in order to bypass the initial estimation of basis coefficients. The key idea of the new objective function is to directly minimize the prediction error of the response variable. In other words, instead of minimizing

the MSE between $\hat{c}_{ik}$ and c_{ik}, here we first transform the NN output, the B-splines basis coefficients, into the predicted response $\hat{Y}_i(t)$ and then minimize the MSE between $\hat{Y}_i(t)$ and $Y_i(t)$. What supports us to implement such objective function is the fact that we rely on differentiable operations to build this new objective function. Doing so guarantees that we can rely on the backpropagation algorithm to train the NN using readily available packages.

Suppose that we want to fit the functional response with a B-spline made of K_b basis functions $\boldsymbol{\theta}$ and K_b associated basis coefficients $\boldsymbol{C}$. Then the predict response $\hat{Y}_i(t)$ at time t is the vector product between $\boldsymbol{\theta}$ and $\boldsymbol{C}_i$ as illustrated in Eq. (2.3). This means that the relation between the predicted response $\hat{Y}_i(t)$ and the predicted basis coefficients $\hat{\boldsymbol{C}}_i$ is linear, thus we can easily compute the derivative of $\hat{Y}_i(t)$ with respect to the coefficients. This further indicates that if we observe the response at time t then we are able to compute the gradient of $(Y_i(t) - \hat{Y}_i(t))^2$ with respect to the basis coefficients and therefore we can also compute the gradient with respect to the parameters η of the NN that outputs those basis coefficients.

More generally, assuming the response is observed at m time points $t_1, t_2, ..., t_m$ for every subject, we can generate a matrix of basis functions evaluated at those time points, denoted as Θ, which is a $K_b \times m$ matrix where each row represents a basis function with entries taking the values of that basis function at each of the m time points. Consequently, we can obtain the predicted response $\hat{Y}_{(n_{\text{train}} \times m)}$ for every m observed time points and for all n_{train} subjects by doing a simple matrix multiplication

$$\hat{Y}_{(n_{\text{train}} \times m)} = \hat{\boldsymbol{C}}_{(n_{\text{train}} \times K_b)} \Theta_{(K_b \times m)}, \tag{2.10}$$

where $\hat{\boldsymbol{C}}$ is a matrix where a row contains the K_b NN-estimated coefficients for a single observed subject. In our proposed model, $\hat{\boldsymbol{C}}$ is the output produced by the NN function as explained in Section 2.2.1. However, we now modify the objective function to train the NN to minimize the MSE between the observed response and the predicted one

$$L_{\boldsymbol{Y}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{m} (Y_i(t_{ij}) - \hat{Y}_i(t_{ij}))^2. \tag{2.11}$$

Note that in implementation, the discrete observation $Z_i(t_{ij}) = Y_i(t_{ij}) + \epsilon_i(t_{ij})$ will replace $Y_i(t_{ij})$, the underlying *true* functional curve. In this way, we entirely bypass the need to estimate the basis coefficients first and fit the parameters η with respect to the functional response directly. We call this variant NNBR because its architecture is an NN with basis coefficients output (B) fitted by minimizing the MSE of the response variable (R). The entire training process of NNBR is detailed in Algorithm 3. Similarly, we can get a prediction for a new input $\boldsymbol{X}_{\text{new}}$ as $\hat{Y}(t) = \boldsymbol{\theta}' \hat{\boldsymbol{C}}_{\text{new}} = \boldsymbol{\theta}' \text{NN}(\boldsymbol{X}_{\text{new}} | \hat{\eta})$.

When comparing NNBR to NNBB, we note that they share a similar architecture but use different objective functions. A significant difference between both models is that in order to

Algorithm 3: Training NNBR

Input: $\boldsymbol{X}_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, $\{Z_i(t_{ij})\}_{j=1}^m$ in the training set n_{train}

Output: NN with the optimized parameter set $\hat{\eta}$

set up hyperparameters, including:
- for smoothing $Z_i(t_{ij})$: basis function vector $\boldsymbol{\theta} = [\theta_1(t), ..., \theta_{K_b}(t)]$, number of basis functions K_b
- for training NN: number of hidden layers (L), number of neurons per hidden layer, activation functions ($g_1, ..., g_{L+1}$), number of epochs (E), batch size, NN optimizer (with a learning rate ϱ), etc.

evaluate $[\theta_1(t), ..., \theta_{K_b}(t)]$ at all observed timestamps $\{t_j\}_{j=1}^m$ and form the matrix $\Theta_{(K_b \times m)}$

randomly initialize NN parameter set and get $\eta = \eta_{\text{initial}}$

train a fully-connected NN with input $\boldsymbol{X}_i$ and output $\{Z_i(t_{ij})\}_{j=1}^m$

for $e = 1$ **to** E **do**

I. *forward propagation*

(i) pass $\boldsymbol{X}_i$ through the NN and get $\{\hat{c}_{ik}\}_{k=1}^{K_b}$ for all $i \in n_{\text{train}}$

(ii) multiply $\{\hat{c}_{ik}\}_{k=1}^{K_b}$ with $\Theta_{(K_b \times m)}$ to get $\{\hat{Y}(t_{ij})\}_{j=1}^m$

(iii) calculate $L(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^m (Z_i(ij) - \hat{Y}_i(t_{ij}))^2$

II. *backward propagation*

(i) update NN parameter set as $\eta^* = \eta - \varrho \frac{\partial L(\eta)}{\partial \eta}$

III. **if** $e < E$ **then** set $\eta = \eta^*$, and then repeat I. & II.

else return η^*

end

return NN with the estimated parameter set $\hat{\eta} = \eta^*$

train the NNBB we need to first estimate the basis coefficients of the functional response. Conversely, training the NNBR requires a modification of the objective function which slightly increases the computational cost but meanwhile bypasses the need of a beforehand estimation of basis coefficients.

A similar process could be conducted with an NN that outputs FPC scores as described in Section 2.2.2, where this time it predicts response $\hat{Y}$ using those scores. We named that model NNSR and describe the training process in Algorithm 4.

It is worth mentioning that the network architectures of NNBR and NNSR remain the same as a classic NN even though we have modified their objective functions. Such modifications only influence the network optimization with no changes to the sensitivity with respect to the hyperparameters and the architecture (number of layers and depth). Additionally, the proposed modification that targets at the minimization of the difference between $Y_i(t)$ and $\hat{Y}_i(t)$ is surprisingly simple, convenient and computationally efficient. Compared to the operation that minimizes the difference between c_{ik} and $\hat{c}_{ik}$ (NNBB) or

between ξ_{ik} and $\hat{\xi}_{ik}$ (NNSS), the modified objective function only requires an additional operation, namely a simple matrix multiplication as Eq. (2.10) in forward propagation and an additional step to compute the gradient of $(Y_i(t) - \hat{Y}_i(t))^2$ with respect to the basis coefficients outputted by NN in backward propagation. This only increases negligibly the computational cost of NNBR and NNSR when compared to NNBB or NNSS.

Algorithm 4: Training NNSR

Input: $\boldsymbol{X}_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, $\{Z_i(t_{ij})\}_{j=1}^m$ in the training set n_{train}

Output: NN with the optimized parameter set $\hat{\eta}$

set up hyperparameters, including:
- for performing FPCA: the desired proportion of variance explained τ for determining K_τ
- for training NN: number of hidden layers (L), number of neurons per hidden layer, activation functions ($g_1, ..., g_{L+1}$), number of epochs (E), batch size, NN optimizer (with a learning rate τ), etc.

forall $i \in n_{\text{train}}$ **do**
- estimate the mean function $\mu(t)$ and the covariance function $K(t, t')$ using $\{Z_i(t_{ij})\}_{j=1}^m$
- perform eigen-decomposition on $\hat{K}(t, t')$ and get $\{\hat{\phi}_k(t)\}_{k=1}^{K_\tau}$
- form $\hat{\Phi}_{(K_\tau \times m)}$, a matrix of eigenfunctions $[\hat{\phi}_1(t), ..., \hat{\phi}_{K_\tau}(t)]$ evaluated at all observed time points $\{t_j\}_{j=1}^m$

end

randomly initialize NN parameter set and get $\eta = \eta_{\text{initial}}$

train a fully-connected NN with input $\boldsymbol{X}_i$ and output $\{Z_i(t_{ij})\}_{j=1}^m$

for $e = 1$ **to** E **do**

I. *forward propagation*

(i) pass $\boldsymbol{X}_i$ through the NN and get $\{\hat{\xi}_{ik}\}_{k=1}^{K_\tau}$ for all $i \in n_{\text{train}}$

(ii) multiply $\{\hat{\xi}_{ik}\}_{k=1}^{K_\tau}$ with $\hat{\Phi}_{(K_\tau \times m)}$ to get $\{\hat{Y}(t_{ij})\}_{j=1}^m$

(iii) calculate $L(\eta) = 1/n_{\text{train}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^m (Z_i(ij) - \hat{Y}_i(t_{ij}))^2$

II. *backward propagation*

(i) update NN parameter set as $\eta^* = \eta - \varrho \frac{\partial L(\eta)}{\partial \eta}$

III. **if** $e < E$ **then** set $\eta = \eta^*$, and then repeat I. & II.

else return η^*

end

return NN with the estimated parameter set $\hat{\eta} = \eta^*$

2.2.4 Irregularly Spaced Functional Data

Earlier in this section we made the common assumption that the functional response $Y(t)$ is observed at the same m equally spaced time points. While this was a useful assumption to

make in order to explain how to train the model, it is actually not a necessary condition to fit any of the four models described above and we can train these models even with irregularly spaced functional response.

For the first two models we introduced, NNBB and NNSS, in order to train the NN, we simply need an estimate for the basis coefficients or an estimate for the FPC scores. We can rely on some existing FDA literature to get those estimates (Ramsay and Silverman, 2005; Yao et al., 2005a) for irregularly spaced functional data.

For both models that utilize the modified objective function, NNBR and NNSR, it is a bit more complicated but not so much. Once again, let us focus on NNBR to simplify the explanations. In the setting with irregularly spaced functional observations, the assumptions $m_1 = m_2 = \cdots = m_N = m$ and $t_{1j} = t_{2j} = \cdots = t_{Nj} = t_j$ no longer hold. Suppose that m_{irr} is the total number of time points with at least one observation given all training subjects and $\{t_{ij}\}_{j=1}^{m_{\text{irr}}}$ represents the union set of $\{t_{ij}\}_{j=1}^{m_i}$ for all i in the training set. The goal is to train the model using only the observations at $\{t_{ij}\}_{j=1}^{m_i}$ when the i-th subject is observed at this time. To achieve this, we need to generate a matrix Θ_{irr} of size $K_b \times m_{\text{irr}}$ with the k-th row containing entries of the k-th chosen basis function evaluated at $\{t_{ij}\}_{j=1}^{m_{\text{irr}}}$. This matrix Θ_{irr} will be employed for the calculation of $\hat{Y}_i(t_{ij})$ at all m_{irr} time points with observation(s) by taking the matrix multiplication $\text{NN}_\eta(\boldsymbol{X})\Theta_{\text{irr}}$. However, because for each subject i, $Y_i(t_{ij})$'s are originally observed at only $\{t_{ij}\}_{j=1}^{m_i}$ instead of all m_{irr} time points, when computing the objective function, we need to eliminate the contribution of those unobserved subject-time point pairs by multiplying them by 0, and accordingly

$$\begin{aligned} L_{\boldsymbol{Y}_{\text{irr}}}(\eta) &= \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{m_i} \left(Y_i(t_{ij}) - \hat{Y}_i(t_{ij})\right)^2 \\ &= \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \sum_{j=1}^{m_{\text{irr}}} \left(Y_i(t_{ij}) - \hat{Y}_i(t_{ij})\right)^2 \cdot \mathbb{1}\left(Y_i(t_{ij}) \text{ is observed}\right), \end{aligned} \tag{2.12}$$

which assures that the unobserved subject-time point pairs do not contribute to the gradient of the objective function.

Because of the common strategy used in FDA, which represents the infinite-dimensional curve using a finite basis set, we are able to construct the curve over the entire interval $\mathcal{T}$. This allows us to compute the predictive accuracy even if the test set contains observations at time points previously unseen, as long as they are within the interval of time points observed in the training set.

2.2.5 Roughness Penalty

When predicting the functional response using NNs, we need to pay attention not only to the predictive accuracy but also the smoothness of the predicted trajectories, as it is standard in FDA. In practice, we observe that the performance of the NN mapping with basis expansion

is influenced by the number of basis functions selected. Usually, the more basis functions we use to re-express the functional response, the higher accuracy can be gained when predicting the response at a set of given time points.

In FDA, it is common to set the number of basis functions to be less than the number of the observed time/location points with enough valid observations. However, in the cases where the observed time points are limited, we can consider increasing the number of basis functions to benefit the predictive accuracy. This benefit brought by more basis functions comes at the sacrifice of smoothness of the fitted functional trajectories, as they introduce in more variations. This is to be expected; because the prediction response for a specific time point $\hat{Y}_i(t)$ is the inner product of the basis coefficients and basis functions that are non-zero at t, then the more non-zero basis functions we have the more flexible that point estimate is. To control the smoothness of fitted curves without limiting the number of basis functions, we follow the common idea in FDA and propose to add some classic roughness penalty to the objective function of the NN. This simple addition ensures the smoothness of the predicted functional curves without putting any burden to the differentialable operation (since the roughness penalty terms we consider are also linearly related to the NN outputs).

There are multiple types of roughness penalties that can be applied to smooth the functional curves. In our implementation, we focus on two conventional and popular approaches: penalizing with the second derivative of the function, and penalizing directly the basis coefficients, to relieve the roughness concern with the NN-fitted functional response. Notice that we do not need to define roughness penalties for NNBB and NNSS as the smoothing is done when fitting the curve a priori.

Penalizing the Second-Order Derivative of $Y(t)$

When the smoothness of fit becomes a concern and the roughness penalty turns out to be a necessity, the most common and popular method would be penalizing the o-th order derivative of the function $Y(t)$. The squared second derivative of a function $Y(t)$ at t reveals its curvature at t, therefore it is natural to measure the roughness of the $\hat{Y}_i(t)$ by taking the integrated square of its second derivative (Ramsay and Silverman, 2005). Adding this penalty term to the objective function of the NN leads to

$$L_{\boldsymbol{pen}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left(\sum_{j=1}^{m} \left(Y_i(t_{ij}) - \hat{Y}_i(t_{ij}) \right)^2 + \lambda \int_{\mathcal{T}} \left(\frac{d^2 \hat{Y}_i(t)}{dt^2} \right)^2 dt \right). \tag{2.13}$$

The parameter λ acts as a smoothness controller for balancing the trade-off between fitting to the data and the variability of the predicted function (Ramsay and Silverman, 2005). When λ becomes larger, the more emphasis will be put on the smoothness of the fitted curves. On the contrary, for a smaller λ, the fitted curve tends to be more wiggly as there is less penalty placed on its roughness. The selection of the optimal λ can be achieved by

using cross-validation, where either the subjects or the time points are randomly divided into sub-samples (in practice, we partition the time points).

Unfortunately, we cannot back-propagate the gradient through the integral of Eq. (2.13), and thus we will approximate the integral with a summation over the domain $\mathcal{T}$. Because we are able to generate $\hat{Y}_i(t)$, we can approximate this integral with as many points as we deemed necessary

$$L_{\boldsymbol{pen}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left(\sum_{j=1}^{m} (Y_i(t_{ij}) - \hat{Y}_i(t_{ij}))^2 + \frac{\lambda T}{Q-1} \sum_{q=2}^{Q} \left(\frac{d^2 \hat{Y}_i(t_{iq})}{dt_{iq}^2} \right)^2 \right), \tag{2.14}$$

where $t_{iq} = t_q$ for all i and $\{t_q\}_{q=1}^{Q}$ are equally spaced time points covering the entire domain $\mathcal{T}$ with length T . Notice that we do not need to actually compute those second order derivatives, because

$$\begin{aligned} \hat{Y}_i(t) &= \sum_{k=1}^{K_b} \hat{c}_{ik} \theta_k(t) \\ \Rightarrow \frac{d^2 \hat{Y}_i(t)}{dt^2} &= \sum_{k=1}^{K_b} \hat{c}_{ik} \frac{d^2 \theta_k(t)}{dt^2}, \end{aligned} \tag{2.15}$$

then what we really need are the second order derivatives of the basis functions, which are much easier to compute and readily available in the **fda** package. Therefore, we can back-propagate the gradient of the objective function of Eq. (2.14) with respect to η without any problems.

Penalizing the Basis Coefficients C

Penalizing directly on the basis coefficients was firstly introduced by Eilers & Marx (Eilers and Marx, 1996) when adjacent B-splines are used to re-express the functional variable in a regression problem. Compared to penalizing the derivative, this idea reduces the dimensionality of the smoothing problem to K_b, the number of basis functions, instead of N, the number of observations (Eilers and Marx, 1996).

The basic structure of this penalty is the difference of a set of consecutive basis coefficients $\Delta^2 c_k = c_k - 2c_{k-1} + c_{k-2}$. This difference has a strong connection with the second derivative of the fitted function, which is revealed by the simple formula for derivatives of B-splines given by De Boor (de Boor, 1978). The summation of the squared differences for all three consecutive coefficients $\{c_k, c_{k-1}, c_{k-2}\}, k = 3, ..., K_b$ would be the main component of the penalty term, with the strength of the penalty further controlled by a tuning parameter λ.

Like previously, the penalty term is added to the objective function of NN as:

$$L_{\boldsymbol{pen}}(\eta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left(\sum_{j=1}^{m} \left(Y_i(t_{ij}) - \hat{Y}_i(t_{ij}) \right)^2 + \lambda \sum_{k=3}^{K_b} (\Delta^2 c_{ik})^2 \right). \tag{2.16}$$

Similarly, the optimal hyperparameter λ is selected using k-fold cross-validation.

Empirically, when fitting one of the B-spline models, especially the NNBR model, including a large number of basis functions, more than m for example, leads to a better predictive accuracy, but also results in a set of very rough predicted curves. When $K_b > m$ the predicted curves quickly become *wiggly*, and thus we would increase the number of basis such that $K_b >> m$ and impose a roughness penalty at the same time to keep the curves smooth. In some cases, this leads to the top performer in terms of predictive accuracy. Unfortunately, using $K_b >> m$ and a roughness penalty could make the complex hyperparameter tuning process even more difficult. Nonetheless, we believe it is important to provide roughness penalties and justifications for those when providing a technique to fit functional data.

We implemented the four models and the two roughness penalties in `R` (R Core Team, 2019). The functional component of our implementation relies on the **fda** package and the NN component uses the `R` implementation of **Keras**. Our implementations of those models along the real data example are available online on the second author's GitHub page.

2.3 Computational Complexity

An advantage of using an NN as the link function instead of a linear combination is that it might have a lower computational cost. More specifically, when using a standard training procedure for both models, the NN approach scales better with the number of predictors. Indeed, we usually find an exact solution for the FoS model which involves the inversion of a matrix, resulting in worse scaling with respect to the number of predictors. On the flip side, the algorithm used to train NN scales linearly with the number of predictors.

Start by looking at the FoS model, we rely on the formulation established in Ramsay & Silverman (Ramsay and Silverman, 2005), chapter 14. This book contains various information about the computational details of the solutions of the FoS optimization problem under various parameterizations. However, its simplest form is quite similar to the least square solution of a traditional regression where:

$$\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y, \tag{2.17}$$

which involves the inversion of a $P \times P$ matrix that requires a number of operations proportional to P^3, say $O(P^3)$ in big O notation, using Gaussian elimination. The Strassen algorithm (Strassen et al., 1969) manages to get that order down to $O(P^{2.8})$ and the more

recent Coppersmith-Winograd algorithm (Coppersmith and Winograd, 1987) gets the order of matrix multiplication down to $O(P^{2.37})$ for a matrix of dimension $P \times P$.

The **fda** package uses the `solve()` function included in the default `R` language which in turn employs the linear algebra package (LAPACK) (Anderson et al., 1999) that relies one Gaussian elimination to inverse matrices. Other than the matrix inversion, the matrix multiplication component of the solution involves a $P \times P$ matrix, a $P \times N$ matrix and a $N \times m$ matrix resulting in a number of operations of order $O(PN)$. Including the matrix multiplication part of the solution the whole FoS estimation requires a number of operations of order $O(P^3N)$. The more complete solutions that consider the smoothness of the predicted curves involve the inversion of a matrix of dimension $K_pP \times K_pP$ where K_p is the number of basis of the functional parameters $\boldsymbol{\beta}(t)$. It means scaling polynomial with power 3 with the number of predictors P when using Gaussian elimination. Hence, we consider that the number of operations needed to obtain the exact solution to the FoS problem requires a number of operations of order $O(P^3N)$.

As far as NNs are concerned, according to Hastie & al. (Hastie et al., 2009), the computational cost of training an NN is of order $O(PNn_wE)$ when backpropagation is adopted (Rumelhart et al., 1986) to estimate the gradient and when a gradient-based approach is applied to fit the NN. In the formulation above, N is the number of observations, P is the number of predictors, n_w is the number of weights (hidden neurons) in the NN and E the number of training epochs.

Therefore, if we strictly focus on how the run time of an NN scales with N and P, we are looking at a linear scaling in both cases, $O(PN)$. Consequently, when comparing computational costs, the NN model has a pretty significant advantage when it comes to its scaling with respect to the number of predictors. Because of this better scaling with respect to the number of predictors, we can claim that our proposed models are better equipped than the FoS regression to deal with data sets containing lots of predictors.

However, let us be a little more nuanced. It is clear that under some circumstances we can fit extremely large NNs which could lead to a slow fitting process. Additionally, the main difference between the traditional ways to fit these models is that we pursue an exact solution for the linear problem and on the contrary we utilize a gradient-based approach to find a solution in the case of NNs. As a result, for a linear model, such as FoS, we do have worse scaling with respect to P but we have guarantees that the optimal solution is reached. In theory, it would be possible to fit a linear model, such as FoS, with a gradient-based approach to improve how its computational cost scales with respect to P, though this is certainly not the standard optimization approach.

2.4 Real Data Application

We evaluate the predictive performance of the proposed methods, along with the conventional FoS model and the FAM model on the ASFR data set introduced in Section 2.1. We consider to compare the proposed models with the FAM model, $Y_i(t) = \sum_{p=1}^{P} f_p\left(X_{ip}, t\right) + \epsilon_i(t)$, because our methods can easily and naturally account for the nonlinear effects of all possible interactions simultaneously, while FAM is predominantly focused on learning the nonlinear relation between the functional response and each of the scalar predictors individually (it is computationally difficult and expensive for FAM to learn all possible interactions from a multi-dimensional aspect). For each of the models, we proceed with 20 repetitions of random subsampling validation: randomly dividing the data set into a training set and a test set, with 80% and 20% of the total samples assigned to them, respectively.

Following the same tuning strategy usually applied in classic NNs, the hyperparameters of all models (except FAM) in comparison are firstly tuned using 5-fold cross-validation in order to fairly improve their performances during actual training. We start the hyperparameter tuning with FoS, considering it involves only one hyperparameter K_b, the number of basis functions. To reduce the computational time consumed by the tuning processes of the NN-based models, the optimal basis system selected for the FoS model is then used for the rest of the models. Afterwards, for each NN-based model, we perform a grid search on all network hyperparameters simultaneously by taking a list of possible values for each of the hyperparameters and running a 5-fold cross-validation for all combinations. An NN consisting of 2 hidden layers with 50 and 30 neurons, respectively, is selected. 99% proportion of variance explained is recommended by cross-validation as the threshold to truncate the FPCs scores in NNSS and NNSR. For NNBR with penalized objective function (NNBR(P)), we apply 10 basis functions and the second-order derivative roughness penalty, together with smoothing parameter $\lambda = 10^{-7}$, which is similarly suggested by the 5-fold cross-validation from a set of possible values $\{10^{-i}\}_{i=1}^{8}$. Considering the number of basis functions is not dramatically large, the fitted curves are already quite smooth without any roughness penalty. Thus, a relatively small λ is acceptable as more emphasis ought to be placed on the fitting to data for an overall smallest $L_{\boldsymbol{pen}}(\eta)$. A summary of the optimal hyperparameters determined for each model is provided in Table A.2 in Appendix A. This hyperparameter tuning approach is also applied in the simulations of Section 2.5. The FoS model used in this experiment is coming from the `R`-package **fda**, and FAM model is trained with the help of function `pffr()` in the **refund** package.

Table 2.1: Mean squared errors of prediction (MSEPs) of 20 random test sets for various models with ASFR data set.

Methods	Mean	Std. Dev.	p-value of t-test
FoS	1177.98	513.58	-
NNBB	1031.87	294.87	0.18
NNSS	**992.50**	296.68	**0.01**
NNBR	1060.50	210.63	0.20
NNBR(P)	1059.07	**208.27**	0.19
NNSR	1047.46	277.37	0.11
FAM	1317.33	919.62	0.28

Table 2.1 shows the predictive performance of each model using the mean squared error of prediction (MSEP) over the 20 replications. The predictive accuracy is measured by the MSEP averaged across the number of samples and the number of observed time points in the test set. A two-sided paired t-test is later conducted to compare the MSEPs of the 20 replicates of our models to the 20 MSEPs of the FoS model. It is clearly shown that our proposed models consistently outperform FoS and FAM models in predicting the fertility curves, with both smaller means and standard deviations (SDs) of the prediction errors of 20 random test sets, demonstrating their advantages in capturing both linear and nonlinear effects of all predictors on the functional response. NNSS model, as indicated in Table 2.1, has the best predictive performance, which interestingly implies that the FPC scores are more informative than the basis coefficients in representing $Y(t)$ with this data set, and consequently the effects of covariates on functional response are better described by the relations between the predictors and FPC scores instead of basis coefficients. In addition, the prediction error of NNBR optimized by the penalized objective function is lower than that of the non-penalized NNBR, indicating that including a relatively large number of basis functions, along with a roughness penalty term added to the objective function, can help improve the prediction without loss on the smoothness of the predicted curves.

As previously mentioned in Section 2.1, several covariates, such as female age and GDP per capita, are seemingly not linearly related to many of the basis coefficients representing the fertility trajectory. Therefore we focus on the basis coefficients $\hat{\boldsymbol{C}}_{\text{FoS}}$, $\hat{\boldsymbol{C}}_{\text{NNBB}}$ and $\hat{\boldsymbol{C}}_{\text{MMBR}}$ predicted by FoS, NNBB and NNBR (they are the models using basis coefficients for regression), respectively. The relations between the predictor female age and each of the obtained second basis coefficients, including $Y(t)$-estimated second basis coefficient $\hat{c}_{2,Y(t)}$ and the model-predicted ones, are displayed in Figure 2.3. We can see that the nonlinear pattern for female age and the second basis coefficient is better recovered by NNBB and NNBR, while the FoS-predicted basis coefficient $\hat{c}_{2,\text{FoS}}$ is more likely linearly related to female age. Likewise, we select under-5 mortality out of those covariates that have highly likely

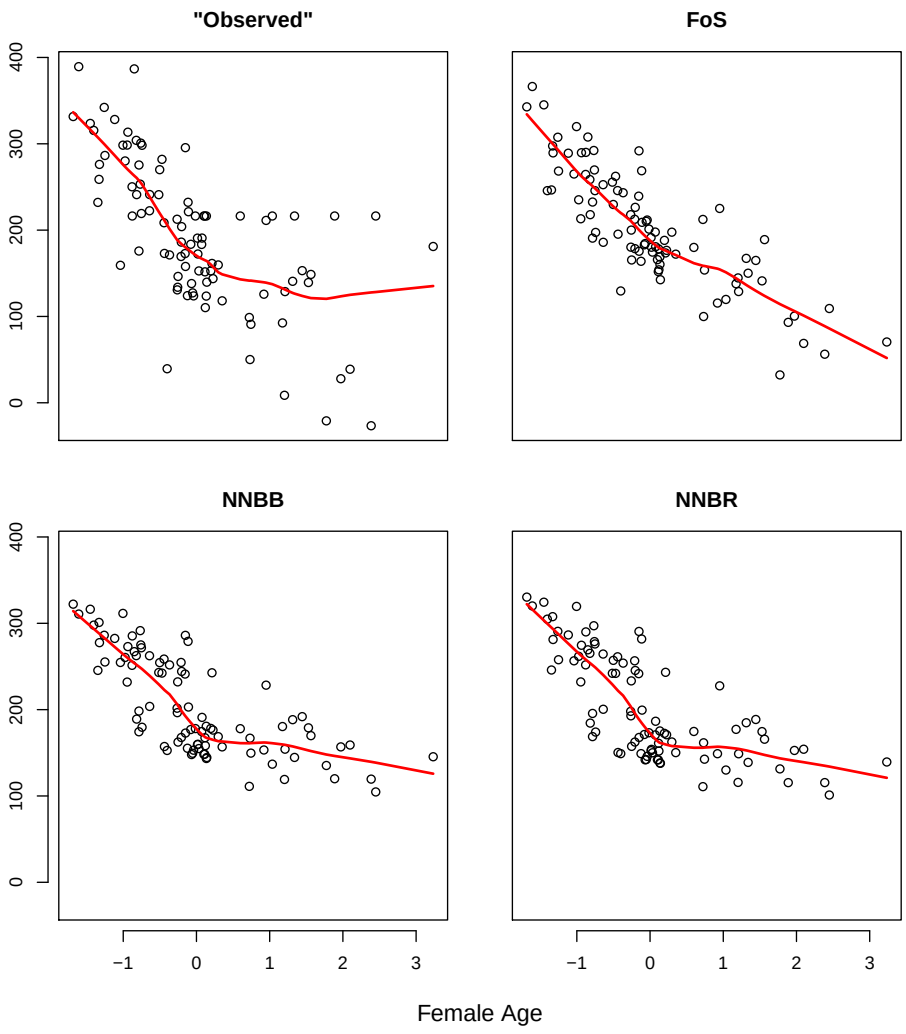

Figure 2.3: The relations between the covariate female age and the $Y(t)$-estimated second basis coefficient ($\hat{c}_{2,Y(t)}$), FoS-predicted second basis coefficient ($\hat{c}_{2,\mathrm{FoS}}$), NNBB-predicted second basis coefficient ($\hat{c}_{2,\mathrm{NNBB}}$) and NNBR-predicted second basis coefficient ($\hat{c}_{2,\mathrm{NNBR}}$), respectively.

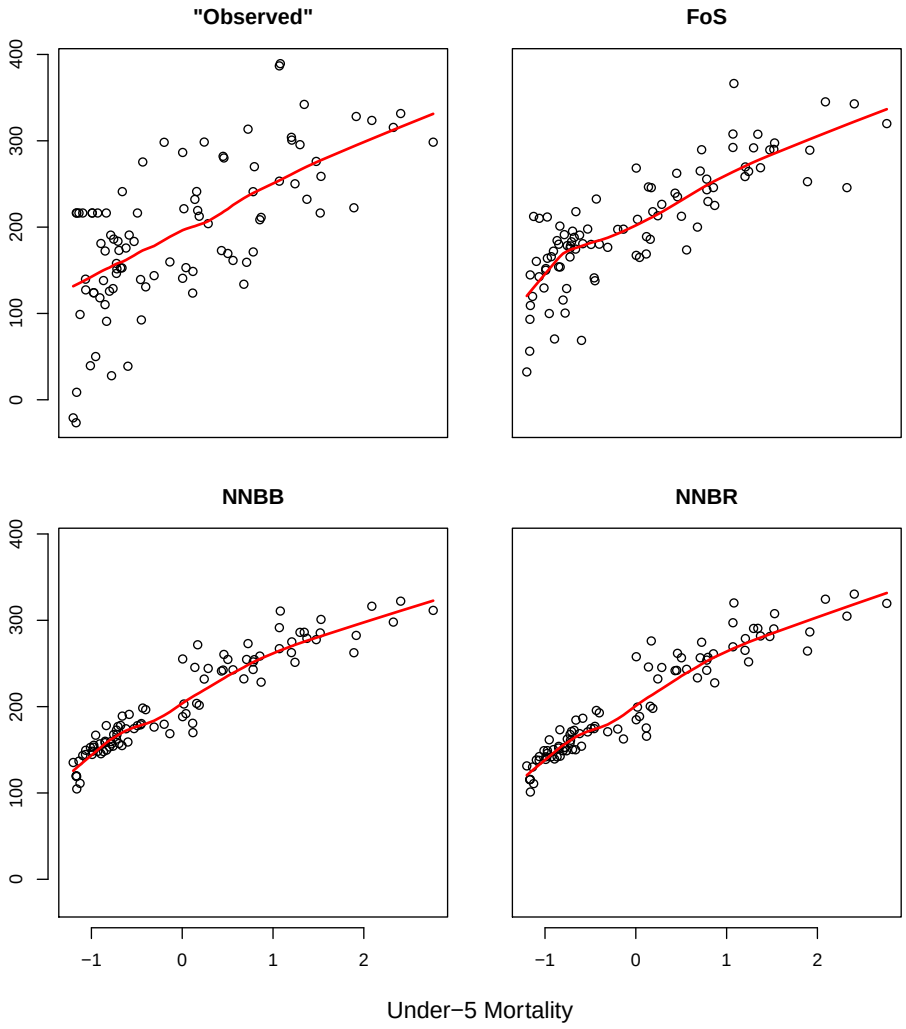

Figure 2.4: The relations between the covariate Under-5 Mortality and the $Y(t)$-estimated second basis coefficient ($\hat{c}_{2,Y(t)}$), FoS-predicted second basis coefficient ($\hat{c}_{2,\mathrm{FoS}}$), NNBB-predicted second basis coefficient ($\hat{c}_{2,\mathrm{NNBB}}$) and NNBR-predicted secondbasis coefficient ($\hat{c}_{2,\mathrm{NNBR}}$), respectively.

linear associations with some of the basis coefficients, and Figure 2.4 reveals how under-5 mortality relates to different model-predict second basis coefficient $\hat{c}_2$. It is noticed there are plainly linear trends between under-5 mortality and the second basis coefficients predicted by the three models mentioned, while under-5 mortality has been shown linearly related to the second basis coefficient estimated by $Y(t)$. It is not surprising that FoS reconstructs the most similar pattern to the one built on the $Y(t)$-estimated basis coefficient, but the proposed models also master the linear information when recovering the associations, which proves that the NN-based models have the ability to take good care of both the nonlinear and linear relations between the predictors and scalar representations simultaneously. Due to the relatively small number of observations, it is worth mentioning that the relations between covariates and basis coefficients may not be fully revealed and visual misunderstanding could occur.

2.5 Simulation Studies

2.5.1 Generating Data

We are interested in data sets that have multiple predictors and a potentially nonlinear relation between the scalar predictors and the functional response. We were inspired by the variable selection literature to design our data generators. More precisely, our data generators are similar to those proposed by Wang & al. (Wang et al., 2007) and Barber & al. (Barber et al., 2017). The basic concept is to build the functional response using a linear combination of a set of K random curves $\psi_k(t)$ and K associated coefficients $\zeta_k(\boldsymbol{X})$:

$$Y(t) = \sum_{k=1}^{K} \zeta_k(\boldsymbol{X})\psi_k(t). \tag{2.18}$$

and we are going to explore multiple approaches to generate the random curves $\psi_k(t)$'s and multiple ways to use the predictors $\boldsymbol{X}$ to build coefficients $\zeta_k(\boldsymbol{X})$'s.

Firstly, we concentrate on how to generate the curves $\psi_k(t)$'s. Wang & al. (Wang et al., 2007) employ spline functions constructed with B-splines. Due to the requirement of their simulation, they use rather simple order 4 B-splines with 1 interior knot corresponding to 5 basis functions. To create a simple scenario for visualization purpose, we consider a configuration where a random curve $\psi_k(t)$ is set to be a single B-spline basis function $B_k(t)$:

$$\psi_k(t) = B_k(t). \tag{2.19}$$

Here, the applied B-spline basis system is of order of 4, while the number of interior knots are calculated using the number of predictors (number of predictors -4). The total number of basis functions, in this scenario, is equivalent to the number of scalar covariates. The main purpose of this configuration is to visualize if models are able to recover the coefficients

$\zeta_k(\boldsymbol{X})$'s. Because in this example, a single covariate only affect a single B-spline basis function, we can fix $\zeta_k(\boldsymbol{X})$ to be a nonlinear function of the coefficients $\boldsymbol{X}$ and to see if the trained model captured that nonlinear effect. We notably use that configuration for our **Design 1**.

The second configuration is a somewhat more realistic case where the covariates affect the response over the entire time interval. In this configuration, each random curve $\psi_k(t)$ will be built using B-splines of order 4 with 9 interiors knots corresponding to 13 basis function $\{B_l(t)\}_{l=1}^{13}$. The 13 associated coefficients β_l's are randomly generated from a normal distribution. Thus, for the B-splines configuration we have:

$$\psi_k(t) = \sum_{l=1}^{13} \beta_{k,l} B_l(t). \tag{2.20}$$

This configuration is used in our **Design 2, 3 & 4**.

In terms of the coefficients $\zeta_k(\boldsymbol{X})$'s we will look at three configurations. Because NNs are known to capture nonlinear relationships, it is important to design nonlinear functions $\zeta_k(\cdot)$'s. First, for visualization, we apply a polynomial function to the continuous predictors. Specifically, a subset of continuous covariates, denoted by $\boldsymbol{X}_{\text{poly}}$, is randomly selected and each coviariate in $\boldsymbol{X}_{\text{poly}}$ is further transformed by a polynomial function with either second or third degree (half of the selected covariates are processed by quadratic functions and the rest by cubic functions), and accordingly $\zeta_k(\boldsymbol{X}) = \text{polynomial}(X_k)$ for $X_k \in \boldsymbol{X}_{\text{poly}}$, otherwise $\zeta_k(\boldsymbol{X}) = X_k$. This configuration is used in **Design 1 & 2** with a visualization example displayed in **Design 1**. Our second configuration is also nonlinear but this time it is a bit more complex. We define a 3-hidden-layer NN with the sigmoid (which is nonlinear) activation functions and random weights. For this configuration, $\zeta_k(\boldsymbol{X})$ is the k-th output of an NN taking $\boldsymbol{X}$ as its input. These coefficients are used for **Design 3**. Finally, we consider the case where $Y(t)$ is a linear combination of curves with $\boldsymbol{X}$ being the coefficients directly. In other words, the k-th coefficient is simply the k-th predictor: $\zeta_k(\boldsymbol{X}) = X_k$. This is a scenario where we expect the FoS to outperform the models we propose, but we want to visit this example regardless. We call this the linear configuration and use it for **Design 4**.

Finally, we add a random noise, which is normally distributed with zero mean and a variance of 2 to every data point $Y_i(t_{ij})$.

2.5.2 Results

Four different simulation designs are considered to illustrate the advantages of the proposed methods. For each design, we generated data sets of 2000 observations and randomly sampled 1800 training observations and 200 testing observations. This random subsampling procedure was repeated 20 times. We opted to experiment with 20 scalar predictors, different random curve configurations and different relations between the coefficients and predictors, either

linear or nonlinear, all with some random noise. To mimic the real-world scenario, the actual observations for $Y(t)$ were simulated discretely at 40 equally spaced time points $\{t_j\}_{j=1}^{40}$ that entirely cover $\mathcal{T} = [0, 1]$. Similar to the real application, we compare the proposed NN-based methods, including NNBB, NNSS, NNBR and NNSR, to the novel FoS model (we exclude FAM model because its training cost is tremendously high and based on the results of the initial replicates, it performs poorly compared to the other models). The predictive accuracy was evaluated using the MSEP on the test set. We also report the p-value of the two-sided paired t-test of the MSEPs of each of our methods to that of the FoS.

Design 1 (nonlinear scenario): We generate 20 random predictors ($K = 20$), where X_k's are i.i.d. uniform random variables from $[a, b]$ with $a \in \{-4, -3, -2, -1, 0\}, b \in \{3, 4, 5, 6, 7\}$ for all k. We apply the polynomial transformation to 50% of the continuous variables by setting $\zeta_k(\boldsymbol{X}) = \text{polynomical}(X_k)$ with degree of 2 and 3 for $k = 8, 10, 12, 13, 14$ and $k = 1, 3, 4, 7, 9$, respectively, and $\zeta_k(\boldsymbol{X}) = X_k$ otherwise. The random curves $\psi_k(t)$'s, in this case, are set to be the B-spline basis functions as $\psi_k(t) = B_k(t)$ for all k, and accordingly the functional response is simulated as $Y(t) = \sum_{k=1}^{20} \zeta_k(\boldsymbol{X}) B_k(t)$.

Table 2.2: Mean squared errors of prediction (MSEPs) of 20 random test sets for various models with data generated by **Design 1**.

Methods	FoS	NNBB	NNSS	NNBR	NNSR
Mean	49.20	5.14	5.98	4.95	5.91
Std. Dev.	1.64	4.11	0.23	0.11	0.19
p-value	-	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16

Table 2.2 summarizes the means and SDs of the MSEPs on the same testing observations achieved by different models in 20 replicates for **Design 1**. In the nonlinear scenario, we observe that the predictive performances of all the NN-based models surpass that of the FoS Model under 1% significance level. Two proposed models using basis coefficients as the output of NN exhibit the best-level performances in predicting the response curve, with smaller means of MSEPs compared to those of the NN-based models outputting FPC scores. This is expected because the functional response in this design is generated as a directly linear combination of the polynomial-transformed predictors and B-spline basis functions, and given this special setting, we are able to visualize the relations as some describable patterns for some selected coefficient-predictor pairs. This design highlights a situation where using FoS would be extremely problematic and where any of our proposed NN models would provide a significant improvement.

Next, we compare NNBB, NNBR with FoS for their abilities in recovering the underlying relationships for different coefficient-predictor pairs. The relations between a predictor and basis coefficients achieved by the aforementioned models are visualized through the scatter

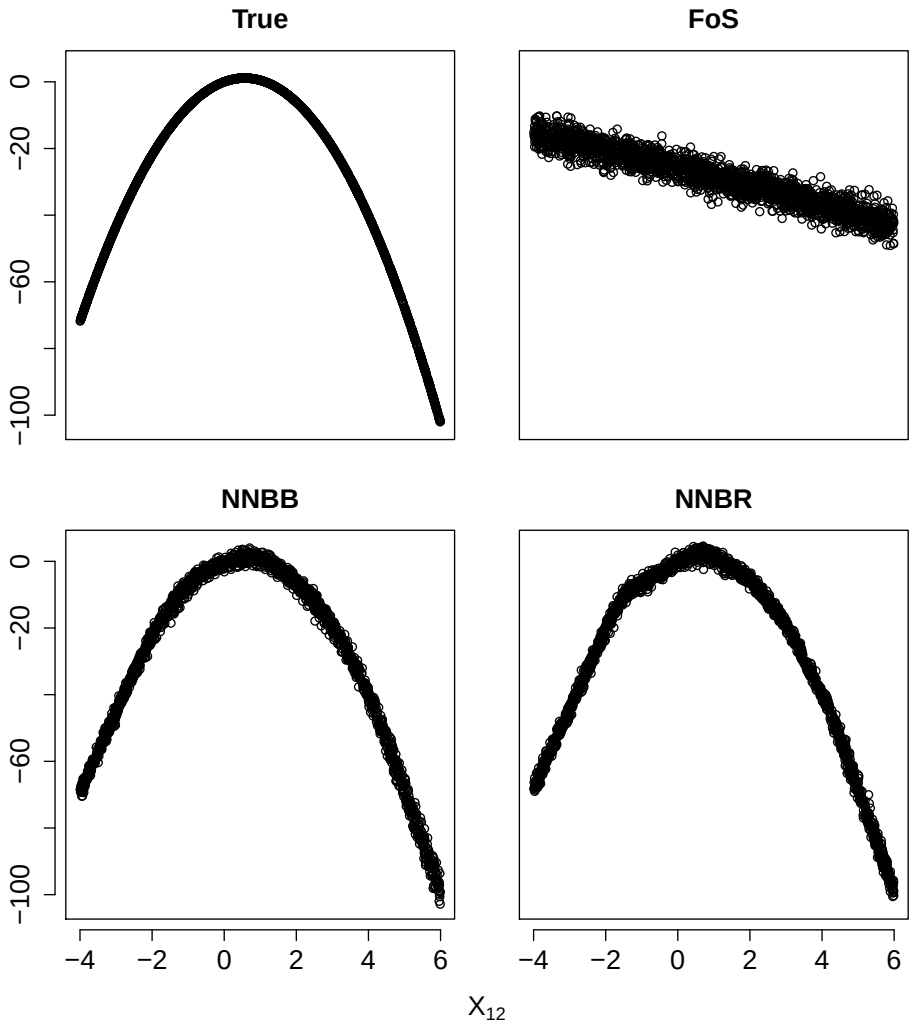

Figure 2.5: Scatter plots of the true c_{12} (ζ_{12} as per the generator), FoS-predicted $\hat{c}_{12,\text{FoS}}$, NNBB-predicted $\hat{c}_{12,\text{NNBB}}$, and NNBR-predicted $\hat{c}_{12,\text{NNBR}}$ against X_{12} in **Design 1**, from left to right respectively.

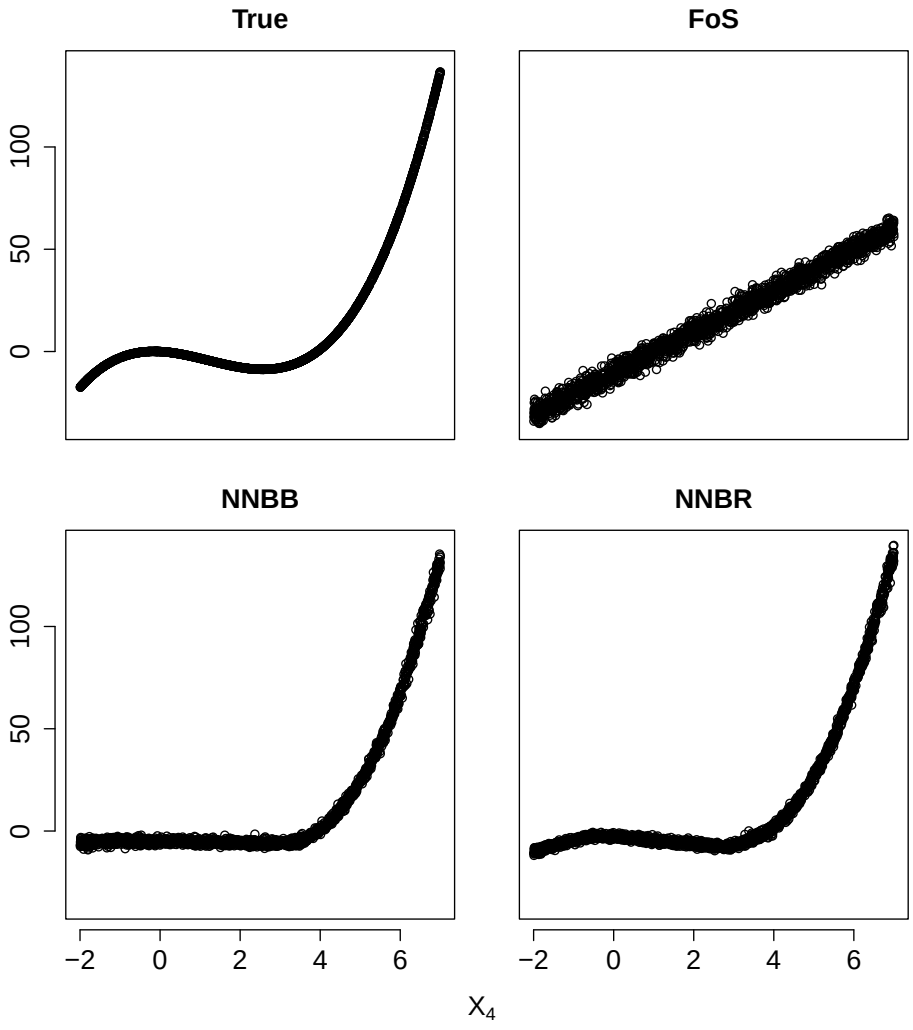

Figure 2.6: Scatter plots of the true c_4 (ζ_4 as per the generator), FoS-predicted $\hat{c}_{4,\text{FoS}}$, NNBB-predicted $\hat{c}_{4,\text{NNBB}}$, and NNBR-predicted $\hat{c}_{4,\text{NNBR}}$ against X_4 in **Design 1**, from left to right respectively.

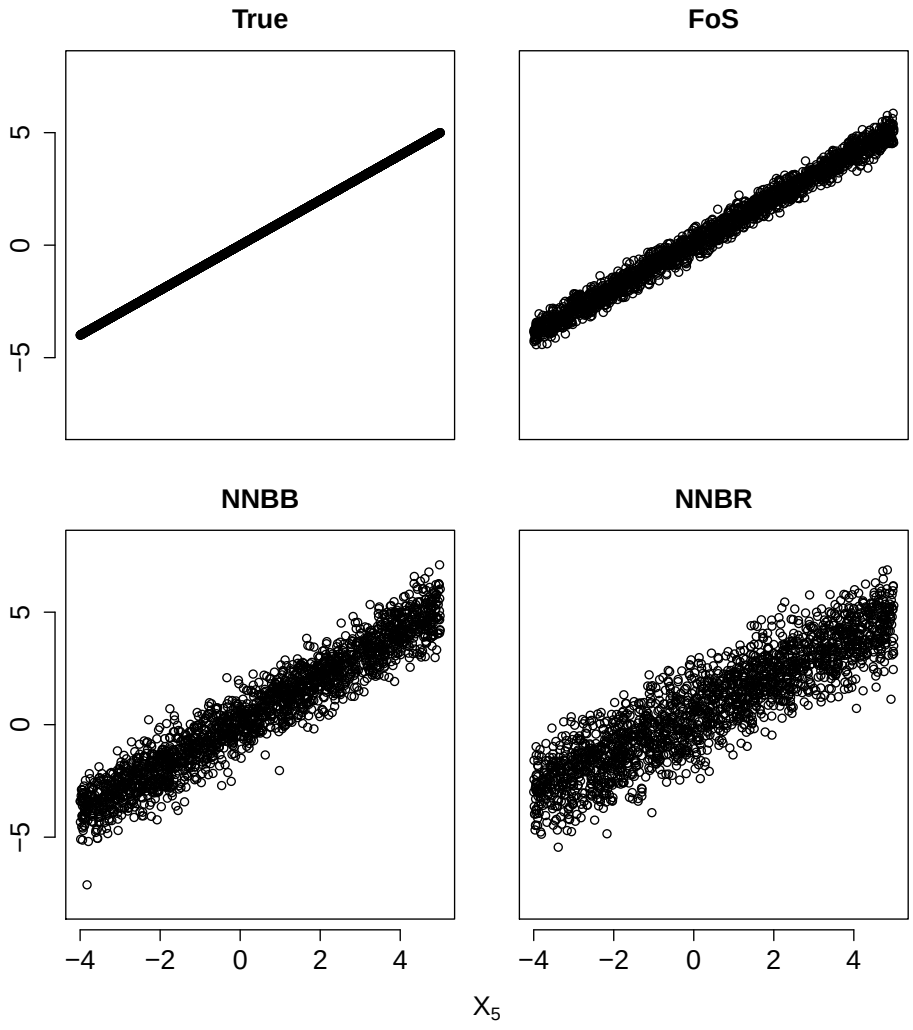

Figure 2.7: Scatter plots of the true c_5 (ζ_5 as per the generator), FoS-predicted $\hat{c}_{5,\text{FoS}}$, NNBB-predicted $\hat{c}_{5,\text{NNBB}}$, and NNBR-predicted $\hat{c}_{5,\text{NNBR}}$ against X_5 in **Design 1**, from left to right respectively.

plots of each of the basis coefficients against the predictor. Remember that we designed this simulation specifically to visualize how NN models learn $\boldsymbol{C} = \mathrm{NN}_\eta(\boldsymbol{X})$ and to compare it with the $\zeta_k(\boldsymbol{X})$ we designed.

We randomly select X_{12}, one of the polynomial-transformed covariates, and c_{12}, the basis coefficient corresponding to the 12-th B-spline basis function, and show their true relation, as $\zeta_{12}(\cdot)$, together with the relations reconstructed by the three mentioned approaches using scatter plots of true c_{12} and $\hat{c}_{12}$ predicted by each of the models against X_{12} in Figure 2.5. Undoubtedly, NNBB and NNBR both precisely capture the $\cap$-shape between the true c_{12} and X_{12}, while FoS replaces the nonlinear trend with a downwards linear pattern. Being interested in a more complex truth, we also select X_4, one of the covariates transformed with a cubic polynomial, coupled with c_4, the basis coefficient corresponding to the 4-th B-spline, and provide the scatter plots illustrating the true and model-reconstructed relations between X_4 and c_4 in Figure 2.6 . As expected, NNBB and NNBR perform well in overall recovering the nonlinear shape while NNBR behaves more accurately in capturing the local curvature occurring in the left component of the true pattern. It is not surprising that FoS again learns a linear relation because it is designed to only learn such relations. Likewise, the associations between X_5 and c_5's by multiple models are revealed in Figure 2.7, but differently, the true relation for this X_5-c_5 pair is linear. We can observe, NNBB and NNBR perform similarly as FoS in retrieving the linear pattern but with comparably thicker bandwidths, indicating that our models can also successfully detect the linear relation for some coefficient-predictor pairs but with higher variance. Figure 2.5, 2.6 and 2.7 highlight the utility of our methods in recovering the true underlying relations, especially the nonlinear relations between the basis coefficients and the predictors, and the success in learning the true relationships contributes to their fabulous performances on predicting the response trajectories. The ability to recover the true underlying nonlinear relations is the main benefit of our proposed approaches.

Design 2 (nonlinear scenario): 20 random predictors ($K = 20$) are generated, with X_k's being binary variables for $k = 1, 3, 5, 7$, eight-level categorical variables for $k = 2, 4, 6$, and i.i.d. uniform random variables from $[a, b]$ with $a \in \{-4, -3, -2, -1, 0\}, b \in \{3, 4, 5, 6, 7\}$ for $k \geq 8$. Likewise, approximate 50% of the continuous variables are later transformed by polynomial functions with different degrees, where $\zeta_k(X) = \text{polynomical}(X_k)$ with the second and the third degree for $k = 14, 16, 17, 18$ and $k = 8, 10, 19, 20$, separately, and $\zeta_k(X) = X_k$ for the remainder. Then we construct the response curve as $Y(t) = \sum_{k=1}^{20} \zeta_k(\boldsymbol{X})\psi_k(t)$, coupled with $\psi_k(t) = \sum_{l=1}^{13} \beta_{k,l} B_l(t)$, where $\{B_l(t)\}_{l=1}^{13}$ are the B-spline basis functions with order 4 and $\beta_{k,l}$ are i.i.d. random variables following the normal distribution $\mathcal{N}(0, 4)$.

Table 2.3: Mean squared errors of prediction (MSEPs) of 20 random test sets for various models with data generated by **Design 2**.

Methods	FoS	NNBB	NNSS	NNBR	NNSR
Mean	4559.20	38.33	286.23	36.38	265.50
Std. Dev.	159.01	9.18	79.68	18.76	16.05
p-value	-	<2.2e-16	<2.2e-16	<2.2e-16	<2.2e-16

The means and SDs of the MSEPs on the testing observations for all methods in comparison can be found in Table 2.3. We observe that all our proposed methods remain superior to the FoS method in this setting. The NN-based models with basis-coefficient output continue to be the top-tier performers, especially the NNBR model trained using the response variable directly. The performance of the FoS is still significantly lower than the NN models and still has the highest variance. For both of these designs, some coefficient functions ζ_k's have significant nonlinear curvature being polynomial functions of degree 2 or 3. Consequently, the dominance of the NN models is expected, even now with a more realistic scenario where a predictor affects the response on its entire domain $\mathcal{T}$.

Design 3 (nonlinear scenario): We continue to set $K = 20$, and X_k's are generated as binary variables, eight-level categorical variables and i.i.d. uniform random variables from $[a, b]$ with $a \in \{-4, -3, -2, -1, 0\}, b \in \{3, 4, 5, 6, 7\}$ for different sets of k (same as **Design 2**). We trigger the nonlinear configuration by passing the predictors $\boldsymbol{X}$ through the 3-hidden-layer NN introduced in Section 2.5.1, resulting in $\zeta(\boldsymbol{X}) = \mathrm{NN}(\boldsymbol{X})$. Same as **Design 2**, we choose to use the random curves $\psi_k(t) = \sum_{l=1}^{13} \beta_{k,l} B_l(t)$, where $\{B_l(t)\}_{l=1}^{13}$ are the B-spline basis functions with order 4 and $\beta_{k,l} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 4)$. In consequence, the functional response is generated following $Y(t) = \sum_{k=1}^{20} \mathrm{NN}(\boldsymbol{X})\psi_k(t)$.

Table 2.4: Mean squared errors of prediction (MSEPs) of 20 random test sets for various models with data generated by **Design 3**.

Methods	FoS	NNBB	NNSS	NNBR	NNSR
Mean	4.17	4.18	4.16	4.13	4.17
Std. Dev.	0.02	0.03	0.02	0.03	0.02
p-value	-	1.1e-02	6.2e-02	2.7e-08	1.8e-01

Table 2.4 reports the means and SDs of MSEPs on the test sets for all models with data generated by **Design 3**. It is shown that most models perform similarly on predicting the functional curve. The $\zeta_k(\boldsymbol{X})$'s produced by the NN generator are barely nonlinear, having only small curvature with respect to the predictors. This leads to a reduction in the

performance gap between our NN models and the FoS model. However, the NNBR model comes ahead once again, being significantly superior to the FoS model.

Design 4 (linear scenario): In the last design, we consider setting up a scenario where the functional response $Y(t)$ is linearly associated to the scalar predictors. 20 predictors are generated in the manner that $\{X_k\}_{k=1}^{20}$ are binary variables, eight-level categorical variables and i.i.d. uniform random variables from $[a, b]$ with $a \in \{-4, -3, -2, -1, 0\}, b \in \{3, 4, 5, 6, 7\}$ for different subsets of $k = 1, 2, ..., 20$ (same as **Design 2 & 3**). The linear setting is simply achieved with $\zeta_k(\mathbf{X}) = X_k$. We continue with the random curves $\psi_k(t) = \sum_{l=1}^{13} \beta_{k,l} B_l(t)$, where $\{B_l(t)\}_{l=1}^{13}$ are the order 4 B-spline basis functions, together with $\beta_{k,l} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 4)$. The functional response is generated as a linear combination of the random curves and predictors, following $Y(t) = \sum_{k=1}^{20} X_k \psi_k(t)$.

Table 2.5: Mean squared errors of prediction (MSEPs) of 20 random test sets for various models with data generated by **Design 4**.

Methods	FoS	NNBB	NNSS	NNBR	NNSR
Mean	4.01	4.19	6.66	4.07	5.94
Std. Dev.	0.05	0.06	0.15	0.05	0.05
p-value	-	3.2e-08	4.3e-13	7.8e-07	2.2e-09

Table 2.5 displays the results of all models under the linear setting described. In the linear scenario, we expect FoS to be the top performer and indeed, it has the lowest prediction error compared to all the proposed methods. This is reasonable as the $Y(t)$ is constructed linearly with respect to the predictors in this design. However, despite the proposed approaches being significantly worse, NNBB and NNBR remain competitive with strong performances closely following the one of FoS. From an absolute perspective, the gap between NNBR and FoS is much smaller in the scenario favouring FoS (**Design 4**), with FoS being marginally ahead, than it is in the scenario favouring NNBR (**Design 1 & 2**), with NNBR being dramatically ahead.

2.6 Conclusions and Discussion

In the chapter, we introduced a new solution for the regression of a functional response on scalar predictors, which consistently outperforms the current models when the relation between the functional response and the scale predictors is nonlinear. We designed and manipulated the standard feed-forward NN to produce two types of output, either basis coefficients or FPC scores, both of them being the traditional techniques to analyse functional data. The proposed modification to the objective function allows us to train the model directly with the functional response variable thus bypassing the necessity to firstly estimate those coefficients using conventional FDA approaches. The modified objective function

enables the transformation of a scalar layer to a functional output by constructing functional curves using a linear operation, which ensures the applicability of backpropagation. In this way, an NN with the proposed functional output layer behaves similarly as a regular NN and allows the usage of typical cross-validation techniques for hyperparameter tuning. Such modifications can be directly implemented to the output layer of any existing NN to produce a functional response, promoting the utilization of various deep learning techniques in a functional regression framework. We implemented all of our proposed models in a way they can be trained on both regularly and irregularly spaced domain. Additionally, we provided the necessary tools to control the smoothness of the predicted response curves by implementing two different roughness penalties. Furthermore, our family of models scales better with the number of predictors.

Based on a real data application, we demonstrated a case where the models we propose are superior to already established techniques such as FoS and FAM. Moreoever, through several simulation studies, we not only showed the superior predictive power of our NN-based approaches, but also their strong ability to recover nonlinear relation between the predictors and the coefficients representing the response curves.

On the other hand, the developed methods rely on a large number of hyperparameters, including the number of hidden layers, the number of neurons in each of the hidden layers, the number of basis functions (NN output size), the number of training epochs, etc. This is the main weakness of our approach since conducting a grid search on that space is particularly annoying and time-consuming. For both B-spline expansion models, the performance varied dramatically from one hyperparameter configuration to another and this is certainly what we need to bring up. Comparatively, the FPCA-based models varied only slightly while changing the number of principal components. Given once we have the first few leading principal components, we can capture the vast majority of the variability between curves. The performances were also much more stable across various parameterizations, though those models were rarely top performers. In contrast, the FoS model is the easiest to fit among all the models applied, albeit it performed poorly in nonlinear situations.

Multiple directions of further research are considered at the moment. Our proposed models can be extended to predict multidimensional (mainly two-dimensional) functional response where $t \in \mathbb{R}^q, q > 1$. In this scenario, we can borrow the basis expansion or the FPCA technique to compress the multidimensional functional data to a vector of finite scalar basis coefficients or FPC scores (Ivanescu, 2013; Hayes and Halliday, 1974; Dierckx, 1984; Zhou and Pan, 2014; Chen and Jiang, 2016). Additionally, our current models rely on the existing NN architecture with a scalar output layer, which requires us to firstly project the functional response to some finite-dimensional coefficients and then feed the NN with the scalar representations obtained. Extending the NN to a more dynamic architecture allowing a functional output layer could be a more appropriate tool for such type of regression problems. This could be achieved by defining functional neurons and functional layers, some concepts

we are currently exploring. Furthermore, a combination of our proposed NN models with existing literature that tackles the SoF problem can be explored to create a general and complete framework for using NN to analyze and solve regression problems for various forms of functional data.

Chapter 3

Functional Autoencoder for Smoothing and Representation Learning

Existing techniques for data smoothing and dimensional reduction in functional data analysis (FDA) primarily focus on linear representations of functional data, however, addressing nonlinear mappings from the data space to the representation space is more effective in applications. In this chapter, we propose to learn the nonlinear representations of functional data using neural network autoencoders (AEs) designed to process data in the form it is usually collected without the need of preprocessing. We design the encoder to employ a projection layer that computes the inner product of the functional data and functional weights over the observed timestamps, and the decoder to apply a recovery layer that maps the finite-dimensional vector extracted from the functional data back to the functional space using a set of predetermined basis functions. The developed architecture can accommodate both regularly and irregularly spaced data. Our experiments demonstrate that the proposed method outperforms functional principal component analysis (FPCA) in terms of prediction and classification. Meanwhile, our approach has superior smoothing ability and better computational efficiency in comparison to the conventional AE under both linear and nonlinear settings. A manuscript of this chapter is presently under review, and a preprint is available online (Wu et al., 2024).

3.1 Introduction

FDA has found extensive application and received growing attention across diverse scientific domains. Functional data, as the core of FDA, is defined as any random variables that assume values in an infinite-dimensional space, such as time or spatial space in theory (Ramsay and Silverman, 2005; Ferraty and Vieu, 2006) , and usually discretely observed at some regularly or irregularly spaced points over the time span in applications. Due to the complexity and difficulty in interpreting and analyzing infinite-dimensional variables, a

common pipeline for FDA is to represent the infinite-dimensional functional data, denoted as $X(t)$, by a finite-dimensional vector of coefficients that extract and summarize the useful information carried by the individual functions (Yao et al., 2021). These coefficients can be of interests themselves or be readily utilized in further analysis (Wang et al., 2016a).

Two predominate approaches for dimensional reduction in FDA are basis expansion and FPCA. The first approach, the conventional basis expansion, represents the functional data as $X_i(t) = \sum_{m=1}^{M_B} c_{im}\phi_m(t)$, where $\phi_m(t)$ are known basis functions and c_{im} are corresponding basis coefficients for the i-th subject containing the information from the original functions (Ramsay and Silverman, 2005). This method requires the predetermination of a basis system, for instance, Fourier or B-spline, and the number of basis functions M_B, in order to learn the representation of functional data. The second approach, FPCA (Ramsay and Silverman, 2005; Ferraty and Vieu, 2006; Sang et al., 2017), is a fully data-driven method that compresses the functional data $X_i(t)$ into functional principal component (FPC) scores as $\xi_{im} = \int\{X_i(t) - \mu(t)\}\psi_m(t)dt$, where $\mu(t)$ is the mean function of $X(t)$, and $\psi_m(t)$'s are the FPCs which are also the eigenfunctions derived from the spectral decomposition of the variance-covariance function of $X(t)$. By Karhunen-Loève expansion, FPCA can construct the functional data as $X_i(t) = \mu(t) + \sum_{m=1}^{M_P} \xi_{im}\psi_m(t)$, with a predetermined proportion of explained variation, which indirectly defines M_P, the number of FPCs identified. The theoretical details and results on asymptotic distributions have been well derived and fully discussed by Dauxois et al. (1982), Hall and Hosseini-Nasab (2006) and Hall and Hosseini-Nasab (2006).

Representations such as FPC scores have been widely used for establishing functional regression models (Yao et al., 2005b; Müller and Yao, 2008; Yao et al., 2010), clustering (Chiou and Li, 2007; Peng and Müller, 2008) and classification (Müller, 2005; Müller and Stadtmüller, 2005) of functional curves. Both aforementioned methods are fundamentally linear mappings from infinite-dimensional data to the vector of finite scalars, however, learning linear projections of functional data might not be sufficient and informative. Furthermore, FPCA relies on the assumption of a common variance-covariance of all curves, which might be violated when the individual trajectories are labelled with classes.

Numerous extensions to the conventional FPCA have been suggested to adapt the linear representation of functional data for diverse scenarios (Yao et al., 2005a; Chen and Lei, 2015; Peng and Paul, 2009; Sang et al., 2017; Zhong et al., 2022). Nevertheless, limited contributions on nonlinear representation learning of functional data can be found in latest literature. Song and Li (2021) extended the standard FPCA to a nonlinear additive functional principal component analysis (NAFPCA) for vector-valued functional data to accommodate nonlinear functions of functional data via two additively nested Hilbert spaces. Similar to the linear FPCA with discrete functional data, however, this technique requires to first estimate the underlying $X(t)$ using the basis expansion or the reproducing Kernel Hilbert space method in the first-level function space. Chen and Müller (2012) developed nonlinear

manifold learning to generate nonlinear representations of functional data by modifying the existing nonlinear dimension reduction methods to satisfy functional data settings. The manifold-based representation is basically designed to be layered on the representation produced by FPCA, while its computational difficulties may arise as the sample size increases.

Meanwhile, the advent use of big data and the gradual popularity of deep learning promote the introduction of neural networks to functional data representation learning. Wang and Cao (2023a) explored a functional nonlinear learning method, namely FunNoL, which relies on recurrent neural networks (RNNs) to represent multivariate functional data in a lower-dimensional feature space and handle the missing observations and excessive local disturbances of observed functional data. This method ignores the basic structure of functional data as it regards $X(t)$ as time series data and captures the temporal dependency across time sequences. Moreover, to enable the use of representation for classifying curves, FunNoL is designed to be a semi-supervised model that combines a classification model with a standard RNN, introducing more complexity to network optimization and representation learning. Hsieh et al. (2021) defined a functional autoencoder that generalizes the conventional neural network AEs to handle continuous functional data, and developed the functional gradient-based learning algorithm for optimizing the AE to study the nonlinear projection of multidimensional functional data. This approach requires smooth functional inputs and overlooks the common issue where functional data are barely fully observed in practice (Yao et al., 2005a).

The main objective of this study is to propose a solution to the nonlinear representation learning and smoothing of discrete functional data using a novel functional autoencoder (FAE) based on a densely feed-forward neural network, which includes FPCA as a special case under the linear representation setting. As an unsupervised learning technique, AEs have been frequently used for feature extraction and representation learning in vector-space problems (Hinton and Salakhutdinov, 2006; Wang et al., 2016b; Meiler et al., 2001; Bengio et al., 2013). A traditional AE consists of an encoder and a decoder connected by a bottleneck layer, where the former one is a mapping from a P-dimensional vector-valued input space to a d-dimensional representation space and the latter one maps from the d-dimensional representation space back to a vector-valued output space of P dimensions, where the output layer consists of a reconstruction of the original input. Assuming $d << P$, the neurons in the bottleneck layer serve as a lower-dimension representation of the input. This representation is a collection of neuron features extracted from the AE and can be of interests themselves or can be used for further research. The relation between AE and principal component analysis (PCA) has been well discussed in several existing studies. Oja (1982, 1992) demonstrated that a neural network employing a linear activation function essentially learns the principal component representation of the input data. Furthermore, Baldi and Hornik (1989) and Bengio et al. (2013) demonstrated that an autoencoder with one hidden layer and identity activation is essentially equivalent to PCA. Bourlard and Kamp (1988) and Baldi and Hornik

(1989) also explained that the representation captured by such autoencoders is a basis of the subspace spanned by the leading principal components (PCs) instead of necessarily coincident with them. The connection between conventional AE and PCA can be naturally transplanted to that of the designed FAE and FPCA with a relevant discussion provided.

Specifically, in this work, we propose to construct an autoencoder under discrete functional data settings. We design the encoder to incorporate a projection layer computing the inner product of the functional data and functional weights over the observed discrete time spans, and the decoder to equip a recovery layer to map the finite-dimensional vector extracted from the functional data to functional space using a set of preselected basis functions. The developed architecture compresses the discretely observed functional data to a set of representations and then outputs smooth functions. The resulting lower-dimensional vector will be the representation/encoding of the functional data, which serves a similar purpose to the basis coefficients or FPC scores previously mentioned and can be inputted into any further analysis.

The autoencoder we design for functional data have at least the following highlights. First, the proposed FAE addresses the learning of a nonlinear representation from discrete functional data with a flexible nonlinear mapping path captured by neural networks, eliminating the conduct of curve smoothing assuming any particular form in advance. In other words, our method performs a one-step model simultaneously learning the representative feature and smoothing the discretely observed trajectories. Second, it allows us to obtain linear and nonlinear projections of functional data, with the former path serving as an alternative approach to FPCA. Third, the proposed method is applicable for both regularly and irregularly spaced data, while the smoothness of the recovered curves is controlled through a roughness penalty added to the objective function in model training. Forth, the architecture of the FAE is flexibly programmable and compatible with existing neural networks libraries/modules. Last but not the least, the robustness and efficiency of our method in representation extraction and curve recovery with small size of data and substantial missing information are supported by the results of various numerical experiments.

The remainder of this chapter proceeds in the following manner. In Section 3.2, we provide the methodological details for the proposed FAE, including a description about the network architecture and an explanation on the corresponding training procedure. A brief discussion on the connections between the proposed method and two well-established methods, FPCA and AE, is given in Section 3.3. In Section 3.4, we compare the proposed FAE with FPCA and AE for functional data representation, focusing on relationship capture and computational efficiency through extensive simulation studies across various scenarios. The designed FAE and the other techniques in comparison are further evaluated in Section 3.5 with a real data application. Finally, we conclude with a discussion and future directions in Section 3.6. The preprocessed data sets and computing codes of the proposed method on selected applications are available at `https://github.com/CedricBeaulac/FAE`.

3.2 Functional Autoencoders (FAEs)

3.2.1 Motivation: Autoencoders for Continuous Functional Data

Suppose there are N subjects and for the i-th subject, a functional variable $X_i(t), t \in \mathcal{T}$ is observed in the $L^2(t)$ space. To address the limitations of linear representations of functional data $X(t)$, we propose to learn nonlinear mappings from functional data space $L^2(t)$ to K-dimensional vector space $\mathbb{R}^K$ through a neural network autoencoder which contains an encoder compressing the functional input to some scalar-valued neurons, and a decoder reconstructing the functional input back from the encoded representations.

We introduce an autoencoder with L hidden layers (excluding the input and output layers) for continuous functional data $X(t)$, which we suppose, are fully observed over a continuum t and $t \in \mathcal{T}$. Different from conventional AEs consuming scalar inputs, in this scenario, functions are served as inputs and fed into the neural network, and the designed autoencoder for continuous functional data is supposed to be trained by minimizing the reconstruction error $L(X(t), \hat{X}(t)) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \int_{\mathcal{T}} (X_i(t) - \hat{X}_i(t))^2 dt$.

We propose to encode the infinite-dimensional functions to some finite number of numerical neurons by introducing functional weights $w^I(t)$ to bridge the input and the first hidden layer of the encoder. Specifically, the scalar inner product, which connects neurons in the input and the first hidden layer of the conventional AE, is generalized by the inner product of the functional input $X(t)$ and functional weight $w^I(t)$ in L^2 space. Consequently, the k-th neuron in the first hidden layer $h_k^{(1)}$ is computed as

$$h_k^{(1)} = g\left(\int_{\mathcal{T}} X(t) w_k^I(t) dt\right), \tag{3.1}$$

where $w_k^I(t)$ is the input functional weight connecting the functional input and the k-th neuron in the first hidden layer, and $g(\cdot)$ is the activation function. To be noted that here we opt to neglect the numerical bias term $b^{(1)}$ for simplicity.

The proposed functional weights together with the inner product of two functions achieve the mapping from L^2 to $\mathbb{R}^{K^{(1)}}$, where $K^{(l)}$ is the number of neurons in the l-th hidden layer and $l \in \{1, 2, ..., L\}$. The resulting numerical neurons are further passed to the continuous hidden layers of the autoencoder, following the same calculation rules as in conventional AEs. Accordingly, the k-th neuron in the l-th hidden layers is given by

$$h_k^{(l)} = g\left(\sum_{j=1}^{K^{(l-1)}} h_j^{(l-1)} w_{jk}^{(l)}\right), \tag{3.2}$$

with h_j^{l-1} being the j-th neuron in the $l-1$ layer connected by the scalar network weight $w_{jk}^{(l)}$.

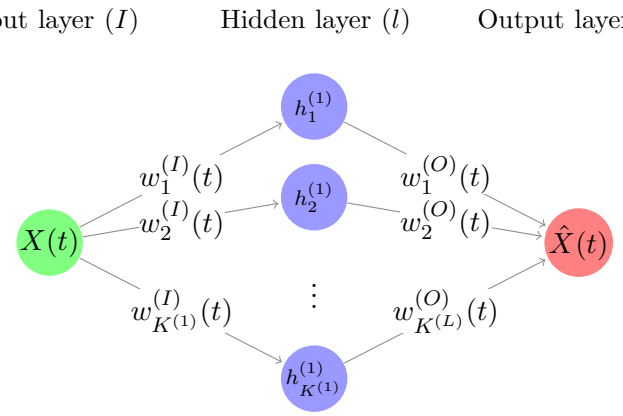

Figure 3.1: Functional autoencoder for continuous data with $L = 1$ hidden layer.

Similarly, a set of functional weights $\{w_k^O(t)\}_{k=1}^{K^{(L)}}$, instead of scalar weights, are applied at the output layer of the decoder to map the second to last layer from $\mathbb{R}^{K^{(L)}}$ back to functional space L^2 and mathematically, the outputted functional neuron is calculated as

$$\hat{X}(t) = \sum_{k=1}^{K^{(L)}} h_k^{(L)} w_k^{(O)}(t), \tag{3.3}$$

where $w_k^O(t)$ is the output functional weight connecting the k-th neuron in the L-th hidden layer and the output functional neuron. For this output layer to produce the functional output required, the linear activation function must be used.

We name this autoencoder the continuous functional autoencoder (CFAE) and a graphical visualization of the CFAE with $L = 1$ hidden layer(s) for functional data can be seen in Figure 3.1. In this scenario, $\{h_1^{(1)}, h_2^{(1)}, ..., h_{K^{(1)}}^{(1)}\}$ is regarded as the vector-valued representation of $X(t)$.

3.2.2 Proposed Model: Autoencoders for Discrete Functional Data

The CFAE introduced in Section 3.2.1 serves as an inspiration for the model we proposed to better suit discrete functional data, which reflects how functional data are collected and stored in practical applications. Considering the functional data are discretely observed at J evenly spaced time points $t_1, ..., t_J$ over the time interval $\mathcal{T}$ for all N subjects, and therefore for the i-th subject, we obtain J pairs of observations $\{t_j, X_i(t_j)\}$, $j = 1, 2, ...J$. As a matter of fact, the real functional data are often contaminated with some observational errors, resulting in a collection of noisy discrete observations $\tilde{X}_i(t_j) = X_i(t_j) + \epsilon_i(t_j)$, where $\epsilon_i(t_j)$ is the i.i.d. measurement error. Without knowing the true underlying curves $X(t)$'s, the contaminated observations $\{\tilde{X}(t_j)\}_{j=1}^J$'s are employed as an alternative to $\{X(t_j)\}_{j=1}^J$'s in applications.

We propose to adapt the CFAE to take data $\{X(t_1), X(t_2), ..., X(t_J)\}$, a J-dimensional vector, as the input. Instead of smoothing the discrete data and then applying the previously defined autoencoder, we develop the architecture to satisfy such discrete functional input. This is a major advantage of our proposed method as the data no longer needs to be preprocessed in any way before being fed to our proposed FAE. To do so, we replace the weight functions $w_k^I(t)$ for the input layer and $w_k^O(t)$ for the output layer with their discrete versions $\{w_k^I(t_j)\}_{j=1}^J$ and $\{w_k^O(t_j)\}_{j=1}^J$ evaluated at the corresponding J time points $t_1, ..., t_J$, respectively. Naturally, the integral $\int_{\mathcal{T}} X(t)w_k^{(I)}(t)dt$ in Eq.(3.1) is approximated numerically using the rectangular or trapezoidal rule, and accordingly the k-th neuron in the first hidden layer is updated as

$$h_k^{(1)} = g\left(\sum_{j=1}^{J} \omega_j X(t_j) w_k^{(I)}(t_j)\right), \tag{3.4}$$

where $\{\omega_j\}_{j=1}^J$ are the weights used in the numerical integration algorithm.

Likewise, the output layer now consists of J neurons corresponding to the J-dimensional input vector as

$$\hat{X}(t_j) = \sum_{k=1}^{K^{(L)}} h_k^{(L)} w_k^{(O)}(t_j). \tag{3.5}$$

As illustrated in Figure 3.2, the mappings $L^2 \to \mathbb{R}^{K^{(1)}}$ and $\mathbb{R}^{K^{(L)}} \to L^2$ in the CFAE are substituted with $\mathbb{R}^J \to \mathbb{R}^{K^{(1)}}$ and $\mathbb{R}^{K^{(L)}} \to \mathbb{R}^J$, respectively, for this discrete setting.

The autoencoder we proposed for discrete functional data seemingly behaves the same as a conventional AE, however, our FAE requires a different training process which accounts for the assumption that functional data are the realization of a underlying smooth stochastic process. This way, the proposed FAE considers the serial correlation of the functional data and returns a smooth and continuous functional data without the need of smoothing in the input preemptively.

We further propose to represent the functional weights used in the input and output layers as $w_k^{(\cdot)}(t) = \sum_{m=1}^{M_k^{(\cdot)}} c_{mk}^{(\cdot)} \phi_{mk}^{(\cdot)}(t)$, where $\{\phi_{mk}^{(\cdot)}(t)\}_{m=1}^{M_k^{(\cdot)}}$'s are some known basis functions from a selected basis system for the k-th functional weight, such as Fourier or B-spline, $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$ are the corresponding basis coefficients remain to be determined, and $M_k^{(\cdot)}$ is some predefined truncation integer for the k-th weight. The snapshots of the k-th input or output weight function are accordingly marked as

$$w_k^{(\cdot)}(t_j) = \sum_{m=1}^{M_k^{(\cdot)}} c_{mk}^{(\cdot)} \phi_{mk}^{(\cdot)}(t_j), \tag{3.6}$$

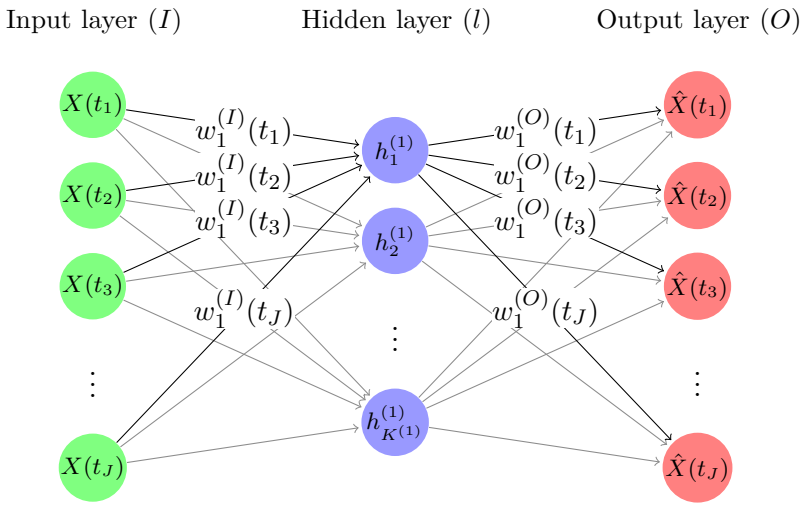

Figure 3.2: Functional autoencoder for discrete data with $L = 1$ hidden layer.

and therefore Eq.(3.4) and Eq.(3.5) calculating the k-th neurons in the first hidden layer and the output layer, respectively, can be re-written as

$$\begin{aligned} h_k^{(1)} &= g\left(\sum_{j=1}^{J} \omega_j X(t_j) \sum_{m=1}^{M_k^{(I)}} c_{mk}^{(I)} \phi_{mk}^{(I)}(t_j)\right) \\ &= g\left(\sum_{m=1}^{M_k^{(I)}} c_{mk}^{(I)} \sum_{j=1}^{J} \omega_j X(t_j) \phi_{mk}^{(I)}(t_j)\right), \end{aligned} \tag{3.7}$$

$$\begin{aligned} \hat{X}(t_j) &= \sum_{k=1}^{K^{(L)}} h_k^{(L)} \sum_{m=1}^{M_k^{(O)}} c_{mk}^{(O)} \phi_{mk}^{(O)}(t_j) \\ &= \sum_{k=1}^{K^{(L)}} \sum_{m=1}^{M_k^{(O)}} h_k^{(L)} c_{mk}^{(O)} \phi_{mk}^{(O)}(t_j). \end{aligned} \tag{3.8}$$

In such a manner, the problem of learning the functional weights $w_k^{(\cdot)}(t)$'s using typical machine learning techniques becomes one of learning $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$, the parameters defining $w_k^{(\cdot)}(t_j)$, for $k = 1, ..., K^{(l)}$, $l = 1$ or L, and consequently, we seek to learn the coefficients $\{c_{mk}^{(\cdot)}\}_{m=1}^{M_k^{(\cdot)}}$ through backpropagation.

Encoder with a Feature Layer

For computational convenience, we let $M_k^{(I)} = M^{(I)}$ and $\phi_{mk}^{(I)}(t) = \phi_m^{(I)}(t)$ for all $k \in \{1, 2, ..., K^{(1)}\}$, indicating that the input weight functions $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$ are expressed with

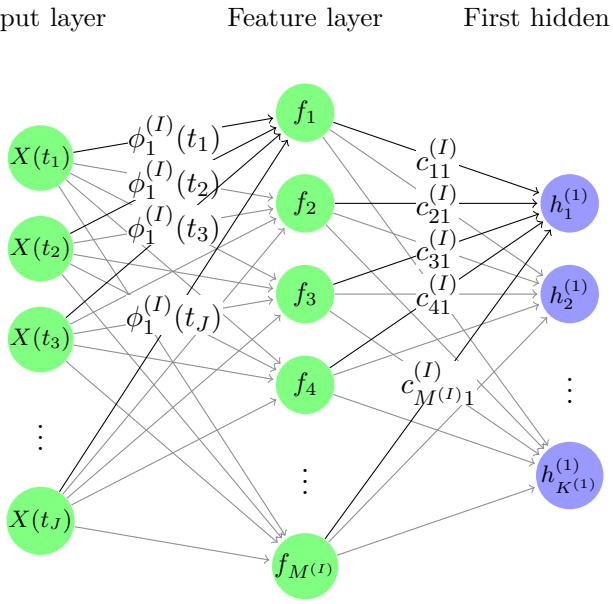

Figure 3.3: Encoder with a *feature layer*. Notice that the input and feature layers are devoid of parameters at this point and are entirely deterministic given the data and the choice of basis functions for $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$.

the same basis expansion. In consequence, we can simplify Eq. (3.7) as

$$\begin{aligned} h_k^{(1)} &= g\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} \sum_{j=1}^{J} \omega_j X(t_j)\phi_m^{(I)}(t_j)\right) \\ &= g\left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} f_m\right), \end{aligned} \tag{3.9}$$

where $f_m = \sum_{j=1}^{J} \omega_j X(t_j)\phi_m^{(I)}(t_j), m = \{1, 2, ..., M^{(I)}\}$, a Riemann sum approximating the inner product of $X(t)$ and $\phi_m^{(I)}(t)$. $\{f_m\}_{m=1}^{M^{(I)}}$ represent the resulting *features* of $X(t)$ projected to the basis function sets and serve as the pivot connecting the input layer and the first hidden layer. Hence, we design our proposed encoder by inserting a *feature layer* of f_m's between the input layer and the first hidden layer, as shown in Figure 3.3. This deterministic layer translates discretely observed functional data into a scalar structure that can then be processed with existing neural network models and training algorithms.

Specifically, the input layer and the *feature layer* are linked by the snapshots of the basis function set $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}$, while $c_{mk}^{(I)}$ becomes the network weights connecting the *feature layer* and the first hidden layer.

A benefit of this layer architecture is that different observations which might be observed at different time points will all get converted to the same features. Consequently, irregularly observed data are managed through this deterministic layer. It is also possible to recover

the continuous functional weights $\{w_k^{(I)}(t)\}_{k=1}^{K^{(1)}}$ for visualization purpose after learning the coefficients $\{c_{mk}^{(I)}\}_{m=1}^{M^{(I)}}$.

Decoder with a Coefficient Layer

Again, for simplicity, we set the basis functions to be the same for representing all output weight functions $\{w_k^{(O)}(t)\}_{k=1}^{K^{(L)}}$ by setting $M_k^{(O)} = M^{(O)}$ and $\phi_{mk}^{(O)}(t) = \phi_m^{(O)}(t)$ for all $k \in \{1, 2, ..., K^{(L)}\}$. Following on Eq.(3.8), we now have:

$$\begin{aligned}
\hat{X}(t_j) &= \sum_{k=1}^{K^{(L)}} \sum_{m=1}^{M^{(O)}} h_k^{(L)} c_{mk}^{(O)} \phi_m^{(O)}(t_j) \\
&= \sum_{m=1}^{M^{(O)}} \left(\sum_{k=1}^{K^{(L)}} h_k^{(L)} c_{mk}^{(O)} \right) \phi_m^{(O)}(t_j) \\
&= \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t_j), \qquad (3.10)
\end{aligned}$$

where $b_m = \sum_{k=1}^{K^{(L)}} h_k^{(L)} c_{mk}^{(O)}$. In fact, $\{\phi_m^{(O)}(t)\}_{m=1}^{M^{(O)}}$ can be regarded as the basis functions used in the representation of $\hat{X}(t)$, the reconstructed functional observation. In turn, $\{b_m\}_{m=1}^{M^{(O)}}$ play the role of the corresponding basis coefficients.

Hence, in Figure 3.4, we visualize b_m's as the neurons of a *coefficient layer* added to the decoder for connecting the last hidden layer and the output layer, while $c_{mk}^{(O)}$ are the network weights between the last hidden layer and the *coefficient layer*. Meanwhile, the *coefficient layer* and the output layer are connected deterministically through snapshots of the basis functions $\{\phi_m^{(O)}(t)\}_{m=1}^{M^{(O)}}$. The proposed decoder is essentially and functionally consistent with NNBR, a neural network designed for scalar input and functional output, developed by Wu et al. (2023), since both approaches decompress the scalar-valued basis coefficients to the functional curves in a linear manner, ensuring the use of backpropagation in model training.

Likewise, an advantage of this layer architecture is its ability to easily handle irregularly spaced data, as explained in Wu et al. (2023). It also provides a smooth reconstruction of the input functional data, which can be evaluated at any point on the domain, effectively smoothing the functional data while simultaneously learning a meaningful and useful representation.

Training the Proposed FAE

A full architecture (with $L = 1$) of the proposed FAE is displayed in Figure 3.5. As detailed in the previous parts of this section, a deterministic *feature layer* of size $M^{(I)}$ is created to follow the input layer without using any unknown parameters or weights for neuron calculation, and each neuron in the *feature layer* produces a scalar value computed as the

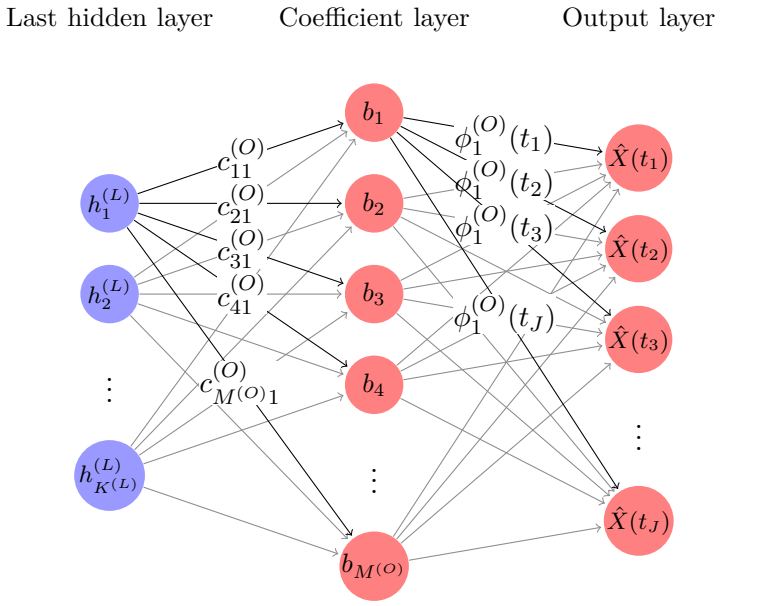

Figure 3.4: Decoder with a *coefficient layer*. Similarly, the last two layers are devoid of parameters and are deterministic.

numerical approximation of the inner product of the input $X(t_j)$ and the preselected basis function $\phi_m^{(I)}(t_j)$ over the observed timestamp. On the other end, a *coefficient layer* of $M^{(O)}$ scalar-valued neurons is handcrafted as the second to last layer, and it connects the output layer through the known basis functions $\phi_m^{(O)}(t)$'s, making the output layer also deterministic. Layers between the *feature layer* and the *coefficient layer* share the same structure as a conventional AE. This specific structure is the essence of the proposed FAE.

Same as traditional AEs, the training process of FAEs comprises of two components, the *forward propagation* and the *backward propagation*, and can be operated using existing neural network libraries or modules, such as **PyTorch** (Paszke et al., 2019) and **TensorFlow** (Abadi et al., 2015). The *forward propagation* has been previously depicted and is summarized in Algorithm 5. Here we put emphasize on the *backward propagation* that updates the network parameters using gradient-based optimizers.

Let $\theta = \{c_{mk}^{((I)}, c_{mk}^{(O)}, \eta\}$ denote the collection of network parameters, where η stands for all the network weights involved in connecting the hidden layers. The training process targets at finding $\hat{\theta} = \text{argmin}_\theta L(X(t_j), \hat{X}(t_j))$, and we employ the standard mean squared error (MSE) between $X(t_j)$ and $\hat{X}(t_j)$ across all the observed time points $t_1, ..., t_J$ and all subjects in the training set as the reconstruction error of the FAE, in specific, $L(X(t_j), \hat{X}(t_j)) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \sum_{j=1}^{J} (X_i(t_j) - \hat{X}_i(t_j))^2$. We design the output layer of FAE to be a linear combination of some preselected basis functions $\{\phi_m^{(O)}\}_{m=1}^{M^{(O)}}$ and the neurons $\{b_m\}_{m=1}^{M^{(O)}}$ outputted by the second to last layer (the *coefficient layer*), and therefore the neuron $\hat{X}(t_j)$ in the output layer of FAE, which is the snapshot of the reconstructed curve $\hat{X}(t)$

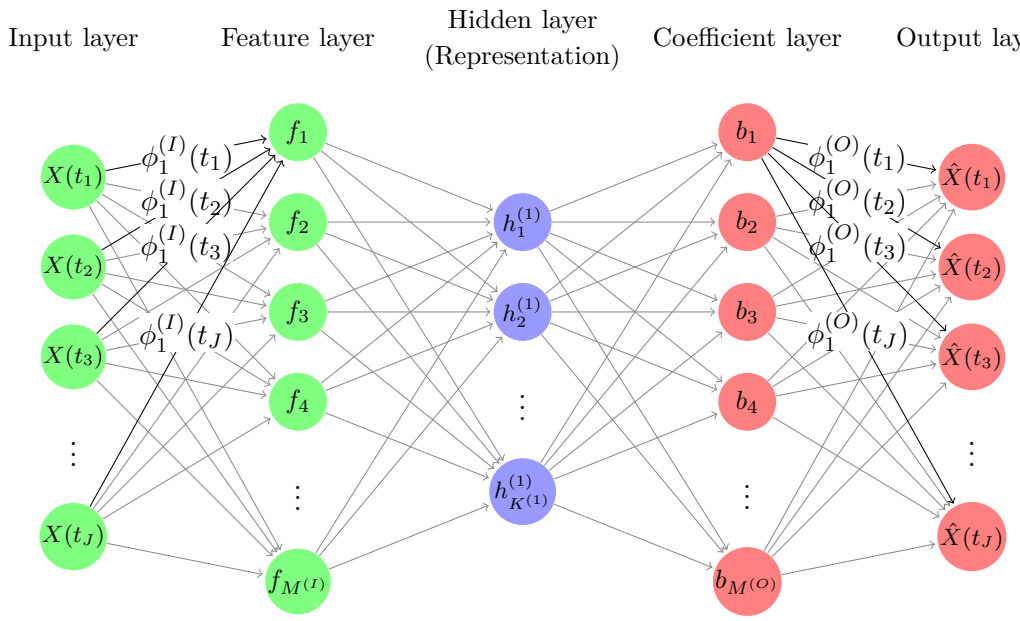

Figure 3.5: A graphical representation of the FAE we propose for discrete functional data. The model represented only has a single hidden layer h, that serves the role of latent representation.

at time t_j is the vector product of $\{b_m\}_{m=1}^{M^{(O)}}$ and $\{\phi_m^{(O)}\}_{m=1}^{M^{(O)}}$ evaluated at the specific t_j. The linear relation between the *coefficient layer* and the output layer, together with the differentials of the known basis functions, ensures the feasibility of computing the gradient of $(X_i(t_j) - \hat{X}_i(t_j))^2$ with respect to the coefficients b_m as:

$$\frac{\partial L}{\partial b_m} = \frac{\partial L}{\partial \hat{X}(t_j)} \frac{\partial \hat{X}(t_j)}{\partial b_m}. \tag{3.11}$$

The gradient with respect to the network weights η in the remaining layers prior to the *coefficient layer* can be subsequently computed in the backward manner as that in a classic neural network until reaching the *feature layer*, while no any further gradient calculation is made from the *feature layer* back to the input layer because they are connected by the predefined input basis functions $\{\phi_m^{(I)}\}_{m=1}^{M^{(I)}}$, instead of some network parameters in need of training. Algorithm 6 details the gradient calculation procedure used to update network parameters in the *backward propagation*.

3.2.3 FAE as a Functional Data Smoother

By design, our proposed FAE outputs a smooth continuous curve over the entire interval of interest as an estimate of the underlying stochastic process for any input distinctly observed

Algorithm 5: FAE Forward Pass

Input: $\boldsymbol{X} = \{X(t_1), X(t_2), ..., X(t_J)\}$

Output: $\hat{\boldsymbol{X}} = \{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

Hyperparameters: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}, \{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, ω_j for all j, a predefined network NN(θ) with L hidden layers, $K^{(l)}$ neurons in the l-th hidden layer, activation functions $g_1, ..., g_L$, E epochs, Optimizer (including learning rate ϱ), etc.

Input Layer $\rightarrow$ ***Feature Layer***

$\{X(t_j)\}_{j=1}^{J} \rightarrow f_m = \sum_{j=1}^{J} \omega_j X(t_j) \phi_m^{(I)}(t_j), m \in \{1, 2, ..., M^{(I)}\}$

Feature Layer $\rightarrow$ ***Coefficient Layer***

$\{f_m\}_{m=1}^{M^{(I)}} \rightarrow b_m = \sum_{k=1}^{K^L} c_{mk}^{(O)} g_L \left(\cdots g_1 \left(\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} f_m \right) \right), m \in \{1, 2, ..., M^{(O)}\}$

Specifically, the k-th neuron in the l-th hidden layer is constructed the same way as that in conventional neural networks as $h_k^{(l)} = g_l(\sum_{j=1}^{K^{(l-1)}} h_j^{(l-1)} w_{jk}^{(l)})$, and $w_{jk}^{(l)}$'s are the scalar network weights

Coefficient Layer $\rightarrow$ **Output Layer**

$\{b_m\}_{m=1}^{M^{(O)}} \rightarrow \hat{X}(t_j) = \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t_j), j \in \{1, ..., J\}$

return $\{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

---

functional data by

$$\hat{X}(t) = \sum_{k=1}^{K^{(L)}} \sum_{m=1}^{M^{(O)}} h_k^{(L)} c_{mk}^{(O)} \phi_m^{(O)}(t) = \sum_{m=1}^{M^{(O)}} b_m \phi_m^{(O)}(t), \tag{3.12}$$

which is achievable thanks to the continuity of the preselected basis functions $\phi_m^{(O)}(t)$'s. This is a core design choice made so that the FAE we propose acts not only as a representation learner but a smoother itself and could substitute other smoothing processes such as fitting a B-spline model.

Following the tradition in FDA, we can promote the smoothness of the output curves by adding a roughness penalty to the objective function of the FAE. With a consideration for computational simplicity, among different choices of roughness penalties, we propose to apply the difference penalty (Eilers and Marx, 1996) on the elements of the *coefficient layer* as they act as the basis coefficients in the output functional curves. Consequently, including such a penalty term leads to the following objective function,

$$L_{\boldsymbol{pen}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(\sum_{j=1}^{J} \left(X_i(t_j) - \hat{X}_i(t_j) \right)^2 + \lambda \sum_{m=3}^{M^{(O)}} (\Delta^2 b_{im})^2 \right) \tag{3.13}$$

and $\Delta^2 b_{im} = b_{im} - 2b_{i(m-1)} + b_{i(m-2)}$, where b_{im} is the m-th neuron in the *coefficient layer* for the i-th training subject, and parameter λ controls the smoothness. In implementation,

Algorithm 6: FAE Backward Pass

Input: $\theta_{\text{current}}, \{X(t_1), X(t_2), ..., X(t_J)\}, \{\hat{X}(t_1), \hat{X}(t_2), ..., \hat{X}(t_J)\}$

Output: θ_{updated}

Hyperparameters: $\{\phi_m^{(I)}(t_j)\}_{m=1}^{M^{(I)}}, \{\phi_m^{(O)}(t_j)\}_{m=1}^{M^{(O)}}$, ω_j for all j, a predefined network NN(θ) with L hidden layers, $K^{(l)}$ neurons in the l-th hidden layer, activation functions $g_1, ..., g_L$, E epochs, Optimizer (including learning rate ϱ), etc.

Compute loss function $L(X(t_j), \hat{X}(t_j))$

Set $\theta = \theta_{\text{current}}$

Output Layer $\rightarrow$ ***Coefficient Layer***

$\frac{\partial L(\theta)}{\partial b_m} = \frac{\partial L(\theta)}{\partial \hat{X}(t_j)} \frac{\partial \hat{X}(t_j)}{\partial b_m}$, because $\hat{X}(t_j) = f(b_m)$ and $f'(b_m)$ exists

Coefficient Layer $\rightarrow$ ***Feature Layer***

$\frac{\partial L(\theta)}{\partial \theta}$, same gradient calculation as used in traditional neural networks

Feature Layer $\rightarrow$ **Input Layer**

No gradient calculation involved (deterministic operation)

Update NN parameters θ^*

return $\theta_{\text{updated}} = \theta^*$

we suggest applying a roughness penalty when $M^{(O)}$ is relatively large ($M^{(O)} >> J$). The optimal λ can be selected using cross-validation.

3.2.4 FAE for Irregularly Spaced Observations

For many existing FDA models, it is quite common to assume that the observed discrete functional data are regularly spaced. A benefit of our designed FAE is that it is free of this assumption and its input layer can actually be of flexible size because of the proposed *feature layer* applied in the early stage of the model.

As detailed in section 3.2.2, we express the input functional weights $w_k^{(I)}(t_j)$ by a fixed representation of $\sum_{m=1}^{M^{(I)}} c_{mk}^{(I)} \phi_m^{(I)}(t_j)$, and therefore every discrete functional input $X_i(t_{ij}), j = 1, ..., J_i$, where J_i varies with i, are all equivalently projected to the same $M^{(I)}$ basis functions, forming the t_{ij}-free features $f_{im} = \sum_{j=1}^{J_i} \omega_{ij} X_i(t_{ij}) \phi_m^{(I)}(t_{ij}), m = \{1, 2, ..., M^{(I)}\}$. These $M^{(I)}$ features then participate in the following forward pass in place of the actual functional inputs $X_i(t_{ij})$ for training the same set of network parameters including the input weight coefficients $c_{mk}^{(I)}$, which are free of i.

The designed *feature layer*, combined with the input functional weight representation, processes irregular inputs by generalizing the problem of estimating irregular snapshots of input weight functions to estimating input weight coefficients that are consistent over all subjects.

3.3 Connection with Existing Models

3.3.1 Relation with FPCA

As previously pointed out by Baldi and Hornik (1989), Bengio et al. (2013), and Bourlard and Kamp (1988), a single-hidden-layer linear autoencoder with its objective function being the squared reconstruction error, i.e., $L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \|X_i - W_{\text{d}} W_{\text{e}} X_i\|^2 = \sum_{i=1}^{N_{\text{train}}} \sum_{p=1}^{P} \{X_{ip} - (W_{\text{d}} W_{\text{e}} X_i)_p\}^2$, where $X_i = \{X_{i1}, X_{i2}, ..., X_{iP}\}$, W_{d}, W_{e} denote the i-th network input of P dimensions, weight matrix of the decoder and weight matrix of the encoder, respectively, is approximately identical to the conventional PCA, because such an autoencoder is learning the same subspace as the PCA. More precisely, the unique global minimum of L is corresponding to the orthogonal projection of X onto the subspace spanned by the leading eigenvectors (principal components) of the covariance matrix of X. It is worth mentioning that at the global minimum, the uniqueness occurs with the global map $W_{\text{d}} \times W_{\text{e}}$, while the matrices W_{d} and W_{e} may not be unique. This is because for multiple appropriate C we have $W_{\text{d}} \times W_{\text{e}} = (W_{\text{d}} C)(C^{-1} W_{\text{e}})$. In other words, the mapping $W_{\text{d}} \times W_{\text{e}}$ is unique but not the encoder and decoder weight matrices.

When it comes to the functional scenario, a homogeneous relationship exists between FAE and FPCA. For a single-hidden-layer FAE with linear activation function under continuous functional data setting, the objective function measuring the mean squared reconstruction error turns out to be

$$L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \int_{\mathcal{T}} \left\{ X_i(t) - \sum_{k=1}^{K^{(1)}} \left(\int_{\mathcal{T}} X_i(t) w_k^{(I)}(t) dt \right) w_k^{(O)}(t) \right\}^2 dt. \quad (3.14)$$

Ramsay and Silverman (2005) concluded that the aforementioned fitting criterion is minimized when the orthonormal-restricted weight functions $w^{(\cdot)}(t)$ are precisely the same set of principal component weight functions of the functional data $X(t)$. Hence, training a one-hidden-layer linear FAE with respect to the squared reconstruction error criterion and an orthonormal constrain on functional weights is exactly approaching to project the input $X(t)$ onto the subspace generated by the FPCs, the same space learned by FPCA.

For discrete functional data, the objective function becomes

$$L = \sum_{i=1}^{N_{\text{train}}} \left\| X_i - \hat{X}_i \right\|^2 = \sum_{i=1}^{N_{\text{train}}} \frac{1}{J} \sum_{j=1}^{J} \left\{ X_i(t_j) - \sum_{k=1}^{K^{(1)}} \left(\sum_{j=1}^{J} \omega_j X_i(t_j) w_k^{(I)}(t_j) \right) w_k^{(O)}(t_j) \right\}^2. \quad (3.15)$$

It is important to notice that this approximation can lead to some difference which should progressively decreases as J increases. Consequently, for relatively large values of J, the FAE optimized by minimizing Eq.(3.15) will yield functional weights that are approximately the same as those obtained by minimizing Eq.(3.14). To put it differently, when subjected to

the orthonormal constraint on the functional weights, the FAE that minimizes the objective model Eq.(3.14) is effectively learning the empirical projection of $X(t)$ onto the same space as FPCA does. Importantly, the proposed FAE with discrete configuration generalizes FPCA up to a few approximations, and the functional weights produced by FAE can be identically interpreted as the FPCs in FAE.

3.3.2 Relation with AE

As pointed out in section 3.2.2, the proposed FAE is structurally similar to the classic AEs based on fully connected neural networks. The main difference lies in the first and last layers. In detail, a classic AE consists of network weights (and bias) free of restrictions and the training task aims at optimizing these vectors of network parameters. The FAE we developed also includes such weights to link layers between the *feature layer* and *coefficient layer*, however, the difference lies in the deterministic weights before the *feature layer* and after the *coefficient layer*, which are comprised of snapshots of continuous basis functions. With the goal of optimizing the non-deterministic weights, the training process of FAE follows the same rule as used in AEs. The FAE can be regarded as an extension of a conventional AE with some deterministic operations added to both ends of the network.

The addition of *feature layer* to the AE architecture enables the FAE to quickly summarize the underlying temporal relationship among observed time span into neurons that actually step into the network, resulting in faster convergence and better generalization during network training compared to conventional AE. Meanwhile, thanks to the application of the functional output weights, the FAE we developed can recover the discrete functional data to smooth curves over a continuous interval, satisfying the smoothness requirement of functional data, while the classic AE is limited to output discontinuous functions evaluated at some discrete timestamp of observations. Additionally, our method is capable to efficiently handle irregularly spaced functional input along with its underlying correlation in the *feature layer* by adjusting the weights ω used for numerical integration calculation, while AE has to address the issue of having insufficient observations at certain time points by training the model with some null-valued input for the corresponding neurons in the input layer. The designed structure benefits our method with better performance in less computational cost when manipulating irregularity, which is further highlighted by a series of simulation studies in the following section.

3.4 Simulation Studies

In this section, we aim to compare our proposed FAE with the two existing baseline methods it extends, FPCA and AE respectively, for representation learning and curve smoothing from discretely observed functional data. We concentrate on investigating the effectiveness

of our method compared to FPCA in capturing the potential nonlinear relationship, as well as evaluating the smoothing ability and computational efficiency of FAE compared to AE.

3.4.1 Simulation Setup

Data Generation

We generate the data by first sampling a d-dimensional representation $\boldsymbol{Z}$ from a Gaussian mixture model. The mean vector and the covariance matrix of each component are designed so that components are separable. We then apply a function $f(\cdot)$ that maps the representation $\boldsymbol{Z}$, to a set of M-dimensional basis coefficients B_m. Finally, we produce the continuous functional data using a linear combination of M basis functions $\gamma_m(t)$'s and the basis coefficients B_m:

$$X(t) = \sum_{m=1}^{M} B_m \gamma_m(t) = f(\boldsymbol{Z})\boldsymbol{\gamma}. \tag{3.16}$$

Finally, we evaluate $X(t)$ at some discrete times $\{t_1, t_2, ..., t_J\} \in [0, 1]$ to obtain a discrete version of the functional data.

The basis system used is the B-spline basis system with an order of 4, and the number of bases M varies throughout experiments. In terms of the mapping function $f(\cdot)$, we employ a neural network NN$(\cdot)$ with multiple architectures aiming to create different mapping paths from the representation vector $\boldsymbol{Z}$ to the basis coefficients of the functional data. The neural network takes the d-dimensional representation vector as input and outputs the M-dimensional basis coefficients. We apply neural networks with no hidden layers and a linear activation function for linear scenarios, and networks with at least one hidden layer and nonlinear activation functions for nonlinear scenarios.

An optional Gaussian noise can be further added to the discrete functional curve to mimic observational errors. The component of the Gaussian mixture model from which the representation was sampled is used as the label for the functional data in classification experiments.

Implementation of Models

FPCA linearly encodes functional curves to FPC scores ξ_{im}'s with corresponding FPCs $\psi_m(t)$'s. We implement FPCA in `Python` relying on the **scikit-fda** library (Ramos-Carreño et al., 2022). The discrete functional data are first converted to smooth functions using basis expansion with customized number of B-spline basis functions, and then the conventional FPCA is performed on the estimated curve with a user-defined number of FPCs. The resulting FPC scores serve as the scalar representation of the functional data and are used for further statistical analyses.

AE based on a densely feed-forward network architecture can learn an encoding from the functional trajectory observed at discrete time points to a lower-dimensional vector of representation without considering any temporal correlations among the discrete observations. We design the input layer of AE to have J neurons with the j-th neuron representing the snapshot of the discrete functional observation at t_j. We adopt different architectures with a bottleneck hidden layer that produces the representation, experiment with both linear and nonlinear activation functions, and initialize the network weights to random values drawn from $\mathcal{N}(0, \sigma)$. We implement the AE using the **PyTorch** library.

Lastly, we implement the proposed **FAE** using **PyTorch**, along with the **scikit-fda** library for applying the basis expansion to functional weights. Analogously, we attempt with different architectures that include a hidden layer for extracting the representation, employ both linear and nonlinear activation functions in model training, and initialize the weights randomly by sampling from a Gaussian distribution $\mathcal{N}(0, \sigma)$.

3.4.2 Results

A series of simulations are performed under various scenarios to investigate the performance of the proposed method in both prediction and classification, comparing it with FPCA and AE individually. The prediction error is measured by the mean squared prediction error (MSE_p) averaged across the number of samples and the number of observed time points in the test set, while the classification accuracy, $\text{P}_{\text{classification}}$, is calculated as the percentage of test observations that can be labelled correctly by a logistic regression based on the representations extracted. For each scenario, we report the mean and standard deviation (SD) of the evaluation metrics across all replications.

FAE vs. FPCA

Scenario 1.1 (Linear & Regular): 6000 discrete functional observations evaluated at 21 equally spaced points over the interval $[0, 1]$ are simulated. A five-dimensional Gaussian mixture model with three components is used to generate the representations and the resulting functional curves are labelled with class 1, 2 and 3. A neural network with no hidden layers and a linear activation function is performed to map the representation to the basis coefficients. We employ 8 B-spline basis functions ($M = 8$) along with the aforementioned basis coefficients to express the underlying functional curves.

We assign 80% of the observations by random to the training set and the remainder to the test set. The FPCA and two types of FAE are successively trained and the model details are summarized in Table B.1 in Appendix B.

Scenario 1.2 (Nonlinear & Regular): We generate 3000 functional observations discretely measured at 51 equally spaced points over the interval $\mathcal{T} = [0, 1]$. We sample a 5-dimensional representation for each curve from a 3-component Gaussian mixture model and label the associated functional curves with class 1, 2 and 3. We map the representations

Table 3.1: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity activation function (FAE(Identity)), functional autoencoder with the softplus activation function (FAE(Softplus)) and functional principal component analysis (FPCA) on 10 random test data sets in Scenario 1.1, with the best results being highlighted in bold.

		FAE (Identity)	**FAE** (Softplus)	**FPCA**
MSE_p	3 Reps	0.0050(0.0001)	**0.0045**(0.0005)	0.0052(0.0001)
	5 Reps	**0.0019(<0.0001)**	0.0022(0.0003)	0.0021(<0.0001)
	10 Reps	**0.0009(<0.0001)**	0.0017(0.0005)	0.0010(<0.0001)
$P_{classification}$	3 Reps	87.24%(0.93%)	**87.72%**(1.62%)	87.68%(0.78%)
	5 Reps	87.94%(0.81%)	86.53%(0.94%)	**89.21%**(0.78%)
	10 Reps	89.16%(0.75%)	**89.61%**(0.99%)	89.22%(0.70%)

Table 3.2: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity activation function (FAE(Identity)), functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) and functional principal component analysis (FPCA) on 10 random test data sets in Scenario 1.2, with the best results being highlighted in bold.

		FAE (Identity)	**FAE** (Sigmoid)	**FPCA**
MSE_p	3 Reps	0.0070(0.0002)	**0.0038**(0.0002)	0.0070(0.0002)
	5 Reps	0.0035(0.0001)	**0.0026**(0.0004)	0.0036(0.0001)
	10 Reps	**0.0013**(<0.0001)	0.0014(<0.0001)	**0.0013**(<0.0001)
$P_{classification}$	3 Reps	85.05%(1.08%)	**88.68%**(1.46%)	85.17%(1.06%)
	5 Reps	86.62%(1.06%)	**92.42%**(1.02%)	86.65%(1.28%)
	10 Reps	87.55%(1.13%)	**91.20%**(1.06%)	87.53%(1.26%)

to the basis coefficients using a neural network with one hidden layer with 20 neurons and the sigmoid activation function. Afterwards, individual functional curve is constructed using 10 B-spline basis functions and the basis coefficients described above.

We continue to randomly generate training and test sets that contain 80% and 20% observations respectively. Again, we put FPCA, linear FAE and nonlinear FAE in comparison with model configuration adjusted and detailed in Table B.2 in Appendix B.

Table 3.1 and Table 3.2 summarize the predictive and classification performances of the proposed FAEs and FPCA. In the linear & regular context, we observe that all three approaches in comparison yield similar performance in both prediction and classification for most representation attempts.

In contrast, under the nonlinear scenario, both linear FAE (FAE with the identity activation function) and FPCA generate relatively higher MSE_p and lower $P_{classification}$

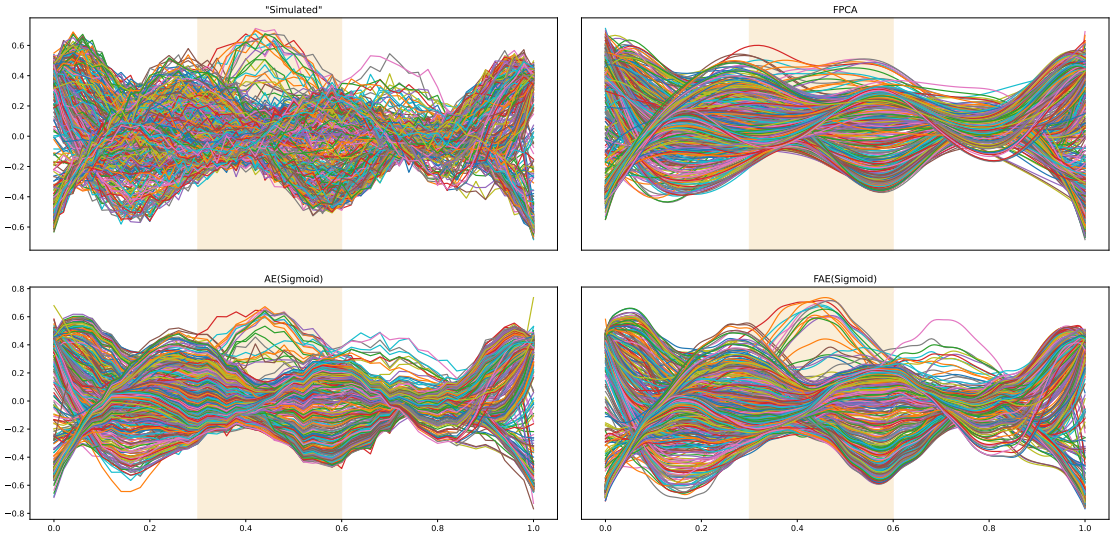

Figure 3.6: The simulated curves and the curves recovered by functional principal component analysis (FPCA), classic autoencoder with the sigmoid activation function (AE(Sigmoid)) and functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) using 5 representations for a random test set in Scenario 1.2 and Scenario 2.1.

due to the violation of the linearity assumption. Meanwhile, the FAE with the sigmoid activation function retains superior performance in both prediction and classification in comparison to the other linear approaches, with only minimal difference when predicting with 10 representations. This indicates that the nonlinear FAE can capture and comprise the information carried by the discrete data more efficiently and accurately.

With regard to curve smoothing, as displayed in Figure 3.6, both FPCA and FAE with the sigmoid activation function produce smooth curves based on the inputted discrete observations, while the designed FAE demonstrates additional benefits in curve recovery. Plainly, FAE can not only correctly reconstruct the complete moving trend but also sensitively capture the individual pop-up variations, e.g. the local $\cap$-shaped mode appearing in the shaded interval.

FAE vs. AE

Scenario 2.1 (Nonlinear & Regular): The simulated data used in the Scenario 1.2 in this section is simultaneously applied for a comparison between FAE and AE. Again, 80% of the random observations are assigned to the training set and the remaining 20% to the test set. Given that this scenario follows the nonlinear setting, we put emphasize on the nonlinear models by training the baseline model AE and the proposed FAE with model configurations listed in Table B.3 in Appendix B.

Table 3.3 presents the means and SDs of MSE_p and $\text{P}_{\text{classification}}$ over 10 replicates trained by AE and FAE with the sigmoid activation function in the nonlinear but regularly-

Table 3.3: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the sigmoid activation function (FAE(Sigmoid)) and classic autoencoder with the sigmoid activation function (AE(Sigmoid)) on 10 random test data sets in Scenario 2.1, with the better results being highlighted in bold.

	FAE (Sigmoid)			**AE (Sigmoid)**		
	3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	**0.0038**(0.0002)	**0.0026**(0.0004)	**0.0014**(<0.0001)	0.0046(0.0005)	0.0030(0.0005)	0.0124(0.0069)
$P_{classification}$	88.68%(1.46%)	92.42% (1.02%)	91.02%(1.06%)	**89.35%**(1.39%)	**92.75%** (1.15%)	**92.65%**(1.81%)

spaced-data scenario. We observe that the two methods achieve competitive performance in representation learning, with the nonlinear AE giving better results in classifying curves, while the nonlinear FAE excels with smaller predictive errors in reconstructing the functional observations. Figure 3.6 visualizes the simulated curves and the full trajectories recovered by FPCA, nonlinear AE and nonlinear FAE, respectively. It is clearly shown that the proposed FAE can directly and accurately output smooth curves using the given discrete observations for the entire domain, while AE is limited to discretely recover the curve at the time points with available observations, indicating that our FAE is capable of efficiently capturing the representative information and simultaneously smoothing the discretely functional observation.

Scenario 2.2 (Nonlinear & Irregular): In this scenario, we continue to use the data simulated in Scenario 1.2 and randomly remove measurements in 25 time points (excluding the start and end time point) for each curve to create irregularly and discretely observed functional data, that is, the resulting functional curve contains 26 irregular observations individually over the domain interval $\mathcal{T}$.

We experiment with two different training set settings: (i) the training set contains 80% observations; (ii) the training set contains 20% data, and focus on a comparison between the nonlinear AE and nonlinear FAE with configurations provided in Table B.4 in Appendix B to examine their performances in handling nonlinearity and irregularity simultaneously. For those time points without observations (randomly removed), we feed the corresponding neurons in the input layer of AE and FAE with 0 and abandon those neurons when computing the loss. When training FAE, we also adjust the weights $\{\omega_j\}_{j=1}^{J_i}$ individually for each discrete curve i for a reasonable numerical integration over all the observed timestamp.

The performances of prediction and classification of nonlinear AE and nonlinear FAE, trained with 80% and 20% irregularly spaced functional data, are illustrated in Table 3.4 and Table 3.5, separately, with the performances of both models reported for each thousand epochs. We can see that the proposed FAE shows more advantages in speedily learning the representation and accurately capturing the information for both prediction and classification, especially when the training epochs remain small. On the other hand, the classic AE needs to gradually master the mapping path in respect of reconstruction error, while its resulting representations can outperform those by FAE in classification when the training cost increases. The visual comparisons of how the mean MSE_p and mean $\text{P}_\text{classification}$ of FAE and AE change with the number of training epochs for 80% and 20% training sizes, corresponding to Table 3.4 and Table 3.5, are presented in Section B.1.2 of Appendix B. As demonstrated, the computational efficiency of the FAE is robust over different representation numbers, which further confirms that the FAE is able to generalize better and converge faster even with fewer epochs and larger batch size over traditional AE that has similar architecture in the matter of curve reconstruction and unsupervised representation learning for classification.

Table 3.4: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the softplus activation function (FAE(Softplus)) and classic autoencoder with the softplus activation function (AE(Softplus)) on 10 random test data sets when training with 80% irregularly observed data in Scenario 2.2, with the better results being highlighted in bold.

		FAE (Softplus)			**AE (Softplus)**		
		3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	epochs=1000	**0.0031**(0.0003)	**0.0023**(0.0002)	**0.0014**(0.0002)	0.0035(0.0002)	0.0029(0.0003)	0.0143(0.0127)
	epochs=2000	**0.0023**(0.0001)	**0.0015**(<0.0001)	**0.0010**(<0.0001)	0.0034(0.0003)	0.0044(0.0059)	0.0103(0.0102)
$P_{classification}$	epochs=1000	86.57%(1.08%)	87.85%(2.03%)	89.22%(1.17%)	**89.85%**(1.32%)	**91.05%**(0.69%)	**90.58%**(1.59%)
	epochs=2000	88.67%(1.22%)	90.12%(1.70%)	**91.75%**(1.10%)	**90.68%**(1.30%)	**91.03%**(1.09%)	90.73%(1.66%)

Table 3.5: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the softplus activation function (FAE(Softplus)) and classic autoencoder with the softplus activation function (AE(Softplus)) on 10 random test data sets when training with 20% irregularly observed data in Scenario 2.2, with the better results being highlighted in bold.

		FAE (Softplus)			**AE (Softplus)**		
		3 Reps	5 Reps	10 Reps	3 Reps	5 Reps	10 Reps
MSE_p	epochs=1000	**0.0057**(0.0009)	**0.0041**(0.0009)	**0.0039**(0.0026)	0.0386(0.0152)	0.0730(0.0237)	0.4591(0.2692)
	epochs=2000	**0.0046**(0.0011)	**0.0030**(0.0004)	**0.0025**(0.0007)	0.0194(0.0094)	0.0464(0.0266)	0.3579(0.2449)
	epochs=3000	**0.0035**(0.0003)	**0.0027**(0.0003)	**0.0019**(0.0004)	0.0104(0.0027)	0.0230(0.1578)	0.1917(0.0934)
	epochs=4000	**0.0031**(0.0002)	**0.0021**(<0.0001)	**0.0029**(0.0039)	0.0086(0.0012)	0.0093(0.0037)	0.0968(0.0632)
	epochs=5000	**0.0029**(0.0002)	**0.0019**(0.0001)	**0.0015**(0.0004)	0.0094(0.0010)	0.0070(0.0015)	0.0588(0.0434)
$P_{classification}$	epochs=1000	78.32%(1.10%)	**81.59%**(2.12%)	**82.30%**(2.84%)	**78.71%**(2.97%)	80.48%(2.30%)	64.36%(5.26%)
	epochs=2000	81.40%(2.00%)	83.86%(1.17%)	**83.50%**(1.20%)	**85.30%**(0.82%)	**86.16%**(1.39%)	80.63%(6.87%)
	epochs=3000	84.09%(1.06%)	85.75%(1.24%)	85.27%(1.02%)	**86.70%**(1.11%)	**87.50%**(1.57%)	**88.61%**(1.29%)
	epochs=4000	85.05%(0.72%)	86.69%(1.26%)	87.17%(1.04%)	**87.18%**(1.27%)	**88.03%**(1.17%)	**90.05%**(0.65%)
	epochs=5000	85.53%(0.94%)	87.63%(1.26%)	88.27%(1.34%)	**87.50%**(1.25%)	**88.23%**(0.86%)	**89.98%**(0.63%)

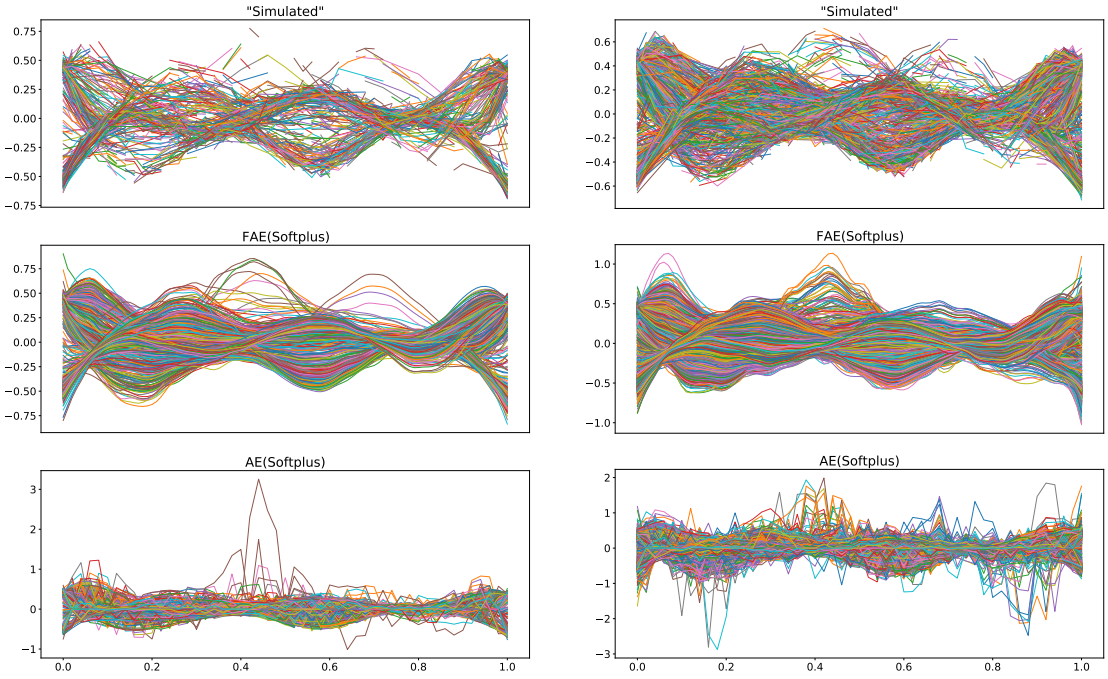

Figure 3.7: The simulated irregularly spaced curves and the curves recovered by classic autoencoder with the softplus activation function (AE(Softplus)) and functional autoencoder with the softplus activation function (FAE(Softplus)) using 5 representations for a random test set in Scenario 2.2, when training with 80% observations (left panel) and 20% observations (right panel), respectively.

Apart from representation learning, we display the simulated irregularly spaced functional segments, along with the full curves reconstructed by the nonlinear AE and nonlinear FAE in Figure 3.7 to reveal the smoothing ability of the FAE. When training with 80% observations, it is not surprising to observe that the proposed FAE oversteps the classic AE by generating predominantly smooth curves that effectively capture the entire underlying patterns and primary modes present in the originally observed data. Nevertheless, trajectories obtained through AE exhibit noticeable oscillations and incoherence with numerous accidents protrudes appearing across the entire domain. In the case of training with only 20% data, as expected, the FAE continues its dominance by showing dramatically leading performance in curve smoothing, highlighting that the FAE with specially designed architecture is able to efficiently learn the representations and simultaneously smooth the unequally spaced and noisy functional data, even with limited training information.

3.5 Real Application

To further demonstrate the effectiveness of our method, in this section, we perform the proposed FAE, together with the conventional FPCA and the classic AE on the El Niño

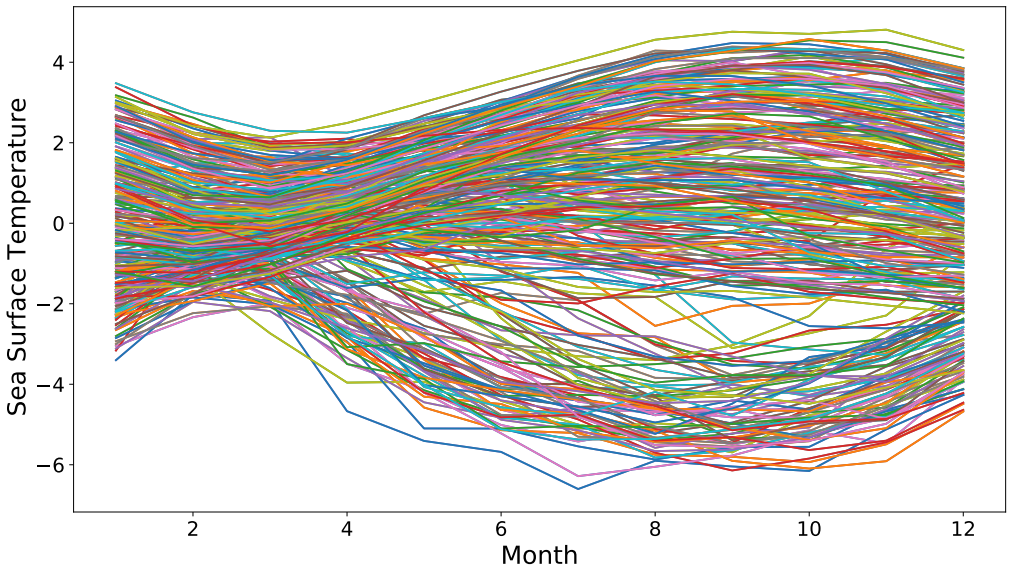

Figure 3.8: Centered monthly sea surface temperature measured in the Niño region defined by the coordinates 0-10 degree South and 90-80 degree West.

data set which is available in `R` package **rainbow** (Shang and Hyndman, 2019). This data set catalogs the monthly sea surface temperatures originally observed in 4 different locations from January 1950 to December 2006. The temperature curves were discretely measured at 12 evenly spaced time points over the entire interval for every year. We treat the measurements of each calendar year as an independent observation of the true underlying functional curve (varying with time), resulting in a total of 267 observations, and we label the 4 locations with numbers from 1 to 4 randomly. To avoid poor random initialization and obtain stable training process for the NN-based methods, we centre the data by subtracting the sample mean curve across all observations, and a visualization of the centered sea surface temperature curves is provided by Figure 3.8.

We continue to compare the proposed method with two benchmark models, FPCA and AE, on their performances in terms of curve reconstruction and representation extraction. We equip the classic AE and the proposed FAE with different combinations of hyperparameters for a linear mapping path and a potential nonlinear mapping pattern, while FPCA is performed with the focus on measuring the linear relationship. The hyperparameters for all models in comparison are tuned in advance to yield a fair improvement in their performances during the actual training. To reduce the computational cost of the tuning process, for each model, we fix the number of representations to be 5 and then perform a grid search over the hyperparameters of our interests with respect to the loss function by simply building a model for each possible combination of all of the hyperparameter values provided, and the optimal model architecture combination of hyperparameters identified by the grid search with 5 representations is further applied to model training with 3 and 8 representations. In Appendix B, Table B.5 provides a

summary of the candidate values considered in hyperparameter tuning for all models, while the details of the optimally identified configurations for models in comparison is narrated in Table B.6. We proceed with 20 repetitions of random subsampling validation: randomly dividing the data set into a training set and a test set, with 80% and 20% of the total samples assigned to them, respectively. We evaluate the prediction and classification performances of all the mentioned approaches on the 20 test sets using 3, 5 and 8 representations, respectively.

Table 3.6 summaries the performances of all the methods applied for different numbers of representation extracted on curve prediction and classification, using $\mathrm{MSE_p}$ and $\mathrm{P_{classification}}$ averaged over 20 random test sets. Apparently, the proposed FAEs consistently and comprehensively outperforms the FPCA and the AE models in both reconstruction and classification, by reaching the lowest prediction error and the highest classification accuracy for all representation attempts. With regard to the predictive performance, the linear FAE retains to be the top performer, closely followed by the nonlinear FAE. On the other hand, the nonlinear FAE continuously oversteps the other models in the competition of classifying curves, exhibiting its advantages in extracting more informative representations. To further confirm this in the context of statistical significance, we conduct two-sided paired t-tests to compare the $\mathrm{MSE_p}$ and $\mathrm{P_{classification}}$ of the 20 replicates of the nonlinear FAE to those of the FPCA, and the corresponding p-values are reported in Table B.7 in Appendix B. We observe that the nonlinear FAE remains superior to the FPCA regarding both evaluation metrics, especially when the representation size increases, demonstrating the importance and necessity of the proposed FAE in nonlinear representation learning.

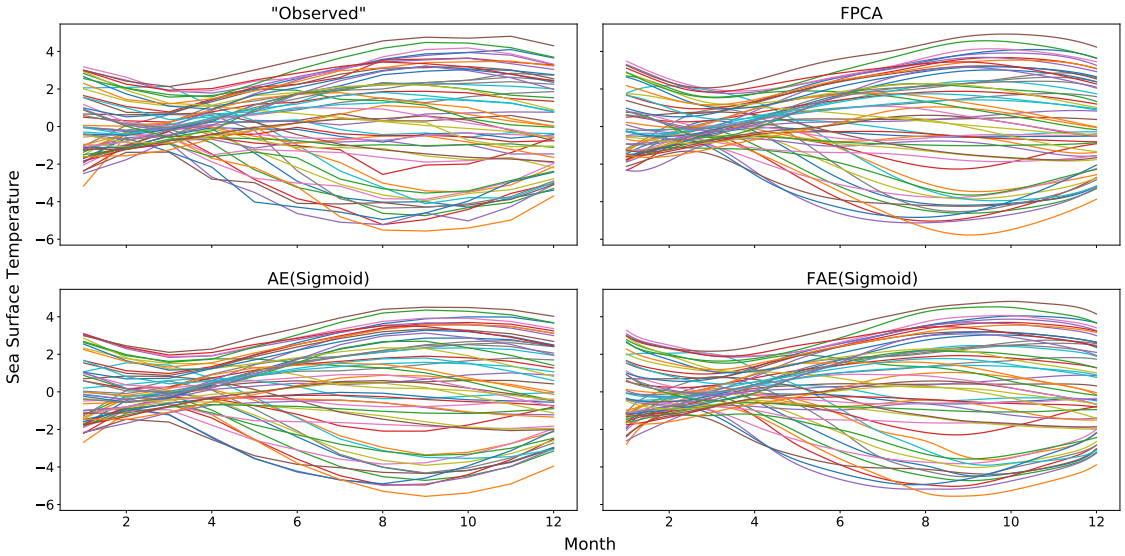

Figure 3.9: The observed curves and curves recovered by functional principal component analysis (FPCA), classic autoencoder with the sigmoid activation function (AE(Sigmoid)) and functional autoencoder with the sigmoid activation (FAE(Sigmoid)) using 5 representations for a test set of El Niño data set.

Table 3.6: Means and standard deviations (displayed inside parentheses) of prediction error and classification accuracy of functional autoencoder with the identity activation function (FAE(Identity)) and the sigmoid activation function (FAE(Sigmoid)), classic autoencoder with the identity activation function (AE(Identity)) and the sigmoid activation function (AE(Sigmoid)) and functional principal component analysis (FPCA) on 20 random test sets with the El Niño data set.

		FAE (Identity)	**FAE (Sigmoid)**	**AE (Identity)**	**AE (Sigmoid)**	**FPCA**
MSE_p	3 reps	0.0616(0.0051)	**0.0582**(0.0045)	0.0619(0.0051)	0.0715(0.0079)	0.0656(0.0054)
	5 reps	**0.0211**(0.0023)	0.0226(0.0031)	0.0261(0.0052)	0.0329(0.0042)	0.0242(0.0031)
	8 reps	**0.0062**(0.0009)	0.0089(0.0014)	0.0064(0.0008)	0.0071(0.0021)	0.0113(0.0013)
$P_{classification}$	3 reps	76.88%(4.01%)	**77.68%**(5.07%)	76.52%(3.67%)	77.14%(6.05%)	77.59%(4.81%)
	5 reps	85.18%(4.86%)	**86.52%**(4.46%)	84.20%(5.15%)	85.71%(3.48%)	84.38%(5.20%)
	8 reps	85.89%(4.58%)	**87.59%**(4.67%)	85.27%(3.91%)	85.80%(3.89%)	84.81%(4.50%)

The other highlight of the proposed FAE is its capability of smoothing the originally discrete data. As illustrated in Figure 3.9, the trajectories recovered by using FAE are smooth over the entire domain due to the fact that they are constructed as the linear combination of the neurons in the *coefficient layer* and the basis functions that can be evaluated at any point within the interval of observation. On the contrary, the classic AE can only achieve point-wise prediction at the timestamp with actual observations, resulting in visible discontinuity in the curve reconstruction. Meanwhile, FPCA can also produce smooth prediction but it usually requires the discrete observation to be firstly smoothed.

In addition, FAE surpasses AE by fast converging to a similarly low prediction error but with a resulting higher classification accuracy in both linear and nonlinear scenarios, as displayed in Figure 3.10 and Figure 3.11, demonstrating its high efficiency in extracting meaningful features and potential advantage in saving computational cost. It is noteworthy that the model configuration for the nonlinear AE is simpler than that of the nonlinear FAE, which brings benefits to the speed of nonlinear AE in training loss convergence.

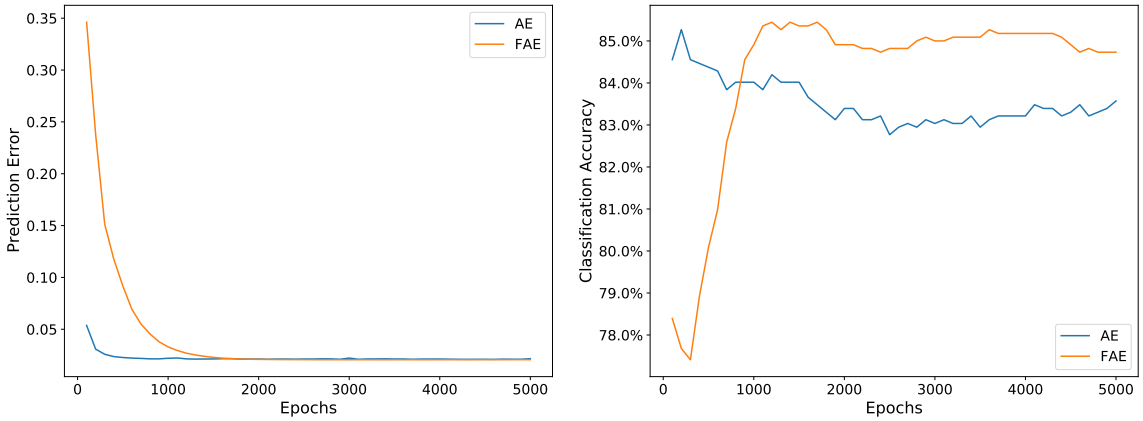

Figure 3.10: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the identity activation function using 5 representations on 20 random test sets of the El Niño data set change with the number of epochs.

The given results might imply that, for the El Niño data, the true relationship between the functional space to representation space for the sea temperature curves can be more accurately learned and revealed through a nonlinear mapping path, with the resulting nonlinear representations carrying more valuable information beneficial for further statistical analysis.

3.6 Conclusion

In this work, we introduced autoencoders with a new architecture design for discrete functional observations targeting at unsupervised representation learning and direct curve

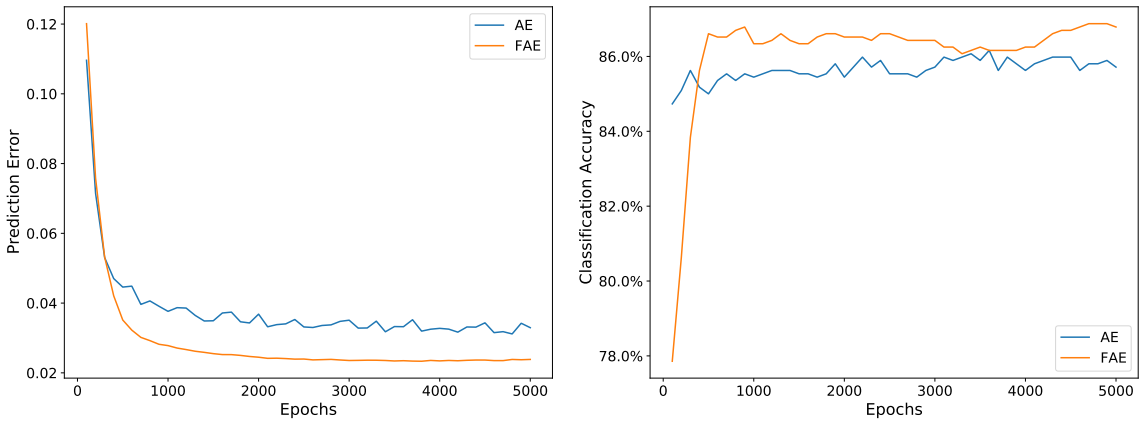

Figure 3.11: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the sigmoid activation function using 5 representations on 20 random test sets of the El Niño data set change with the number of epochs.

smoothing concurrently. The deterministic *feature layer* added to the encoder reduces the computational effort and enhances the model robustness in learning performance, while the additional *coefficient layer* similarly incorporated into the decoder ensures the usage of backpropagation in model training and allows the decompression from scalar neurons to functional curve in a linear manner. Additionally, we implemented our proposed FAE in a way that accommodates both regularly and irregularly functional data with flexible necessity on the size of training data for achieving satisfactory performance. Through several simulation studies and a real application, we demonstrated that the method we proposed is superior to the classic linear representation method for functional data, i.e., FPCA, under nonlinear scenarios and retains competitive performance in linear cases. Moreover, the numerical experiments endorse that our model can lead to an improvement over the classic AE in terms of prediction by generalizing and converging rapidly even with reduced training observations.

Nevertheless, the developed method depends on numerous hyperparameters, including the number of hidden layers, the number of neurons in each hidden layer, the training optimizer, etc., and unfortunately, conducting a grid search on that space can be time-consuming. It is necessary to bring up that the performance of the FAE varies from one hyperparameter configuration to another, and the randomness in initializing network parameters will introduce more variance to the results across training replicates. In contrast, the FPCA can be effortlessly fitted with only a few hyperparameters necessitating predetermination, but in sacrifice of the ability to accurately capturing the learning maps in nonlinear situations. Another weakness of our approach is its inability to process multidimensional functional data in its current form. Therefore, in a future work we could extend the current network architecture to a more dynamic architecture allowing discrete multivariate functional data.

This might be achieved by introducing micro-neural networks (Lin et al., 2014; Yao et al., 2021) to replace the neurons in the *feature layer* and the *coefficient layer*. Furthermore, an analogous architecture of our proposed FAE can be implemented to tackle nonlinear functional regression problems with both a functional predictor and a functional response.

Chapter 4

Simplified Survival Neural Network for Time-to-Event Prediction

Time-to-event is a common outcome measure in biomedical applications, with Cox regression (Cox, 1972) being the conventional approach for evaluating the relationship between features and this outcome. To address the computational challenge posed by the growth of modern data sets, various survival neural networks have been suggested as alternatives. Each network depends on a customized loss function and its corresponding training procedure, which could introduce inconsistency and bias. In this chapter, we present three simple approaches for transforming the time-to-event outcomes into a quantitative response vector. This transformed response can be readily utilized in any deep neural network without requiring the customization of the loss function to accommodate right censoring. We propose a novel simplified survival neural network to tackle the time-to-event prediction. It is essentially comprised of three steps: survival outcome transformation, feature extraction, and hazard model fitting. Our numerical studies show that the proposed framework maintains competitiveness with existing methods in terms of model discrimination under both the proportional hazards and nonproportional hazards settings, and it is demonstrated that our simplest loss function is the most computationally efficient when compared to the loss functions of other popular survival neural networks regardless of data size and batch size. The content showcased in this chapter is currently undergoing the journal review process.

4.1 Introduction

Time-to-event analysis has found wide application and gained significant attention in various health science domains. The Cox proportional hazards model (Cox regression) (Cox, 1972; Shen and Cook, 2013; Keogh and Morris, 2018), which assumes that the underlying hazard function is proportional over the follow-up time period across different covariates, has been predominately taken as the conventional approach to time-to-event prediction and the investigation of the relation between the covariates and survival outcome. Methods that

relax the proportional hazard assumption have also been well-developed in the literature (Schemper, 1992; Klein and Moeschberger, 2003; Borucka, 2014). However, with the advent use of big data, the computational capacity of the classic Cox regression for time-to-event prediction becomes insufficient.

Several existing studies have proposed survival neural networks, which integrate deep neural networks with the Cox proportional hazards model, focusing on the development of specialized matching loss functions. Faraggi and Simon (1995) first explored the extension of Cox regression with neural network by substituting the linear predictor of Cox with a one-hidden-layer neural network, while retaining the Newton-Raphson's method for model parameterization. Katzman et al. (2018) expanded on the work of Faraggi-Simon by incorporating modern deep learning techniques into the Cox proportional hazard loss function under the proportional assumption. Their method, known as DeepSurv, aims to minimize the averaged negative log partial likelihood with regularization. Kvamme et al. (2019) revisited these methods and developed nonproportional alternatives, Cox-MLP and Cox-Time, relying on an approximation to the negative loglikelihood loss function that can reduce the computational cost for batching survival data under nonlinearity and nonproportionality. Lee et al. (2018) distinguished their method, DeepHit, by using a deep neural network to model the distribution of the survival times directly under no assumptions about the underlying stochastic process.

The aforementioned approaches, however, primarily focus on constructing the neural networks directly using time-to-event outcomes, each with its distinctly customized loss function and training procedure to address right censoring. As we demonstrate in the subsequent sections of the chapter, each of the diverse loss functions proposed by the aforementioned methods carries unique interpretations and varying levels of computational complexity. In this study, we instead propose a simplified survival neural network that does not necessitate customization of the loss function or training approach to address time-to-event prediction. The core of the proposed method is to transform the time-to-event outcomes into a quantitative response vector that captures the essence of survival information such that the prediction problem becomes an ordinary regression that can be inputted into any deep neural network. We introduce and compare three such transformations in this chapter along with their theoretical justifications.

The proposed method is summarized in three steps shown in Figure 4.1. As illustrated, in *Step 1*, we implement a survival outcome transformation using the observed time and event indicator to obtain the transformed response Y free of right-censoring. Inspired by Beaulac et al. (2023), in the following *Step 2* aimed at feature extraction, we construct a densely feed-forward neural network with the P covariates $\{x_1, x_2, ..., x_P\}$ and the quantitative response Y serving as its input and output, respectively. We then take $\{Z_1, Z_2, ..., Z_{P'}\}$, the P' neurons of the last hidden layer of the trained neural network, as the features that capture the nonlinear relation among the original covariates on predicting the target Y. Finally, we

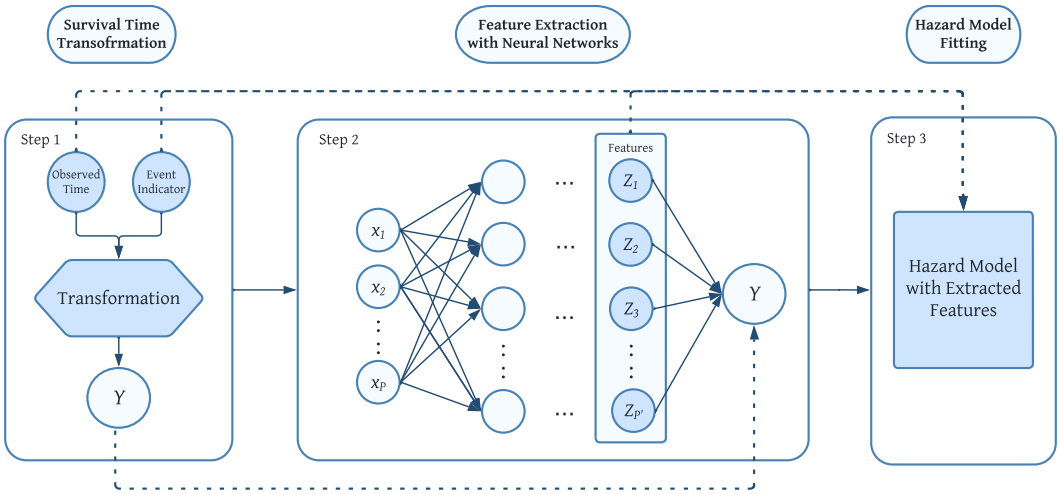

Figure 4.1: Diagram of the simplified survival neural network in the form of a 3-step pipeline, consisting of survival time transformation, feature extraction with neural networks and hazard model fitting.

fit a hazard model in *Step 3* using the extracted features $\{Z_1, Z_2, ..., Z_{P'}\}$, together with the originally observed time and event indicator, to achieve the goal of individualized survival prediction.

Our simplified survival neural networks have two major advantages. First, we reduce the complexity of survival neural networks to an ordinary neural network that does not require a tailored loss function and training procedure by proposing transformation of the time-to-event outcomes. Such transformation will not only simplify existing survival neural networks, but will also be directly applicable to various types of networks, such as convolutional neural networks (Zhu et al., 2016) and segmentation networks (Jha et al., 2020), among many others with application in survival data. Second, the proposed pipeline as illustrated in Figure 4.1 is computationally efficient. It accommodates ties in the time-to-event outcomes and both proportional and nonproportional hazards. We provide open source functions for implementation (`https://github.com/sidiwu/SSurvNN`) such that it can be used for future machine learning research with time-to-event prediction.

The remainder of this chapter proceeds as follows. In Section 4.2, we provide methodological details for time-to-event outcome transformation, feature extraction, and hazard model fitting. The finite sample performance is investigated in Section 4.3 via intensive simulation studies. We compare the proposed method to several existing survival neural networks in Section 4.4 under real data applications. Finally, we conclude with discussion and future directions in Section 4.5.

4.2 Methodology

In this section, we provide details of the proposed method that aligns with each of the steps presented in Figure 4.1, i.e., transformation, feature extraction, and hazard model fitting. Suppose that there are n independent individuals in the cohort. For individual i, we let $\boldsymbol{x}_i$ denote a vector of P covariates. We denote the observed time-to-event outcome with the pair (T_i, δ_i), where T_i is the minimum of event time T_i^* and censoring time C_i, and δ_i is the event indicator with $\delta_i = 1$ indicating that the observed T_i is the event time and $\delta_i = 0$ indicating censoring time.

4.2.1 Step 1: Time-to-Event Outcome Transformation under Right-Censoring

We first introduce three transformations to the time-to-event outcome and denote the transformed quantitative response by Y_i for individual i throughout the chapter.

Reweighing

The reweighing method is designed to transform T_i^* by applying a function $f(T_i^*)$ where $f : [0, \infty) \to \mathbb{R}$. A widely adopted choice of such function is the log transformation. Nevertheless, in survival study, it is quite common to obtain right-censoring individuals whose actual event times T^*'s are unobserved, and ignoring the censored observations would lead to biased estimates. To address this issue, one approach is to reweigh the observed survival times T_i's using a suitable weight that accounts for censoring by

$$Y_i = \frac{\delta_i f(T_i)}{\hat{S}^c(T_i^-)} , \tag{4.1}$$

where $\hat{S}^c(\cdot)$ represents the Kaplan-Meier survival function of the censoring random variable C, and T_i^- denotes the left limit of T_i. It can be shown that the expectation of Y_i conditional on the observed covariates is approximately equal to the expectation $f(T_i^*)$ conditional on the observed covariates (Koul et al., 1981).

Mean Imputation

As an alternative strategy to accommodate the presence of right-censoring, mean imputation transforms the time-to-event outcome T_i differently according to the corresponding δ_i for each cohort individual. Specifically, for an event individual i ($\delta_i = 1$), we directly set the response Y_i as $f(T_i)$, where f is again the function of log transformation; while for an censored subject i ($\delta_i = 0$) without the actual event observation, we substitute its unknown event time with the expected value that accounts for all event times larger than the censored time C_i, and the response becomes

$$Y_i = \frac{\sum_{T^*_{(j)} > C_i} f(T^*_j) \triangle \hat{S}(T^*_{(j)})}{\hat{S}(C_i)} \,, \tag{4.2}$$

where $T^*_{(1)} < T^*_{(2)} < \ldots < T^*_{(J)}$ are the J ordered distinct event times, and $\hat{S}(\cdot)$ and $\triangle \hat{S}(T^*_{(j)})$ denote the Kaplan-Meier survival function and the jump size of $\hat{S}(\cdot)$ at time $T^*_{(j)}$, respectively. Consequently, in this setting, the largest observation $T^*_{(J)}$ will be automatically regarded as the true event and simultaneously serves as the largest mass point of the estimated survival function of T^*. Under the self consistency property of the Kaplan-Meier estimator (Efron, 1967), the reweighing method can be mathematically proven to be equivalent to mean imputation; this has been discussed in detail in Datta (2005).

Deviance Residual

This last approach developed for time-to-event transformation entails the substitution of the survival endpoint with suitably chosen residuals. This design effortlessly inherits the simple algorithms applicable to continuous outcomes and masterly bypasses the challenges associated with right-censoring. Specifically, we first consider the null martingale residual for the i-th individual defined as:

$$G_i = \delta_i - H_0(t),$$

where $H_0(t) = \int_0^t h_0(t)dt$ denotes the cumulative hazard function at time t. One way to interpret G_i is as the discrepancy between the number of events that were observed and the number of events that were expected. However, the main limitation of the martingale residuals is its high skewness, taking values from $-\infty$ to 1, and therefore we opt to employ a common alternative named deviance residual to facilitate the transformation. Accordingly, the transformed time-to-event outcome is further denoted as:

$$Y_i = \text{sign}(\hat{G}_i)[2\{ - \hat{M}_i - \delta_i \log\Big(\delta_i - \hat{G}_i\Big)\}]^{1/2} \,. \tag{4.3}$$

As a normalized version of the martingale residual (Davison and Gigli, 1989), the deviance residual Y_i enables the expansion of G_i on its range from $[-\infty, 1]$ to a wider real line of $[-\infty, \infty]$ by utilizing a simple log transformation.

Figure 4.2 visualizes a comparison of the relationship between the observed censoring/event time T vs. transformed response Y under the three transformation methods using the Rot. & GBSG dataset (Kvamme et al., 2019) from the real application section. It shows that the reweighing method transforms all censoring observations to a response Y valued at 0 and the event observations to Y's that are positively correlated to the original event times. Likewise, the mean imputation approach log-transforms the event times, naturally giving rise to an increasingly concave-down shape curve that signifies a positive correlation

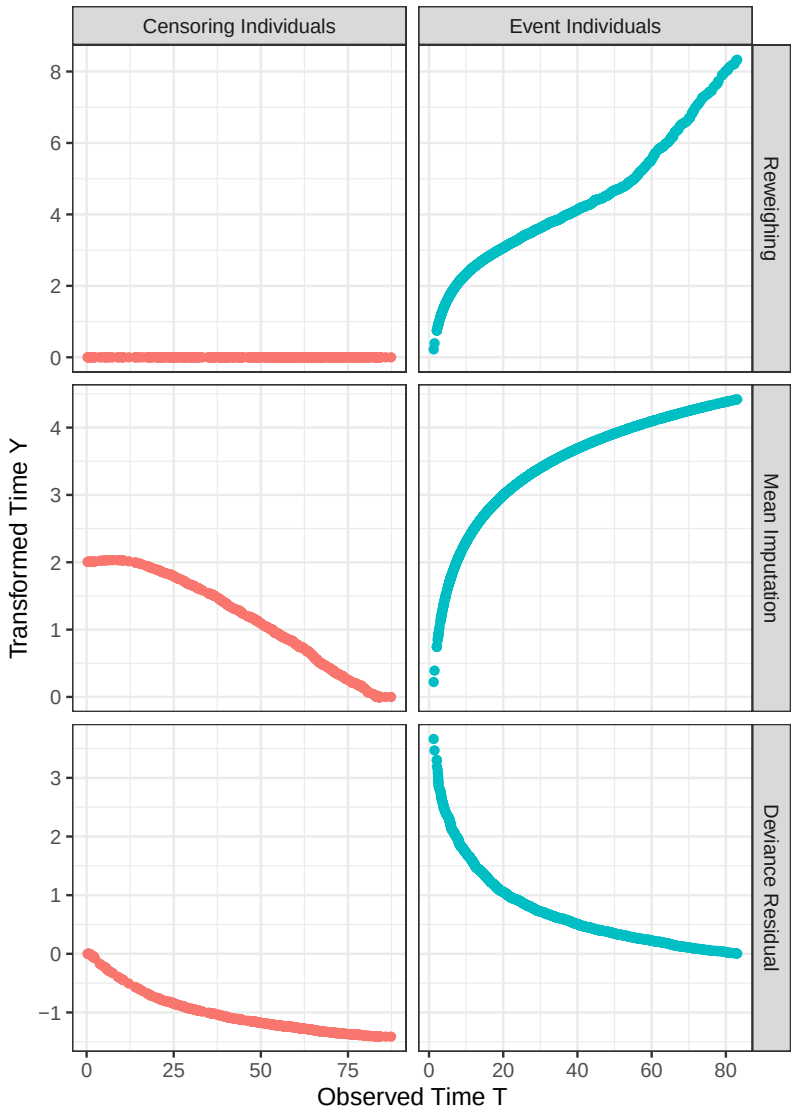

Figure 4.2: Transformed time-to-event response Y, obtained from reweighing, mean imputation, and deviance residual, vs. the observed time T for censoring and event individuals in the Rot. & GBSG dataset (Kvamme et al., 2019).

between the transformed Y's and the original event observations. On the other hand, employing mean imputation introduces variability in the relationship between the censoring observations and their corresponding transformed response across different cases. Different from the first two transformations, the deviance residual method produces transformed quantitative responses that are negatively correlated to the original survival observations for both event and censoring individuals. The figures that display the relationship between the observed censoring/event time and the response transformed by the three transformation approaches with the other data sets used in the real application are provided in Section C.4 of Appendix C.

4.2.2 Step 2: Feature Extraction with Neural Networks

In this study, we concentrate on employing densely feed-forward neural networks (NNs), which constitute the most prevalent architecture in neural network applications. Here, we define an NN composed of a P-dimensional input layer, L hidden layers, and the scalar output layer. While we define a densely feed-forward neural network in this subsection, we note that this can be flexibly adjusted to any neural network of interest with the transformed Y. In learning the potentially nonlinear mapping path from the input $\boldsymbol{x}$ to the output Y, we have

$$Y_i = \mathrm{NN}_\eta\left(\boldsymbol{x}_i\right) = a_{L+1}\left(\cdots a_1\left(\sum_{p=1}^{P} w_{1p}x_{ip} + b_1\right)\right), \tag{4.4}$$

where $a_1, ..., a_{L+1}$ are the activation functions connecting each two consecutive layers, and η is comprised of weights $\{w_{\ell p}\}_{\ell=1}^{L+1}$ and bias $\{b_\ell\}_{\ell=1}^{L+1}$ of all layers. We intuitively force the activation function a_{L+1} connecting the output layer to be the linear activation function and withdraw the neurons in the last hidden layer, denoted by $\boldsymbol{Z} = (Z_1, Z_2, ..., Z_{P'})$ as shown in Figure 4.1. These neuron are identified as the meaningful features that are inputted in the subsequent hazard function in *Step 3*, because they gradually capture and condense the nonlinear relationship among the covariates, effectively summarizing the valuable information carried by the input variables $\boldsymbol{x}$ and their association with the outcome Y. To ensure the identifiability of the survival model employed in the following step, we enforce the dimension of features, to be much less than the number of observations, i.e., $P' \ll n$ in practice.

It is worth pointing out that $Y_i = \mathrm{NN}_\eta(\boldsymbol{x}_i)$ is free of the restriction on training with respect to a specific loss function and can be fitted with mini-batch stochastic gradient descent (SGD), as the quantitative output Y is treated equivalently for both censored and event individuals.

Loss Function Comparison

The time-to-event transformation described in *Step 1* removes the requirement of a customized loss function for time-to-event outcome when constructing a survival neural network. In this section, we provide three such examples of how the customization of the loss function differs among various existing survival neural networks.

Assuming the mean squared error loss is used, the objective function of the neural network applied in *Step 2* of our pipeline is simply formulated as the mean squared difference between the predicted and actual network output,

$$L_{\mathrm{NN}}(\eta) = \frac{1}{n_{\mathrm{train}}} \sum_{i=1}^{n_{\mathrm{train}}} (Y_i - \hat{Y}_i)^2, \tag{4.5}$$

where n_{train} represents the number of training individuals in a single batch. Such a loss function is commonly used, computationally efficient and easily implemented in network training. Additionally, the neural network is configured to output the transformed response, an unbiased estimator of the original time-to-event outcome.

In comparison, the majority of the existing survival neural networks employ customized loss functions that can result in additional computational cost and inconsistent estimation. For instance, Kvamme et al. (2019) equipped their method DeepSurv with a loss function measuring the negative log partial likelihood with regularization,

$$L_{\text{DeepSurv}}(\eta) = \frac{1}{n_{\delta=1}} \sum_{i:\delta_i=1} \log \left(\sum_{j \in \mathcal{R}(T_i)} \exp[\hat{g}_\eta(\boldsymbol{x}_j) - \hat{g}_\eta(\boldsymbol{x}_i)] \right) + \lambda \cdot \|\eta\|_2^2 , \tag{4.6}$$

where $n_{\delta=1}$ denotes the number of events in the training set, $\mathcal{R}(T_i)$ is the risk set at time T_i, $\hat{g}_\eta(\boldsymbol{x}_i)$ is the log-rank function outputted by the network for the i-th subject, and λ acts as the regularization parameter. Considering the loss in Eq.(4.6) takes the sum over risk set $\mathcal{R}(T_i)$, which can be as large as the full training set (Katzman et al., 2018), the computation of DeepSurv necessitates the input of the entire data set for each training epoch, ballooning the computational cost of network training.

To tackle this limitation and enable the usage of mini-batch SGD, Katzman et al. (2018) incorporated the developed nonlinear and nonproportional survival neural networks, Cox-MLP and Cox-Time, with a $L_{\text{DeepSurv}}(\eta)$-approximated loss function,

$$L_{\text{Cox-MLP}}(\eta) = L_{\text{Cox-Time}}(\eta) = \frac{1}{n_{\text{train},\delta=1}} \sum_{i:\delta_i=1} \log \left(\sum_{j \in \tilde{\mathcal{R}}(T_i)} \exp[\hat{g}_\eta(\boldsymbol{x}_j) - \hat{g}_\eta(\boldsymbol{x}_i)] \right), \tag{4.7}$$

where $n_{\text{train},\delta=1}$ denotes the number of individuals with an observable event in a single training batch and $\tilde{\mathcal{R}}(T_i)$ is a reasonable approximation of the full sum over $\mathcal{R}(T_i)$ in Eq.(4.6). They also demonstrated that it is sufficient to sample only one individual j from the risk set. While this approximation is seemingly efficient, the definition of sampling only one individual is quiet arbitrary.

On the other hand, DeepHit (Lee et al., 2018) was designed to be trained by minimizing a total loss function, $L_{\text{DeepHit}}(\eta) = L_1 + L_2$, composed of two terms: the log-likelihood of the joint distribution of the first hitting time and event L_1, and a combination of cause-specific ranking loss function L_2 that adapts the idea of concordance (Harrell et al., 1982). In the cases with single event of interest, the two sub-loss functions L_1 and L_2 can be separately expressed as

$$L_1 = -\sum_{i=1}^{n_{\text{train}}} \left\{ \mathbb{1}(\delta_i = 1) \cdot \log\left(y_{i,T_i}\right) + \mathbb{1}(\delta_i = 0) \cdot \log\left(1 - \hat{F}(T_i|\boldsymbol{x}_i)\right) \right\}, \tag{4.8}$$

where $\mathbb{1}$ denotes an indicator function, y_{i,T_i} is the network-outputted probability $\hat{P}(T_i|\boldsymbol{x}_i)$ that the subject i will experience the event at time T_i, and $\hat{F}(T_i|\boldsymbol{x}_i) = \hat{P}(t \leq T_i|\boldsymbol{x}_i) = \sum_{T=0}^{T_i} \hat{P}(t = T|\boldsymbol{x}_i)$, and

$$L_2 = \alpha \cdot \sum_{i \neq j} A_{i,j} \cdot \zeta \left(\hat{F}(T_i|\boldsymbol{x}_i), \hat{F}(T_i|\boldsymbol{x}_j) \right), \tag{4.9}$$

where $A_{i,j} \triangleq \mathbb{1}(\delta_i = 1, T_i < T_j)$ is the indicator function to filter comparable pairs (i, j)'s, the coefficient α is chosen to trade off ranking loss, and $\zeta(\cdot, \cdot)$ is a convex loss function. Consequently, this design enforces the calculation of two loss components along with an additional step to search acceptable pairs, resulting in expensive computational cost in the training stage.

We demonstrate the trade-off of computational speed and prediction performance between these competing methods with the proposed method in Section 4.3.4.

4.2.3 Step 3: Individualized Survival Prediction

With the goal of predicting individualized survival, we focus on modelling the hazard function using the extracted features $\boldsymbol{Z}$ in *Step 2*,

$$h(t|\boldsymbol{Z}) = h_0(t)\exp\{\boldsymbol{\beta}^T \boldsymbol{Z}\}, \tag{4.10}$$

where $h_0(t)$ is the non-parametric baseline hazard function, and $\boldsymbol{\beta}$ is the vector of coefficients corresponding to the set of feature covariates $\boldsymbol{Z}$. Given Eq.(4.10), we can easily predict future survival for any new individual i' at some time t as

$$S(t|\boldsymbol{Z}) = S_0(t)^{\exp\left(\widehat{\beta}^T \boldsymbol{Z}_{i'}\right)}, \tag{4.11}$$

where $S_0(t) = \exp\left(-\int_0^t h_0(u)du\right)$ is the baseline survival function. Note that the proportional hazard assumption should be tested rigorously (Schoenfeld, 1982).

Violation of the Proportional Hazards Assumption

If the fitted Cox regression violates the proportional hazards assumption, we propose to introduce the time-dependent coefficient $\boldsymbol{\beta}(t)$ (Klein and Moeschberger, 2003), i.e.,

$$h(t|\boldsymbol{Z}) = h_0(t)\exp\{\boldsymbol{\beta}^T(t) \boldsymbol{Z}\}. \tag{4.12}$$

To practically implement the time-dependent coefficient modelling, we utilize the idea of time splitting (Therneau et al., 2017). Simply put, time splitting serves to divide individuals' survival records into multiple sub-records according to the given $M - 1$ cut points lying inside the observation interval $\mathcal{T}$, leading to M sub-intervals $\{\mathcal{T}^{(m)}\}_{m=1}^{M}$. The resulting data

Table 4.1: The original data set with two individuals (left), and the new data set after splitting the follow-up interval at time points 5 and 10, resulting in consecutive time intervals of length 5 (right).

ID	Duration	Event	x_1	x_2
1	9	0	-1.6	5.4
2	13	1	-0.6	3.6

$\Rightarrow$

ID	Start time	Stop time	Event	x_1	x_2
1	0	5	0	-1.6	5.4
1	5	9	0	-1.6	5.4
2	0	5	0	-0.6	3.6
2	5	10	0	-0.6	3.6
2	10	13	1	-0.6	3.6

set follows the 'counting process' format with the start and stop times of the sub-interval, the event status, as well as the values of the covariates (Cook and Lawless, 2018). The implementation of time splitting is detailed in Section C.1 of Appendix C and an example of time splitting for two individuals is illustrated in Table 4.1.

Handling Ties

It is common to encounter ties in the observed time in survival analysis and most neural networks do not correctly handle ties (Yang et al., 2022). Thus, it is worth mentioning that our proposed framework supports both Breslow and Efron methods for tied events (Hertz-Picciotto and Rockhill, 1997). The two options of handling ties are given as part of our pipeline and can be user-specified.

4.3 Simulation Studies

We aim to investigate the finite sample performance of the proposed method under 12 simulation settings. We denote the three transformations with RW, MI, and DR for reweighing, mean imputation and deviance residual, respectively. We let Cox and Cox(NP) refer to the proportional hazards model and the nonproportional hazards model with time-varying coefficient. Two sample sizes (n = 500 and 5000) as well as two censoring proportions (CR = 0.3 and 0.6) are also investigated in this simulation study.

The proposed method is readily implemented in `R` with self-defined functions for transformations, neural network constructed within **TensorFlow** (Allaire and Tang, 2020) and **Keras** (Allaire and Chollet, 2020), and hazard models using the **survival** (Therneau, 2023) package. All the simulations were conducted on the Compute Canada server.

We focus on the concordance index or C-index (Harrell et al., 1982), one of the most commonly used evaluation metrics in survival analysis, to examine the prediction accuracy of the proposed method. In the case of proportional hazards assumption being satisfied, the C-index can be estimated with the `concordance()` function within the **survival** package. When Cox regression is fitted with time-dependent coefficients, on the other hand, we utilize the time-dependent C-index (Antolini et al., 2005; Jiang et al., 2021) from M sub-intervals.

Specifically,

$$C_{\text{dynamic}} = \frac{\sum_{m=1}^{M}\sum_{i=1}^{n_m}\sum_{j=1;j\neq i}^{n_m} \text{conc}_{ij}^{(m)}}{\sum_{m=1}^{M}\sum_{i=1}^{n_m}\sum_{j=1;j\neq i}^{n_m} \text{comp}_{ij}^{(m)}}, \tag{4.13}$$

with

$$\begin{aligned}\text{comp}_{ij}^{(m)} &= I\{T_i^{(m)} < T_j^{(m)} \ \&\ \delta_i^{(m)} = 1\} \\ &\quad + I\{T_i^{(m)} = T_j^{(m)} \ \&\ \delta_i^{(m)} = 1 \ \&\ \delta_j^{(m)} = 0\}, \qquad (4.14)\\ \text{conc}_{ij}^{(m)} &= I\{\hat{S}(T_i^{(m)}|\boldsymbol{Z}_i) < \hat{S}(T_i^{(m)}|\boldsymbol{Z}_j)\} \cdot \text{comp}_{ij}^{(m)}, \qquad (4.15)\end{aligned}$$

where $T_i^{(m)}$ and $\delta_i^{(m)}$ denote the observed time and the event indicator for individual i at m-th sub-interval, and n_m is the number of records available in $\mathcal{T}^{(m)}$. For implementation of the dynamic C-index, we rely on the `concordance()` function in **survival** package on the dataset post time splitting as described in Section 4.2.3.

In each of the following scenarios, we randomly selected 50% of the individuals for training and the remaining 50% for testing. All experiments were repeated for 100 times.

4.3.1 Scenario 1: Proportional & Linear

We begin with a scenario where the underlying hazard model for data generation satisfies the proportional hazards and linear assumptions. We simulated five covariates where $x_1 \sim$ Bern(0.5), x_2, x_3 were assumed to follow Unif$[-1, 1]$, and x_4, x_5 were assumed to follow Unif$[-2, 1]$. The corresponding coefficients of $\boldsymbol{x}_i$ were randomly sampled from a uniform distribution over $[-1, 1]$. The details of all other parameters used for simulations in this scenario can be found in Table C.1 of Appendix C.

For all repetitions, we consistently used 10 features extracted from a two-hidden-layer (with 20 and 10 neurons in the first and second hidden layer, respectively) feed-forward neural network that were subsequently used in the Cox regression. Table C.4 of Appendix C describes the hyperparameters involved in the neural network as well as the details for fitting nonproportional hazard models.

Table 4.2 displays the means and standard deviations (SDs) of the C-indices on the testing sets averaged over 100 replicates for difference methods. We can see that under this subsection, the classic Cox retains the best discriminatory performance in all the simulations. This is as expected as the classic Cox is the true model for data generation. Meanwhile, we observe that the discriminatory performances using the proposed NN-Cox and NN-Cox(NP) are both competitive to that of the classic Cox model, especially when the dataset size increases to 5000. Under this simulation study, we see that using DR transformation gives the highest C-index. However, the difference in C-index brought by the three transformation

Table 4.2: Means and standard deviations (in parentheses) of C-indices over the 100 replicates for different methods, including classic cox proportional hazard model (Classic Cox (Linear)), and our proposed simplified neural networks using reweighing (RW), mean imputation (MI) or deviance residual (DR) in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and either classic Cox regression (Cox) or Cox regression with time-dependent coefficient (Cox(NP)) in *Step 3*, on simulated data sets with different sample size (n) and censoring rate (CR) and satisfying the proportional assumption of Cox model.

	n = 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Classic Cox (Linear)	0.733 (0.019)	0.746 (0.021)	0.757 (0.005)	0.774 (0.005)
RW-NN-Cox	0.728 (0.018)	0.740 (0.022)	0.756 (0.005)	0.773 (0.005)
MI-NN-Cox	0.729 (0.018)	0.740 (0.023)	0.756 (0.005)	0.774 (0.005)
DR-NN-Cox	0.729 (0.019)	0.742 (0.022)	0.756 (0.005)	0.774 (0.005)
RW-NN-Cox(NP)	0.728 (0.019)	0.736 (0.023)	0.756 (0.005)	0.773 (0.005)
MI-NN-Cox(NP)	0.728 (0.018)	0.736 (0.023)	0.756 (0.005)	0.774 (0.005)
DR-NN-Cox(NP)	0.728 (0.019)	0.738 (0.022)	0.756 (0.005)	0.774 (0.005)

methods are minimal, indicating that all three approaches can efficiently capture the survival information.

4.3.2 Scenario 2: Proportional & Nonlinear

Next, we investigate the finite sample performance of a more realistic scenario where the data sets were simulated free of the linearity assumption. The simulation details are described in Section C.2.2 of Appendix C.

Table 4.3 summarizes the results under all 4 settings in this scenario. As expected, we see that the classic Cox generates the lowest C-index due to the violation of the linearity assumption. It is not surprising to see that our proposed method is significantly superior to the classic Cox under all settings, because the neural network in the second step of the pipeline does not rely on the linearity assumption. The DR transformation method continues to show its advantages by giving the highest C-index when the data size increases to 5000 while the MI transformation seems to benefit the most when the data size is relatively small.

4.3.3 Scenario 3: Nonproportional & Nonlinear

Lastly, we are interested in investigating how well the proposed method performs when both the proportional hazards and linearity assumptions are violated. We provide the simulation details for this scenario in Section C.2.3 of Appendix C.

We illustrate the results in Table 4.4. As expected, we see that the classic Cox generates the lowest C-index due to the violation of proportional hazards and linearity assumptions. We can see that the NN-Cox(NP) retains superior discriminatory performance in comparsion

Table 4.3: Means and standard deviations (in parentheses) of C-indices over the 100 replicates for different methods, including classic cox proportional hazard model (Classic Cox (Linear)), and our proposed simplified neural networks using reweighing (RW), mean imputation (MI) or deviance residual (DR) in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and either classic Cox regression (Cox) or Cox regression with time-dependent coefficient (Cox(NP)) in *Step 3*, on simulated data sets with different sample size (n) and censoring rate (CR), satisfying the proportional but violating the linear assumption of Cox model.

	n = 500		**n = 5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Classic Cox (Linear)	0.692 (0.018)	0.727 (0.023)	0.719 (0.005)	0.767 (0.006)
RW-NN-Cox	0.734 (0.028)	0.750 (0.033)	0.772 (0.014)	0.819 (0.014)
MI-NN-Cox	0.739 (0.030)	0.773 (0.037)	0.761 (0.022)	0.809 (0.018)
DR-NN-Cox	0.720 (0.029)	0.761 (0.038)	0.785 (0.010)	0.826 (0.012)
RW-NN-Cox(NP)	0.735 (0.029)	0.749 (0.032)	0.772 (0.014)	0.819 (0.014)
MI-NN-Cox(NP)	0.739 (0.031)	0.772 (0.038)	0.763 (0.021)	0.809 (0.017)
DR-NN-Cox(NP)	0.721 (0.029)	0.760 (0.038)	0.785 (0.010)	0.827 (0.012)

to the other approaches. Similarly to Scenario 2, we observe that the RE and MI lead to slightly better results when the sample size is small (500), while the DR transformation demonstrates more advantages when the size of the data increases to 5000. Additionally, our pipeline is quiet robust and was not effected by a larger censoring rate as demonstrated under all simulation scenarios.

4.3.4 Computation Speed with Different Loss Functions

Comparisons of computational costs among the aforementioned loss functions in Section 4.2.2 on training sets with different sample sizes using mini-batch SGD are demonstrated in this subsection. Because the training procedure of DeepSurv requires the input of the entire training set, we compute its corresponding loss function using the full training data instead of reduced training subjects in batches. We perform the comparison on the same programming platform `Python` (Van Rossum and Drake, 2009) and apply the mean squared loss function directly accessible in **PyTorch** (Paszke et al., 2019) for the proposed simplified survival neural network and the loss functions in default settings provided by the `Python` library **pycox** (Kvamme et al., 2019; Antolini et al., 2005; Katzman et al., 2018; Lee et al., 2018; Fotso, 2018; Graf et al., 1999) for the remaining methods. Specifically, we set DeepHit to the single-event mode and let 10 be the size of the equidistant discretization grid of the continuous event times for its output layer.

Table 4.5 illustrates the result of the comparison among four loss functions on their computational costs for running 10000 replicates of data sets with different sample sizes and around 30% of censoring subjects. It is not surprising to observe that our simplest loss

Table 4.4: Means and standard deviations (in parentheses) of C-indices over the 100 replicates for different methods, including classic cox proportional hazard model (Classic Cox (Linear)), and our proposed simplified neural networks using reweighing (RW), mean imputation (MI) or deviance residual (DR) in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and either classic Cox regression (Cox) or Cox regression with time-dependent coefficient (Cox(NP)) in *Step 3*, on simulated data sets with different sample size (n) and censoring rate (CR) and violating the proportional and linear assumptions of Cox model.

	n = 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Classic Cox (Linear)	0.624 (0.024)	0.683 (0.031)	0.658 (0.009)	0.659 (0.013)
RW-NN-Cox	0.686 (0.042)	0.734 (0.043)	0.702 (0.031)	0.719 (0.042)
MI-NN-Cox	0.692 (0.040)	0.745 (0.045)	0.729 (0.037)	0.752 (0.044)
DR-NN-Cox	0.666 (0.042)	0.728 (0.040)	0.731 (0.037)	0.766 (0.042)
RW-NN-Cox(NP)	0.703 (0.037)	0.746 (0.039)	0.725 (0.028)	0.7467 (0.035)
MI-NN-Cox(NP)	0.709 (0.037)	0.754 (0.042)	0.750 (0.031)	0.776 (0.036)
DR-NN-Cox(NP)	0.685 (0.038)	0.743 (0.035)	0.746 (0.032)	0.789 (0.035)

function $L_{\text{NN}}(\eta)$ outperforms the others in terms of computational efficiency for all scenarios. Specifically, the execution time of our loss function experiences only a slight increase as the data size grows, whereas the computational costs of the other losses rise dramatically with the size of the data. On the other hand, in an individual mini-batch step, with a decrease in the batch size, the time for running a single batch declines but the corresponding batch number climbs, generally resulting in a growing cost in the evaluation of the loss function in one full epoch. However, in practice, mini-batch SGD tends to progress much faster in terms of convergence to SGD (in other words, mini-SGD usually need less number of epochs to reach a stable solution), especially when the network is initialized far from the point of convergence (Bottou, 2010; Bertsekas, 2015).

4.4 Real Applications

We further compare our proposed method with existing methods in three real world data sets. The data sets include the Molecular Taxonomy of Breast Cancer International Consortium (METABRIC), the Rotterdam tumor bank and German Breast Cancer Study Group (Rot. & GBSG) and the Assay of Serum Free Light Chain (FLCHAIN). These data sets are previously processed and publicly available in the `Python` module **pycox**. Among the existing survival neural networks, we considered two nonlinear models (DeepSurv and Cox-MLP (Kvamme et al., 2019)), together with two models that can handle nonproportionality (Cox-Time (Kvamme et al., 2019) and DeepHit). All methods are compared with the classic Cox proportional hazards model to achieve a benchmark.

Table 4.5: Computational times (in seconds) of the loss function of DeepSurv, the loss function of Cox-MLP/Cox-Time with one control case, the loss function of DeepHit with single event of interest, and the loss function of the neural network of the proposed method for running 10,000 replicates of data sets with training size n_{train} and batch size s_{batch}.

n_{train}	s_{batch}	$L_{\text{DeepSurv}}(\eta)$	$L_{\text{Cox-MLP}}(\eta)/L_{\text{Cox-Time}}(\eta)$	$L_{\text{DeepHit}}(\eta)$	$L_{\text{NN}}(\eta)$
1,000	10	-	33.65	568.81	**30.26**
	100	-	3.63	162.64	**3.08**
	1,000	3.12	0.75	>1000	**0.42**
10,000	100	-	41.13	>1000	**32.31**
	1,000	-	6.70	>1000	**3.83**
	10,000	20.03	4.07	>1000	**1.04**
100,000	1,000	-	96.13	>1000	**33.85**
	10,000	-	40.54	>1000	**9.43**
	100,000	200.69	17.08	>1000	**3.80**

Note: DeepSurv necessitates the usage of the complete training set during the training stage and therefore mini-batch Stochastic Gradient Descent is not applicable to it.

For the proposed framework, all three transformations are used in this analysis. In the feature-extraction step involving neural networks, we apply a densely feed-forward neural network with two hidden layers and a nonlinear activation function. The network is trained using the Adam optimizer with the default learning rate and weight decay setting. No batch normalization and early stopping are used. Table C.6 of Appendix C catalogs the hyperparameters of the neural networks along with the time splitting interval designed for the nonproportional hazard model.

All reported C-indices in this section are averaged over a 5-fold cross-validation. It is clearly shown in Table 4.6 that our proposed approach consistently outperforms the Classic Cox in discrimination under various settings, particularly when the data set fails to satisfy the assumptions of Cox hazards model. From Table 4.7, we can see that our proposed NN-Cox retains similar discriminatory performance with those of nonlinear approaches, including DeepSurv and Cox-MLP. Meanwhile, Tabel 4.8 illustrates that our proposed NN-Cox(NP) also yields similar prediction performance to those of the competing non-proportional models Cox-Time and DeepHit. Thus, our proposed method is competitive to many existing survival neural networks with more convenience in computation and less complexity in interpretation. In addition, we observe that the NN-Cox(NP) generally oversteps NN-Cox, which agrees with the conclusions made by Kvamme et al. (Kvamme et al., 2019). Additionally, we see minimal differences among the three transformation methods with the DR transformation having a slightly better predictive performance.

Table 4.6: C-indices, averaged over the five cross-validation folds, for Classic Cox and our proposed simplified neural networks using RW, MI or RD in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and either Cox or Cox(NP) in *Step 3* on three common survival datasets.

Method	METABRIC	Rot. & GBSG	FLCHAIN
Classic Cox	0.628	0.666	0.790
RW-NN-Cox	0.637	0.669	0.788
MI-NN-Cox	0.635	0.669	0.791
DR-NN-Cox	0.632	0.673	0.792
RW-NN-Cox(NP)	0.661	0.673	0.789
MI-NN-Cox(NP)	0.659	0.669	0.791
DR-NN-Cox(NP)	0.660	0.676	0.792

Table 4.7: C-indices, averaged over the five cross-validation folds, for two existing nonlinear models DeepSurv and Cox-MLP, as well as our proposed simplified neural networks for nonlinear scenarios with RW, MI or RD in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and Cox in *Step 3* on three common survival datasets.

Method	METABRIC	Rot. & GBSG	FLCHAIN
DeepSurv	0.636	0.674	0.790
Cox-MLP	0.643	0.669	0.793
RW-NN-Cox	0.637	0.669	0.788
MI-NN-Cox	0.635	0.669	0.791
DR-NN-Cox	0.632	0.673	0.792

Table 4.8: C-indices, averaged over the five cross-validation folds, for two existing nonlinear and nonproportional methods, including Cox-Time and DeepHit, as well as our proposed simplified neural networks for nonproportional scenarios with RW, MI or RD in *Step 1*, a feed-forward neural network (NN) in *Step 2*, and Cox(NP) in *Step 3* on three common survival datasets.

Method	METABRIC	Rot. & GBSG	FLCHAIN
Cox-Time	0.662	0.677	0.790
DeepHit	0.675	0.670	0.792
RW-NN-Cox(NP)	0.661	0.673	0.789
MI-NN-Cox(NP)	0.659	0.669	0.791
DR-NN-Cox(NP)	0.660	0.676	0.792

4.5 Conclusion

In this chapter, we introduced a simplified survival neural network based on transformed time-to-event outcomes that does not necessitate customization of loss function and training procedure. The proposed transformations enable the utilization of any existing neural networks without modifying loss functions to accommodate right censoring, eliminating inconsistency and bias in estimation. Through numerical experiments, we observed the computational efficiency of the proposed method in comparison to competing methods. We demonstrated that the proposed method is robust against violation of the linearity and proportional hazards assumptions, and is competitive in model discrimination to several existing survival neural networks. Comprised of three easily implementable steps, the proposed approach is computationally efficient and flexible. By transforming the time-to-event outcomes into quantitative responses, our proposed framework enables the conversion of survival neural networks into an ordinary regression framework. This, in turn, facilitates the utilization of various neural network models and further allows the inclusion of high-dimensional data, e.g., imaging data, for time-to-event prediction.

Chapter 5

Conclusion and Future Work

This thesis is dedicated to the integration of machine learning techniques into statistical analysis, developing novel methodologies that utilize neural networks to address statistical challenges associated with functional data and survival data.

Chapter 2 introduces new approaches to tackle regression problem with scalar covariates and a functional response. Specifically, we designed and implemented a densely feed-forward neural network to output either basis coefficients or FPC scores, the two common representations of functions that are further used to construct the predicted functional response. To enable training the neural network directly with the functional response, we proposed to modify the objective function through a multiplication operation to bypass the initial estimation of the scalar representations. Such modifications are readily applicable to the output layer of various types of networks to produce a functional response, promoting the future research in utilization of various machine learning techniques in a functional regression framework. The developed models can accommodate both regularly and irregularly spaced functional data, and additionally, a roughness penalty can be incorporated into the objective function to enforce smoothness regularization of the predicted curves. To the best of our knowledge, this is the first endeavor to model function-on-scalar regression using artificial neural networks. Numerical results have demonstrated that our methods outperform the conventional function-on-scalar regression model in capturing relation and predicting curve, particularly under nonlinear settings. We are interested in establishing a comprehensive and versatile framework for addressing nonlinear regression problems across various types of functional data. This framework will include considerations for locally sparse regularization, enhancing its adaptability to different data scenarios.

We presented a novel neural network-based solution, named functional autoencoder (FAE), aiming at representation learning of functional data in Chapter 3. The proposed FAE features a distinctive architecture comprising customized input and output layers. These layers are designed to compress discrete functional observations into either linear or nonlinear representation while concurrently generating smooth reconstruction of the functional data. Through simulation studies and a real application, we found that the proposed FAE compares

favorably with FPCA and the conventional autoencoder in different aspects. It achieves satisfactory performance in prediction and classification when handling both regularly and irregularly spaced functional data. Employing the designed architecture to address functional regression issues is valuable, allowing for the exploration of nonlinear associations between a functional covariate and a functional response.

The last component of this thesis discusses time-to-event prediction with neural networks in a simplified manner. The proposed method reduces the complexity of survival neural networks to an ordinary neural network by first transforming the time-to-event outcomes under right censoring into a quantitative response vector that serves as the output of a neural network for feature extraction. The extracted features contain the survival information and act as the substitutes for the original covariates in a following hazard model fitting process to achieve individualized survival prediction. Different from most survival neural networks, our strategy eliminates the requirement of a customized loss function or training procedure for time-to-event outcome when constructing a survival neural network. Simulation studies show that our simplified survival neural network outperforms Cox regression in addressing concerns with nonlinearity and nonproportionality. Furthermore, compared with several existing survival neural networks in three real data sets, the developed approach achieves competitive performance in model discrimination with significant advantages in computational efficiency and model interpretation. The flexibility of employing various types of neural networks for feature extraction in the proposed pipeline provides a potential solution to predicting survival duration of individuals using not only scalar biomarkers but imaging diagnosis.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Allaire, J. and Chollet, F. (2020). *keras: R Interface to 'Keras'*. R package version 2.3.0.0.

Allaire, J. and Tang, Y. (2020). *tensorflow: R Interface to 'TensorFlow'*. R package version 2.2.0.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition.

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

Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58.

Barber, R. F., Reimherr, M., and Schill, T. (2017). The function-on-scalar lasso with applications to longitudinal gwas. *Electronic Journal of Statistics*, 11(1):1351–1389.

Beaulac, C., Rosenthal, J. S., and Hodgson, D. (2018). A deep latent-variable model application to select treatment intensity in survival analysis. *Proceedings of the Machine Learning for Health (ML4H) Workshop at NeurIPS 2018*.

Beaulac, C., Wu, S., Gibson, E., Miranda, M. F., Cao, J., Rocha, L., Beg, M. F., and Nathoo, F. S. (2023). Neuroimaging feature extraction using a neural network classifier for imaging genetics. *BMC Bioinformatics*, 24(1):271.

Bengio, Y., Courville, A. C., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bertsekas, D. P. (2015). Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *arXiv preprint arXiv:1507.01030*.

Borucka, J. (2014). Extensions of Cox model for non-proportional hazards purpose. *Ekonometria*, 3(45):85–101.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294.

Cai, X., Xue, L., and Cao, J. (2021a). Robust penalized m-estimation for function-on-function linear regression. *Stat*, 10(1):e390.

Cai, X., Xue, L., and Cao, J. (2021b). Variable selection for multiple function-on-function linear regressionl. *Statistica Sinica*, 32(4):1–43.

Cai, X., Xue, L., and Cao, J. (2022). Robust estimation and variable selection for function-on-scalar regression. *Canadian Journal of Statistics*, 50(1):162–179.

Carroll, C., Gajardo, A., Chen, Y., Dai, X., Fan, J., Hadjipantelis, P. Z., Han, K., Ji, H., Mueller, H.-G., and Wang, J.-L. (2020). *fdapace: Functional Data Analysis and Empirical Dynamics*. R package version 0.5.5.

Chen, D. and Müller, H.-G. (2012). Nonlinear Manifold Representations for Functional Data. *The Annals of Statistics*, 40(1):1–29.

Chen, K. and Lei, J. (2015). Localized functional principal component analysis. *Journal of the American Statistical Association*, 110(511):1266–1275.

Chen, L.-H. and Jiang, C.-R. (2016). Multi-dimensional functional principal component analysis. *Statistics and Computing*, 27(5):1181–1192.

Chiou, J.-M. and Li, P.-L. (2007). Functional clustering and identifying substructures of longitudinal data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(4):679–699.

Chiou, J.-M., Müller, H.-G., and Wang, J.-L. (2004). Functional response models. *Statistica Sinica*, pages 675–693.

Chiou, J.-M., Müller, H.-G., and Wang, J.-L. (2003). Functional quasi-likelihood regression models with smooth random effects. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(2):405–423.

Cook, R. J. and Lawless, J. F. (2018). *Multistate Models for the Analysis of Life History Data*. CRC Press.

Coppersmith, D. and Winograd, S. (1987). Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of Computing*, pages 1–6.

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

Datta, S. (2005). Estimating the mean life time using right censored data. *Statistical Methodology*, 2(1):65–69.

Dauxois, J., Pousse, A., and Romain, Y. (1982). Asymptotic theory for the principal component analysis of a vector random function: Some applications to statistical inference. *Journal of Multivariate Analysis*, 12(1):136–154.

Davison, A. C. and Gigli, A. (1989). Deviance residuals and normal scores plots. *Biometrika*, 76(2):211–221.

de Boor, C. (1978). *A Practical Guide to Splines*. Springer.

Dierckx, P. (1984). Computation of least-squares spline approximations to data over incomplete grids. *Computers & Mathematics with Applications*, 10(3):283–289.

Efron, B. (1967). The two sample problem with censored data. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 4, pages 831–853.

Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2):89–121.

Fabian Scheipl, A.-M. S. and Greven, S. (2015). Functional additive mixed models. *Journal of Computational and Graphical Statistics*, 24(2):477–501.

Fabian Scheipl, J. G. and Greven, S. (2016). Generalized functional additive mixed models. *Electronic Journal of Statistics*, 10(1):1455–1492.

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

Ferraty, F. and Vieu, P. (2006). *Nonparametric Functional Data Analysis: Theory and Practice*. Springer.

Fotso, S. (2018). Deep neural networks for survival analysis based on a multi-task framework. *ArXiv Preprint arXiv:1801.05512v1*.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471.

Graf, E., Schmoor, C., Sauerbrei, W., and Schumacher, M. (1999). Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17–18):2529–2545.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

Hall, P. and Hosseini-Nasab, M. (2006). On properties of functional principal components analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(1):109–126.

Harrell, Frank E., J., Califf, R. M., Pryor, D. B., Lee, K. L., and Rosati, R. A. (1982). Evaluating the yield of medical tests. *JAMA*, 247(18):2543–2546.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer.

Hayes, J. G. and Halliday, J. (1974). The least-squares fitting of cubic spline surfaces to general data sets. *IMA Journal of Applied Mathematics*, 14(1):89–103.

Hertz-Picciotto, I. and Rockhill, B. (1997). Validity and efficiency of approximation methods for tied survival times in Cox regression. *Biometrics*, 53(3):1151–1156.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hsieh, T.-Y., Sun, Y., Wang, S., and Honavar, V. (2021). Functional autoencoders for functional data representation learning. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 666–674.

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

Ivanescu, A. (2013). A note on bivariate smoothing for two-dimensional functional data. *International Journal of Statistics and Probability*, 2(2).

Jha, D., Riegler, M. A., Johansen, D., Halvorsen, P., and Johansen, H. D. (2020). Doubleu-net: A deep convolutional neural network for medical image segmentation. In *2020 IEEE 33rd International symposium on computer-based medical systems (CBMS)*, pages 558–564.

Jiang, S., Xie, Y., and Colditz, G. A. (2021). Functional ensemble survival tree: Dynamic prediction of alzheimer's disease progression accommodating multiple time-varying covariates. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 70(1):66–79.

Katzman, J. L., Shaham, U., Cloninger, A., Bates, J., Jiang, T., and Kluger, Y. (2018). Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1):1–12.

Keogh, R. H. and Morris, T. P. (2018). Multiple imputation in Cox regression when there are time-varying effects of covariates. *Statistics in Medicine*, 37(25):3661–3678.

Klein, J. P. and Moeschberger, M. L. (2003). *Survival Analysis: Techniques for Censored and Truncated Data*, volume 1230. Springer.

Koul, H., Susarla, V., and Van Ryzin, J. (1981). Regression analysis with randomly right-censored data. *The Annals of Statistics*, 9(6):1276–1288.

Kvamme, H., Borgan, Ø., and Scheel, I. (2019). Time-to-event prediction with neural networks and Cox regression. *arXiv preprint arXiv:1907.00825*.

Lecca, P. (2021). Machine learning for causal inference in biological networks: perspectives of this challenge. *Frontiers in Bioinformatics*, page 45.

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

Lin, M., Chen, Q., and Yan, S. (2014). Network in network. *arXiv preprint arXiv:1312.4400*.

Lin, Z., Cao, J., Wang, L., and Wang, H. (2017). Locally sparse estimator for functional linear regression models. *Journal of Computational and Graphical Statistics*, 26(2):306–318.

Meiler, J., Müller, M., Zeidler, A., and Schmäschke, F. (2001). Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Molecular Modeling Annual*, 7(9):360–369.

Morris, J. S. (2015). Functional regression. *Annual Review of Statistics and Its Application*, 2(1):321–359.

Müller, H.-g. (2005). Functional modelling and classification of longitudinal data. *Scandinavian Journal of Statistics*, 32(2):223–240.

Müller, H.-G. and Stadtmüller, U. (2005). Generalized Functional Linear Models. *The Annals of Statistics*, 33(2):774–805.

Müller, H.-G. and Yao, F. (2008). Functional additive models. *Journal of the American Statistical Association*, 103(484):1534–1544.

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273.

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks*, 5(6):927–935.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035.

Peng, J. and Müller, H.-G. (2008). Distance-based clustering of sparsely observed stochastic processes, with applications to online auctions. *The Annals of Applied Statistics*, 2(3):1056–1077.

Peng, J. and Paul, D. (2009). A geometric approach to maximum likelihood estimation of the functional principal components from sparse longitudinal data. *Journal of Computational and Graphical Statistics*, 18(4):995–1015.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ramos-Carreño, C., Torrecilla, J. L., Hong, Y., and Suárez, A. (2022). scikit-fda: Computational tools for machine learning with functional data. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 213–218.

Ramsay, J. O., Graves, S., and Hooker, G. (2020). *fda: Functional Data Analysis*. R package version 5.1.5.1.

Ramsay, J. O. and Silverman, B. W. (2005). *Functional Data Analysis (Second Edition)*. Springer.

Rao, A. R., Wang, Q., Wang, H., Khorasgani, H., and Gupta, C. (2020). Spatio-temporal functional neural networks. *arXiv preprint arXiv:2009.05665*.

Rossi, F. and Conan-Guez, B. (2005). Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60.

Rossi, F., Conan-Guez, B., and Fleuret, F. (2002). Functional data analysis with multi layer perceptrons. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, volume 3, pages 2843–2848.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sang, P., Wang, L., and Cao, J. (2017). Parametric functional principal component analysis. *Biometrics*, 73(3):802–810.

Schemper, M. (1992). Cox analysis of survival data with non-proportional hazard functions. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 41(4):455–465.

Schoenfeld, D. (1982). Partial residuals for the proportional hazards regression model. *Biometrika*, 69(1):239–241.

Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., and Bengio, Y. (2021). Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634.

Shang, H. and Hyndman, R. (2019). *rainbow: Bagplots, Boxplots and Rainbow Plots for Functional Data*. R package version 3.6.

Shen, H. and Cook, R. J. (2013). Regression with incomplete covariates and left-truncated time-to-event data. *Statistics in Medicine*, 32(6):1004–1015.

Song, J. and Li, B. (2021). Nonlinear and additive principal component analysis for functional data. *Journal of Multivariate Analysis*, 181:104675.

Strassen, V. et al. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356.

Suchit Mehrotra, A. M. (2022). Simultaneous variable selection, clustering, and smoothing in function-on-scalar regression. *Canadian Journal of Statistics*, 50(1):180–199.

Therneau, T. (2023). *A Package for Survival Analysis in R*. R package version 3.5-3.

Therneau, T., Crowson, C., and Atkinson, E. (2017). Using time dependent covariates and time dependent coefficients in the Cox model. *Survival Vignettes*, 2(3):1–25.

Thind, B., Multani, K., and Cao, J. (2020). Neural networks as functional classifiers. *arXiv preprint arXiv:2010.04305*.

Thind, B., Multani, K., and Cao, J. (2023). Deep learning with functional inputs. *Journal of Computational and Graphical Statistics*, 32:171–180.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Wang, H. and Cao, J. (2023a). Functional nonlinear learning. *Journal of Computational and Graphical Statistics*, 0:1–32.

Wang, H. and Cao, J. (2023b). Nonlinear prediction of functional time series. *Environmetrics*, 34(5):e2792.

Wang, J.-L., Chiou, J.-M., and Müller, H.-G. (2016a). Functional data analysis. *Annual Review of Statistics and Its Application*, 3(1):257–295.

Wang, L., Chen, G., and Li, H. (2007). Group scad regression analysis for microarray time course gene expression data. *Bioinformatics*, 23(12):1486–1494.

Wang, Q., Wang, H., Gupta, C., Rao, A., and Khorasgani, H. (2020). A non-linear function-on-function model for regression with time series data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 232–239.

Wang, Y., Yao, H., and Zhao, S. (2016b). Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242.

Wu, S., Beaulac, C., and Cao, J. (2023). Neural networks for scalar input and functional output. *Statistics and Computing*, 33(5):118.

Wu, S., Beaulac, C., and Cao, J. (2024). Functional autoencoder for smoothing and representation learning. *arXiv preprint arXiv:2401.09499*.

Yang, X., Abraham, L., Kim, S., Smirnov, P., Ruan, F., Haibe-Kains, B., and Tibshirani, R. (2022). Fastcph: Efficient survival analysis for neural networks. *arXiv preprint arXiv:2208.09793*.

Yao, F., Fu, Y., and Lee, T. C. M. (2010). Functional mixture regression. *Biostatistics*, 12(2):341–353.

Yao, F., Müller, H.-G., and Wang, J.-L. (2005a). Functional data analysis for sparse longitudinal data. *Journal of the American statistical association*, 100(470):577–590.

Yao, F., Müller, H.-G., and Wang, J.-L. (2005b). Functional linear regression analysis for longitudinal data. *The Annals of Statistics*, 33(6):2873–2903.

Yao, J., Mueller, J., and Wang, J. (2021). Deep learning for functional data analysis with adaptive basis layers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 11898–11908.

Zhang, X. and Wang, J.-L. (2014). Varying-coefficient additive models for functional data. *Biometrika*, 102(1):15–32.

Zhong, R., Liu, S., Li, H., and Zhang, J. (2022). Functional principal component analysis estimator for non-gaussian data. *Journal of Statistical Computation and Simulation*, 92(13):2788–2801.

Zhou, L. and Pan, H. (2014). Principal component analysis of two-dimensional functional data. *Journal of Computational and Graphical Statistics*, 23(3):779–801.

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

Appendix A

Appendix to Chapter 2

A.1 List of All Notations

A full list of notations used throughout Chapter 2 is provided in Table A.1.

Table A.1: A summary of notations used in the manuscript.

Notation	Description
N	Number of subjects
n_{train}	Number of subjects in the training set
n_w	Number of weights in the neural network
P	Number of scalar predictors
t	The time space where Y can take value
$\mathcal{T}$	The domain interval of t
t_{ij}, t_{iq}	The $j(q)$-th observed time point for the i-th subject
t_j, t_q	The $j(q)$-th equally spaced time point in the interval $\mathcal{T}$
T	The length of the domain interval $\mathcal{T}$
m_i	Number of discrete observations of the functional response for the i-th subject
m	The same number of discrete observations of the functional response assumed for all subjects
m_{irr}	Number of time points with at least one observation given all training subjects for irregularly observed functional data
Q	A predefined number of equally spaced time points covering $\mathcal{T}$ entirely
X_{ip}	The p-th scalar predictor for the i-th subject
$\boldsymbol{X}_i$	The P-dimensional vector of predictors for the i-th subject
$\boldsymbol{X}_{\text{new}}$	The matrix of a new set of inputs
$\boldsymbol{X}_{\text{poly}}$	A subset of continuous covariates randomly selected for polynomial transformations in simulation study
$\mathbf{X}$	The matrix of P-dimensional predictors for all subjects in a traditional regression
$Y(t)$	The underlying stochastic process of the functional response
$Y_i(t), Y_i$	The functional response for the i-th subject

$\hat{Y}_i(t)$	The predicted functional response for the i-th subject
$\hat{\boldsymbol{Y}}(t)$	The vector of the predicted functional response for all subjects
$\hat{\boldsymbol{Y}}$	The matrix of the predicted functional response for all training subjects evaluated at all m observed time points
$\tilde{Y}_i(t)$	The centered functional response for the i-th subject
$Y_i(t_{ij})$	The discrete observation of the functional response at time t_{ij} for the i-th subject
$\hat{Y}_i(t_{ij})$	The predicted functional response evaluated at time t_{ij} for the i-th subject
Y	The vector of response for all observations in a traditional regression
$Z_i(t_{ij})$	The noisy observation of the functional response at time t_{ij} for the i-th subject
$\boldsymbol{Z}$	The $N \times m$ matrix of noisy observations at m time point of the functional response for all subjects
$\epsilon_i(t_{ij})$	The observation error for the functional response at time t_{ij} for the i-th subject
c_{ik}	The basis coefficient corresponding to the k-th basis function $\theta_k(t)$ for the i-th subject
$\boldsymbol{C}_i$	The K_b-dimensional vector of basis coefficients c_{ik}'s for the i-th subject
c_{ik}°	The least square estimator of c_{ik}
$\hat{c}_{ik}$	The neural network estimator of c_{ik}
$\boldsymbol{C}$	The matrix of basis coefficients for all subjects in the training set
$\hat{\boldsymbol{C}}$	The matrix of neural network-estimated basis coefficients for all subjects in the training set
$\hat{\boldsymbol{C}}_{\text{new}}$	The matrix of neural network-estimated basis coefficients for a new set of inputs
$\theta_k(t)$	The k-th basis function
$\boldsymbol{\theta}$	The vector of K_b basis functions $\theta_k(t)$'s
Θ	The $K_b \times m$ matrix with the k-th row being entries of $\theta_k(t)$ evaluated at all m observed time
Θ_{irr}	The $K_b \times m_{\text{irr}}$ matrix with the k-th row being entries of $\theta_k(t)$ evaluated at all m_{irr} time points
$\mu(t)$	The mean function of $Y(t)$
$\hat{\mu}(t)$	An estimator of $\mu(t)$ with the observed data
$K(t,t')$	The covariance function of $Y(t)$
$\hat{K}(t,t')$	An estimator of $K(t,t')$ with the observed data
γ_k	The k-th eigenvalue (in decreasing order) by the spectral decomposition of $K(t,t')$
ξ_{ik}	The k-th functional principal component score (FPC score) for the i-th subject
ξ_{ik}°	An estimator of ξ_{ik} with the observed data
$\boldsymbol{\xi}$	The matrix of FPC scores for all subjects in the training set
$\hat{\boldsymbol{\xi}}_{\text{new}}$	The matrix of neural network-estimated FPC scores for a new set of inputs
$\phi_k(t)$	The k-th functional principal component (FPC); The eigenfunction corresponding to the k-th eigenvalue γ_k
$\boldsymbol{\phi}$	The vector of the K_τ leading FPCs
$\hat{\phi}_k(t)$	An estimator of $\phi_k(t)$ with the observed data
$\hat{\boldsymbol{\phi}}$	The vector of the K_τ estimated FPCs

K_b	A predefined truncation integer determining the number of basis functions used
K_τ	The truncation integer determining number of the FPCs used
K_p	The truncation integer determining the number of basis functions of functional parameters
K	Number of random curves used for generating functional response in simulation study
τ	The percentage of variance explained that determines K_τ
$F(\cdot)$	The mapping function
g_ℓ	The activation function at the ℓ-th hidden layer of the neural network
$w_{\ell p}$	The network weight connecting the p-th neuron in the ℓ-th hidden layer of the neural network
b_p	The network bias in the ℓ-th hidden layer of the neural network
η	The parameter set containing weights $\{w_{\ell p}\}_{\ell=1}^{L+1}$ and bias $\{b_\ell\}_{\ell=1}^{L+1}$ of all layers of the neural network
$\hat{\eta}$	The optimized parameter set of the neural network
E	Number of epochs for training neural networks
ϱ	The learning rate for training neural networks
L	Number of hidden layers of the neural network
$L(\eta)$	The objective function calculating the mean squared error (MSE) between the actual and estimated outputs of the neural network
$L_{\boldsymbol{C}}(\eta)$	The objective function calculating the MSE between c_{ik} and $\hat{c}_{ik}$ in NNBB
$L_{\boldsymbol{\xi}}(\eta)$	The objective function calculating the MSE between ξ_{ik} and $\hat{\xi}_{ik}$ in NNSS
$L_{\boldsymbol{Y}}(\eta)$	The objective function calculating the MSE between $Y_i(t_{ij})$ and $\hat{Y}_i(t_{ij})$ in NNBR and NNSR with regularly-spaced functional data
$L_{\boldsymbol{Y}_{\text{irr}}(\eta)}$	The objective function calculating the MSE between $Y_i(t_{ij})$ and $\hat{Y}_i(t_{ij})$ in NNBR and NNSR with irregularly spaced functional data
$L_{\boldsymbol{pen}}(\eta)$	The penalized objective function
λ	A tuning parameter that controls the smoothness of the predicted functional curve
$\Delta^2 c_{ik}$	The second-order difference of $\{c_{ik}\}_{k=1}^{K_b}$ for the i-th subject; $\Delta^2 c_{ik} = c_{ik} - 2c_{ik-1} + c_{ik-2}, k \in \{3, ..., K_b\}$
$\hat{\mathbf{B}}$	The vector of the least squares estimators of the coefficients in a traditional regression
$\psi_k(t)$	The k-th random curve for generating functional response in simulation study
$\zeta_k(\cdot)$	The function that maps $\boldsymbol{X}$ to coefficients for generating functional response in simulation study
$\zeta_k(\boldsymbol{X})$	The k-th associated coefficient mapped from $\boldsymbol{X}$ for generating functional response in simulation study
$B_k(t), B_l(t)$	The $k(l)$-th B-spline basis function
$\beta_{k,l}$	The l-th randomly generated basis coefficient for $\psi_k(t)$
$O(\cdot)$	The big O

A.2 Model Configurations in Real Application

Table A.2 summarizes the configurations for models applied in the real application.

Table A.2: Configurations of all models applied (except FAM) in the real application with ASFR data set.

Model	FoS	NNBB	NNSS	NNBR	NNBR(P)	NNSR
Number of hidden layers (L)	-	2	2	2	2	2
Number of neurons per hidden layer	-	[50, 30]	[50, 30]	[50, 30]	50, 30]	[50, 30]
Activation functions	-	[Sigmoid, ReLU]	[ReLU, ReLU]	[Sigmoid, ReLU]	[Sigmoid, ReLU]	[Sigmoid, ReLU]
Batch size	-	8	8	8	8	8
Epochs (E)	-	1500	1500	1500	1500	1500
Number of basis functions (K_b)	6	6	-	6	6	-
Type of basis functions	B-spline	B-spline	-	B-spline	B-spline	-
% of variance explained (τ)	-	-	99%	-	-	99%
Roughness penalty parameter (λ)	-	-	-	-	10^{-7}	-
Type of roughness penalty	-	-	-	-	2nd Derivative	-

Appendix B

Appendix to Chapter 3

B.1 Simulation Studies: Additional Details

B.1.1 Model Configurations

The details of configurations for models utilized in Scenario 1.1, 1.2, 2.1, and 2.2 in the simulation stuides section are presented in Table B.1, Table B.2, Table B.3, and Table B.4, separately.

Table B.1: A summary of configurations for FPCA and FAEs trained in Scenario 1.1.

	FPCA	**FAE** (linear)	**FAE** (nonlinear)
No. of representations attempted (K)	3, 5, 10	3, 5, 10	3, 5, 10
No. of hidden layer (L)	-	1	1
No. of neurons in hidden layers ($K^{(l)}$)	-	$[K]$	$[20, K, 20]$
Activation function ($g(\cdot)$)	-	Identity	Softplus
Training epochs	-	500	1000
Batches size	-	256	256
Optimizer	-	AdamW	AdamW
Learning rate	-	10^{-2}	10^{-2}
SD for weight initialization (σ)	-	1	1
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	10, 10	10, 10
Type of basis $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$	-	B-spline, B-spline	B-spline, B-spline
No. of basis for curve smoothing	10	-	-
Type of basis for curve smoothing	B-spline	-	-
Smoothing penalty parameter (λ)	-	0	0

Table B.2: A summary of configurations for FPCA and FAEs trained in Scenario 1.2.

	FPCA	**FAE** (linear)	**FAE** (nonlinear)
No. of representations attempted (K)	3, 5, 10	3, 5, 10	3, 5, 10
No. of hidden layer (L)	-	1	3
No. of neurons in hidden layers ($K^{(l)}$)	-	[K]	[150, K, 150]
Activation function ($g(\cdot)$)	-	Identity	Sigmoid
Training epochs	-	1000	2000
Batches size	-	128	128
Optimizer	-	AdamW	AdamW
Learning rate	-	10^{-2}	10^{-2}
SD for weight initialization (σ)	-	1	1
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	50, 50	50, 50
Type of basis $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$	-	B-spline, B-spline	B-spline, B-spline
No. of basis for curve smoothing	15	-	-
Type of basis for curve smoothing	B-spline	-	-
Smoothing penalty parameter (λ)	-	0.001	0.001

Table B.3: A summary of configurations for AE and FAE trained in Scenario 2.1.

	AE	**FAE**
No. of representations attempted (K)	3, 5, 10	3, 5, 10
No. of hidden layer (L)	3	3
No. of neurons in hidden layers ($K^{(l)}$)	[150, K,150]	[150, K, 150]
Activation function ($g(\cdot)$)	Sigmoid	Sigmoid
Training epochs	2000	2000
Batches size	128	128
Optimizer	AdamW	AdamW
Learning rate	10^{-2}	10^{-2}
SD for weight initialization (σ)	1	1
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	50, 50
Type of basis $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$	-	B-spline, B-spline
Smoothing penalty parameter (λ)	-	0.001

Table B.4: A summary of configurations for AE and FAE trained in Scenario 2.2.

	AE		**FAE**	
Size of training set	80%	20%	80%	20%
No. of representations attempted (K)	3, 5, 10	3, 5, 10	3, 5, 10	3, 5, 10
No. of hidden layer (L)	3	3	3	3
No. of neurons in hidden layers ($K^{(l)}$)	[50, K, 50]	[50, K, 50]	[50, K, 50]	[50, K, 50]
Activation function ($g(\cdot)$)	Softplus	Softplus	Softplus	Softplus
Training epochs	2000	5000	2000	5000
Batches size	128	128	128	128
Optimizer	AdamW	AdamW	AdamW	AdamW
Learning rate	10^{-2}	10^{-2}	10^{-2}	10^{-2}
SD for weight initialization (σ)	1	1	1	1
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	-	50, 50	50, 50
Type of basis $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$	-	-	B-spline, B-spline	B-spline, B-spline
Smoothing penalty parameter (λ)	-	-	0.001	0.001

B.1.2 Statistical Results

This section displays how the mean prediction error (MSE_p) and mean classification accuracy ($\text{P}_\text{classification}$) of the proposed FAE and conventional AE changes with the number of training epochs for different training sizes and different dimensions of the representation for scenario 2.2 in the simulation studies section. The prediction error is measured by the mean prediction error squared prediction error (MSE_p) averaged across the number of samples and the number of observed time points in the test set. The classification accuracy, $\text{P}_\text{classification}$, is calculated as the percentage of test observations that can be labelled correctly by a logistic regression based on the representations extracted.

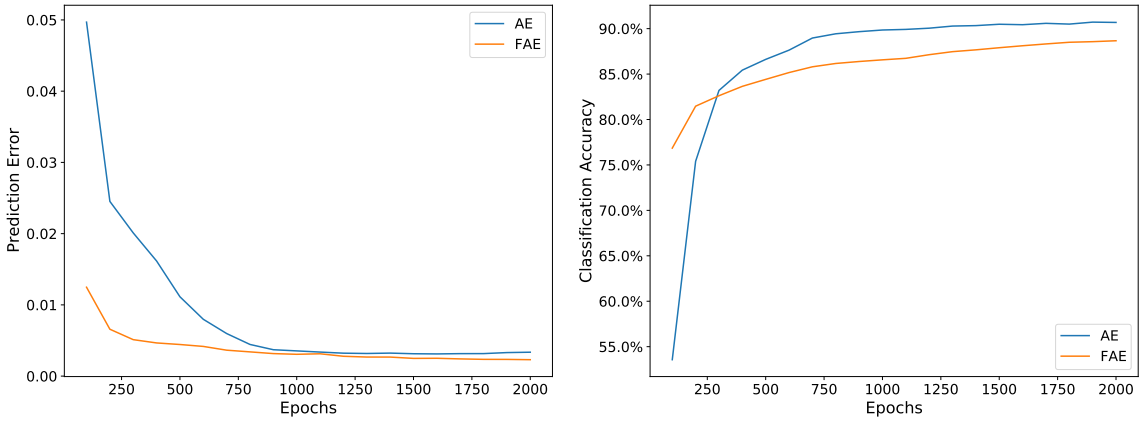

Figure B.1: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 3 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 80% of data used for training in Scenario 2.2.

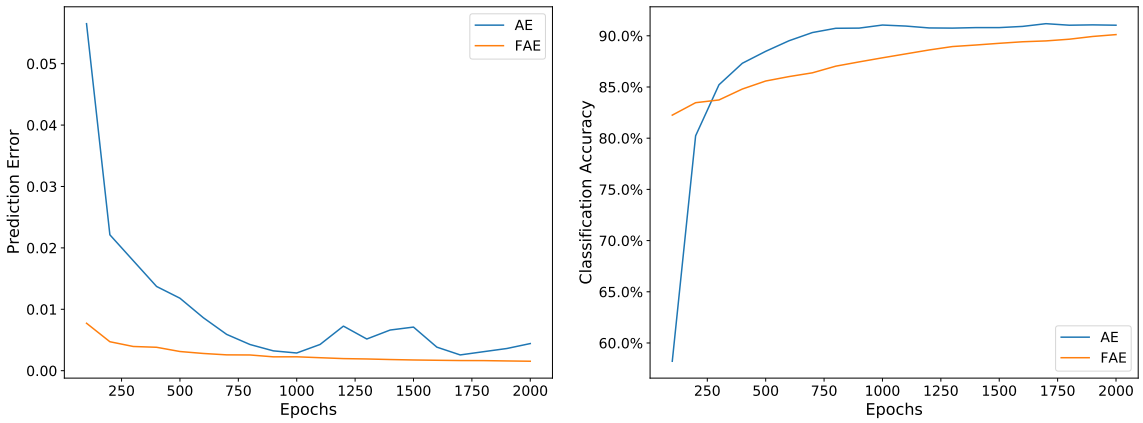

Figure B.2: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 5 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 80% of data used for training in Scenario 2.2.

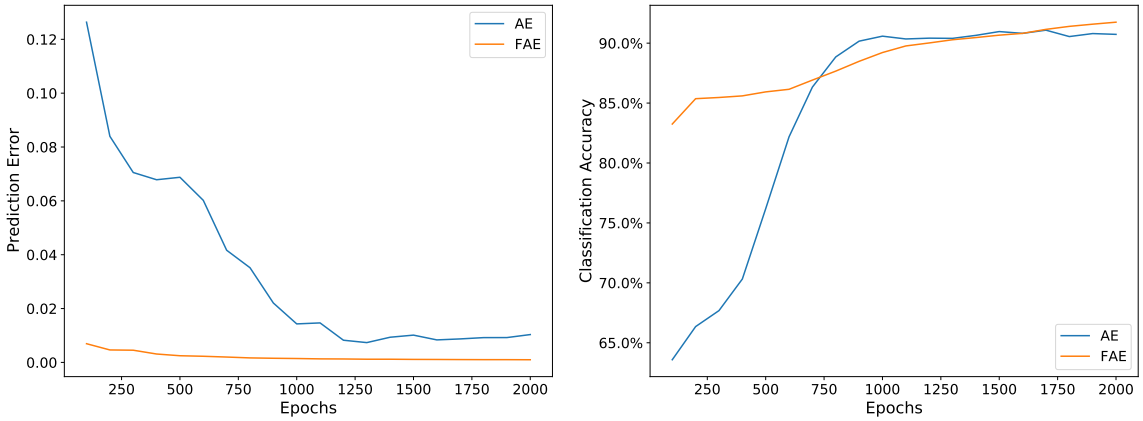

Figure B.3: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 10 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 80% of data used for training in Scenario 2.2.

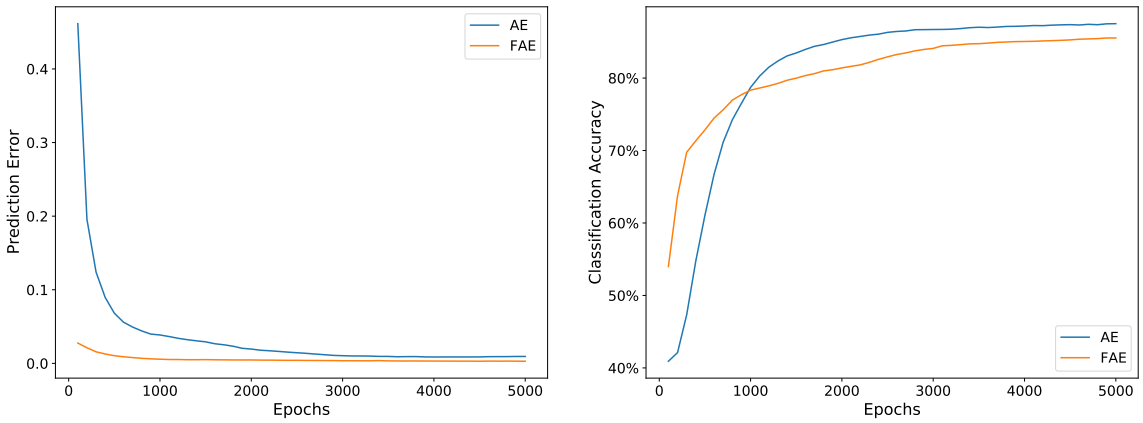

Figure B.4: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 3 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 20% of data used for training in Scenario 2.2.

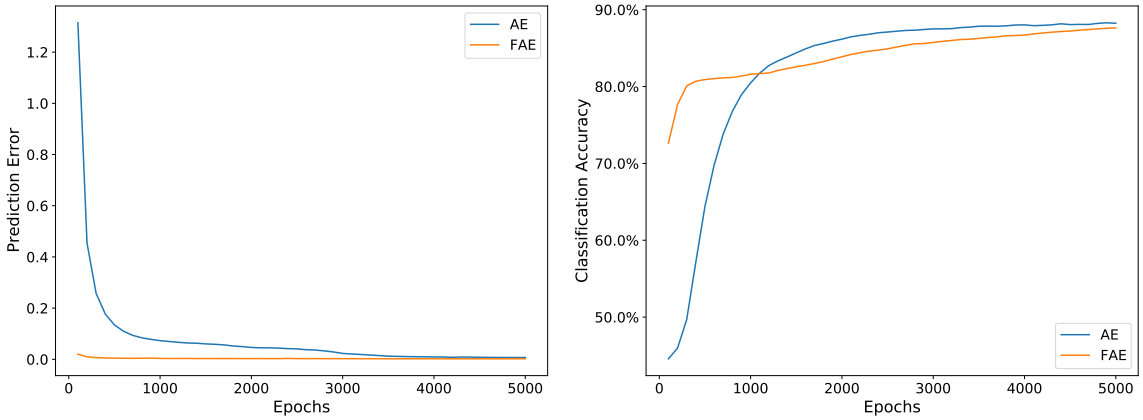

Figure B.5: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 5 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 20% of data used for training in Scenario 2.2.

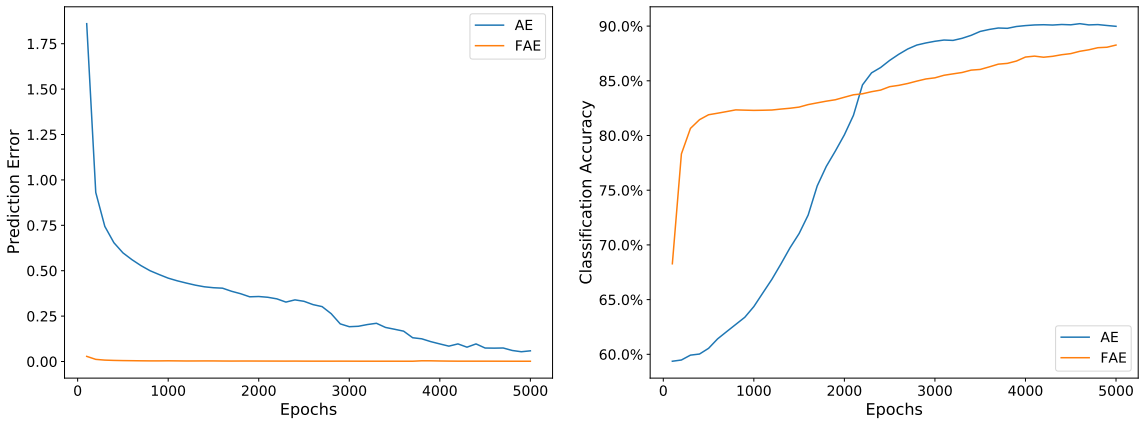

Figure B.6: How the averaged prediction error and classification accuracy of functional autoencoder (FAE) and classic autoencoder (AE) with the softplus activation function using 10 representations on 10 random test sets change with the number of epochs, given the functional data are irregularly observed and 20% of data used for training in Scenario 2.2.

B.2 Real Application: Additional Details

B.2.1 Hyperparameter Tuning

Table B.5 catalogs the candidate values of interest used in hyperparameter tuning in the real application.

B.2.2 Model Configurations

The details of configurations for models applied in the real application are provided in Table B.6.

Table B.6: A summary of the identified configurations for FPCA, AEs and FAEs trained with the El Niño data set.

	FPCA	**AE** (Linear)	**AE** (Nonlinear)	**FAE** (Linear)	**FAE** (Nonlinear)
No. of representations attempted (K)	3, 5, 8	3, 5, 8	3, 5, 8	3, 5, 8	3, 5, 8
No. of hidden layer (L)	-	1	3	1	3
No. of neurons in hidden layers ($K^{(l)}$)	-	[K]	[10, K, 10]	[K]	[100, K, 100]
Activation function ($g(\cdot)$)	-	Identity	Sigmoid	Identity	Sigmoid
Training epochs	-	5000	5000	5000	5000
Batches size	-	28	28	28	28
Optimizer	-	Adam	Adam	Adam	Adam
Learning rate	-	10^{-2}	10^{-2}	10^{-3}	10^{-3}
SD for weight initialization (σ)	-	0.5	0.5	0.5	0.5
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	-	-	20, 20	20, 20
Type of basis $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$	-	-	-	B-spline, B-spline	B-spline, B-spline
No. of basis for curve smoothing	10	-	-		
Type of basis for curve smoothing	B-spline	-	-		
Smoothing penalty parameter (λ)	-	-	-	0.001	0.001

Table B.5: A list of candidate values of interest used in hyperparameter tuning for all models in comparison.

	FPCA	**AE** (Linear)	**AE** (Nonlinear)	**FAE** (Linear)	**FAE** (Nonlinear)
No. of neurons in hidden layers ($K^{(l)}$)	-	-	[10, K, 10], [20, K, 20], [50, K, 50],[100, K, 100]	-	[10, K, 10], [20, K, 20], [50, K, 50],[100, K, 100]
Activation function ($g(\cdot)$)	-	-	Sigmoid, Softplus, Tanh, ReLU	-	Sigmoid, Softplus, Tanh, ReLU
Training epochs	-	2000, 5000	2000, 5000, 10000	2000, 5000	2000, 5000, 10000
Optimizer	-	Adam, AdamW, Adamax	Adam, AdamW, Adamax	Adam, AdamW, Adamax	Adam, AdamW, Adamax
Learning rate	-	10^{-2}, 10^{-3}	10^{-2}, 10^{-3}	10^{-2}, 10^{-3}	10^{-2}, 10^{-3}
No. of $\phi_m^{(I)}(t)$ & $\phi_m^{(O)}(t)$ ($M^{(I)}$ & $M^{(O)}$)	-	-	-	(15, 15), (20, 20)	(15, 15), (20, 20)
No. of basis for curve smoothing	5, 8, 10	-	-		
Type of basis for curve smoothing	B-spline, Fourier	-	-	-	-
Smoothing penalty parameter (λ)	-	-	-	0.01, 0.001, 0.0001	0.01, 0.001, 0.0001

B.2.3 Statistical Results

Table B.7 contains the results of the two-sided paired t-tests comparing the performances of the proposed FAE with nonlinear activation function and the classic FPCA in the real application.

Table B.7: Means, standard deviations (displayed inside parentheses) and the p-values of two-sided paired t-test of the prediction error and classification accuracy of functional autoencdoer with the sigmoid activation function (FAE(Sigmoid) and functional principal component analysis (FPCA) on 20 random test sets with the El Niño data set.

		FAE (Sigmoid)	**FPCA**	**p-value of t-test**
MSE_p	3 reps	0.0582(0.0045)	0.0656(0.0054)	3.1262×10^{-8}
	5 reps	0.0226(0.0031)	0.0242(0.0031)	0.0038
	8 reps	0.0089(0.0014)	0.0113(0.0013)	2.2727×10^{-15}
$\text{P}_\text{classification}$	3 reps	77.68%(5.07%)	77.59%(4.81%)	0.9237
	5 reps	86.52%(4.46%)	84.38%(5.20%)	0.0102
	8 reps	87.59%(4.67%)	84.81%(4.50%)	0.0358

Appendix C

Appendix to Chapter 4

C.1 Time Splitting

The time splitting approach partitions individuals' survival records into sub-intervals $\{\mathcal{T}^{(m)}\}_{m=1}^{M}$ containing corresponding records. For the individual i, the observed duration T_i and the event indicator δ_i are together rewritten as $T_i^{(m)}$ and $\delta_i^{(m)}$ for $\mathcal{T}^{(m)}, m = 1, ..., M_i$, where $T_i \in \mathcal{T}^{(M_i)}$, following

$$T_i^{(m)} = \begin{cases} T_i, \text{ if } T_i \in \mathcal{T}^{(m)}, \\ \mathcal{T}_{\max}^{(m)}, \text{ if } T_i \notin \mathcal{T}^{(m)}. \end{cases} \tag{C.1}$$

$$\delta_i^{(m)} = \begin{cases} \delta_i, \text{ if } t_i \in \mathcal{T}^{(m)}, \\ 0, \text{ if } t_i \notin \mathcal{T}^{(m)}. \end{cases} \tag{C.2}$$

Here, $\mathcal{T}_{\max}^{(m)}$ denotes the upper bound of the sub-interval $\mathcal{T}^{(m)}$. For individuals experiencing several time intervals, all intervals ahead of the last one are marked as censored status, while only the last record contains the final status.

C.2 Additional Details for Simulation Studies

We provide additional simulation details, including formulations, descriptions of parameters, and model configurations used in the simulation studies section.

C.2.1 Scenario 1: Proportional & Linear

In this scenario, we simulated the survival time for a fixed interval of $\mathcal{T} = [0, 300]$ from a proportional hazard model

$$h(t|\boldsymbol{x}) = h_0(t)\exp\{g(\boldsymbol{x})\}, \tag{C.3}$$

$$g(\boldsymbol{x}) = \boldsymbol{\beta}^T\boldsymbol{x}, \tag{C.4}$$

Table C.1 provides the details of the parameters for the simulations in this scenario.

Table C.1: List of parameters used for simulating survival time from a proportional hazard model for different simulation settings in Scenario 1.

	n= 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Baseline hazard (h_0)	0.01	0.01	0.01	0.01
Censoring hazard ($c(t)$)	0.005	0.02	0.005	0.02
Observation interval ($\mathcal{T}$)	$[0, 300]$	$[0, 200]$	$[0, 300]$	$[0, 200]$
$\boldsymbol{\beta}$	[0.8310, 0.5437, 0.1725, 0.9421, -0.7515]			

Note: The identical set of coefficients $\boldsymbol{\beta} = [0.8310, 0.5437, 0.1725, 0.9421, -0.7515]$ is consistently used by different simulation setups in Scenario 1.

C.2.2 Scenario 2: Proportional & Nonlinear

We extended the data generation in Scenario 1 with non-linearity by replacing Eq.(C.4) with a nonlinear and proportional hazard model of form

$$g(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{x} + \frac{3}{2}\left(x_2 x_3 + x_4 x_5 + x_2^2 + x_3^2 + x_5^2\right). \tag{C.5}$$

Again, four data sets differ in size or censoring rate were generated with the constant baseline hazard $h_0 = 0.01$. Table C.2 details the parameters used for scenario 2. Same as the setting for scenario 1, we continued with $\boldsymbol{x} = \{x_1, x_2, ..., x_5\}$, while the first covariate x_1 was drawn from a binomial distribution of $p = 0.5$, and x_2, x_3 and x_4, x_5 were from the uniform distribution of $[-1, 1]$ and $[-2, 1]$, respectively. The coefficient vector $\boldsymbol{\beta}$ contains elements randomly drawn from an uniform distribution between -1 and 1. We sampled the censoring time using various observation intervals and censoring hazard, and labeled whoever were observed at risk at the end of the observation period to be censored individuals to ensure the desired censoring percent can be achieved.

C.2.3 Scenario 3: Nonproportional & Nonlinear

In this scenario, the survival data sets were generated from a nonproportional hazard model

$$\begin{aligned} h(t|\boldsymbol{x}) &= h_0(t)\exp\{g(\boldsymbol{x}, t)\}, \\ g(\boldsymbol{x}, t) &= a(\boldsymbol{x}) + b(\boldsymbol{x})t, \\ a(\boldsymbol{x}) &= \boldsymbol{\beta}^T \boldsymbol{x} + \sin(x_5), \\ b(\boldsymbol{x}) &= \left|\frac{3}{2}\sum_{i=1}^{5} x_i + x_4 x_5\right|, \end{aligned} \tag{C.6}$$

Table C.2: List of parameters used for simulating survival time from a proportional but nonlinear hazard model for different simulation settings in Scenario 2.

	n = 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Baseline hazard (h_0)	0.01	0.01	0.01	0.01
Censoring hazard ($c(t)$)	0.02	0.18	0.02	0.18
Observation interval ($\mathcal{T}$)	$[0, 100]$	$[0, 20]$	$[0, 100]$	$[0, 20]$
$\boldsymbol{\beta}$	$[0.6455, 0.8015, 0.7111, 0.7286, 0.6662]$			

Note: The identical set of coefficients $\boldsymbol{\beta} = [0.6455, 0.8015, 0.7111, 0.7286, 0.6662]$ is consistently used by different simulation setups in Scenario 2.

Table C.3: List of parameters used for simulating survival time from a nonproportional & nonlinear hazard model for different simulation settings in Scenario 3.

	n = 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Baseline hazard (h_0)	0.001	0.001	0.001	0.001
Censoring hazard ($c(t)$)	0.05	0.15	0.05	0.15
Observation interval ($\mathcal{T}$)	$[0, 30]$	$[0, 20]$	$[0, 30]$	$[0, 20]$
$\boldsymbol{\beta}$	$[-0.3480, -0.0127, -0.5468, 0.8260, 0.3143]$			

Note: The identical set of coefficients $\boldsymbol{\beta} = [-0.3480, -0.0127, -0.5468, 0.8260, 0.3143]$ is consistently used by different simulation setups in Scenario 3.

and this time the baseline hazard h_0 is set to be 0.001. The covariates $\boldsymbol{x}$ and the corresponding coefficients $\boldsymbol{\beta}$ were randomly sampled from the identical distributions as those mentioned in the other scenarios. Again, we simulated data sets of 500 subjects and 5000 subjects with an approximate censoring rate of 0.3 and 0.6, respectively, by controlling the length of observation interval $\mathcal{T}$ and censoring hazard $c(t)$. Analogously, Table C.3 lists the values of mentioned parameters used across different simulations under this scenario.

In practice, when fitting NN-Cox(NP) models, we sliced the domain interval into consecutive sub-intervals and performed proportional hazards on the dataset post time splitting with the extracted features plus several time-involved interactions. The model configurations, given in Table C.4, were applied in all three scenarios.

Table C.4: A summary of the configurations used for neural network training and hazard model fitting in all scenarios.

	n = 500		**n=5000**	
	CR≈0.3	CR≈0.6	CR≈0.3	CR≈0.6
Neurons of the 1st hidden layer	20	20	20	20
Neurons of the 2nd hidden layer (number of features)	10	10	10	10
Activation function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Batch size	48	48	48	48
Epochs	100	100	100	100
Length of sub-intervals				
Scenario 1	30	20	30	20
Scenario 2	10	2	10	2
Scenario 3	1	1	1	1
Number of time-involved interactions	4	4	4	4

C.3 Additional Details for Real Application

We include more information about the 3 real data sets and hyperparameters involved in the model fitting procedure in the real applications section in Table C.5 and Table C.6, respectively.

Table C.6: A description of the hyperparameters used for neural network training and hazard model fitting for the 3 data sets.

	METABRIC	Rot. & GBSG	FLCHAIN
Neurons of the 1st hidden layer	30	30	20
Neurons of the 2nd hiddenlayer (number of features)	15	15	10
Activation function	Sigmoid	Sigmoid	Sigmoid
Batch size	32	24	64
Epochs	100	50	50
Length of sub-intervals	50	10	400
Number of time-involved interactions	8	8	5

C.4 Additional Figures

In this section, we provide the figures that display the relationship between the observed censoring/event time T vs. transformed response Y under the three transformation methods,

Table C.5: A summary of the 3 data sets employed in the real application.

Dataset	No. of individuals	Covariates	Shortest & longest duration	Unique durations	Censoring rate
METABRIC	1903	9	[0.1, 355.2]	1685	42.04%
Rot. & GBSG	2232	7	[0.2628337, 87.3593440]	1230	43.23%
FLCHAIN	6521	8	[1, 5166]	2714	69.96%

including reweighing, mean imputation and deviance residual, using the Molecular Taxonomy of Breast Cancer International Consortium (METABRIC) dataset and the Assay of Serum Free Light Chain (FLCHAIN) dataset (Kvamme et al., 2019) from the real application section, respectively.

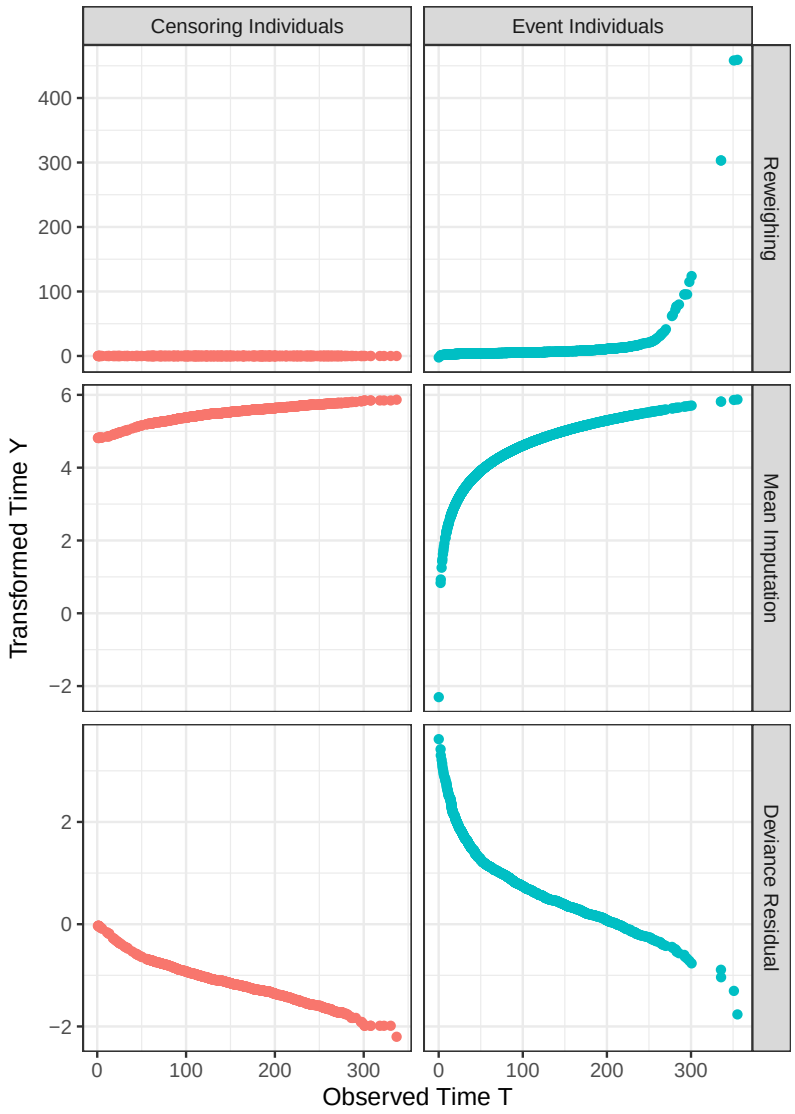

Figure C.1: Transformed time-to-event response Y, obtained from reweighing, mean imputation, and deviance residual, vs. the observed time T for censoring and event individuals in the METABRIC dataset (Kvamme et al., 2019).

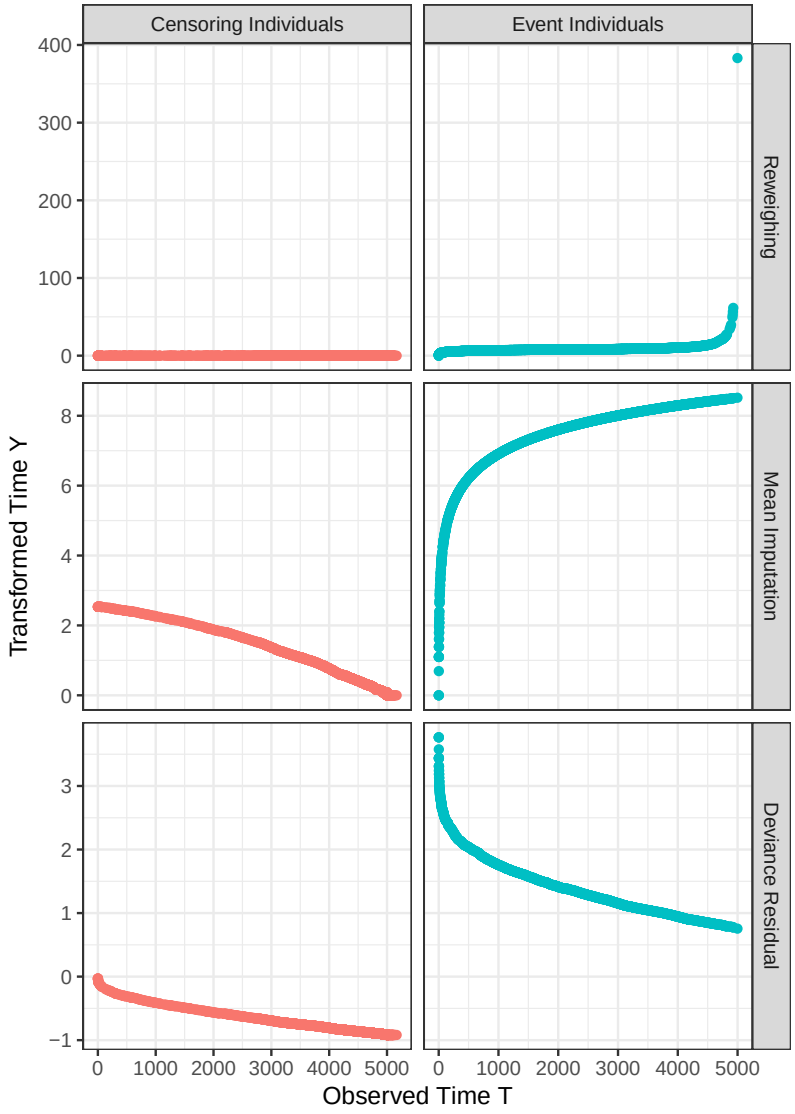

Figure C.2: Transformed time-to-event response Y, obtained from reweighing, mean imputation, and deviance residual, vs. the observed time T for censoring and event individuals in the FLCHAIN dataset (Kvamme et al., 2019).