UNIVERSITY OF CALIFORNIA,
IRVINE

Conditional Distribution Function Estimation Using Neural Networks

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Statistics

by

Bingqing Hu

2024

DEDICATION

To my parents.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vii**

ACKNOWLEDGMENTS **viii**

VITA **ix**

ABSTRACT OF THE DISSERTATION **x**

1 Introduction **1**

2 Conditional Survival Function Estimation Using Neural Networks for Right Censored Data with Time-Varying Covariates **4**

2.1 Introduction 4

2.2 The Estimating Method Using Neural Networks 8

2.2.1 Data and Notation 8

2.2.2 Likelihood 10

2.2.3 Data Structure and Discretized Loss 11

2.2.4 Neural Networks, Hyperparameters and Regularization 11

2.3 Simulations 13

2.4 Real-World Data Sets 16

3 Conditional Distribution Function Estimation Using Neural Networks for Continuous Response with Multiple Covariates **21**

3.1 Introduction 21

3.2 The Generalized Estimating Method 22

3.3 Simulations 25

3.4 Comparisons to Kernel Methods for Uncensored Data 31

3.5 Real-World Data Sets 32

3.6 Numerical Comparison to ODE Neural Networks 35

4 Survival Estimation with Arbitrary Effects of Time-varying Covariates Using Neural Networks **40**

4.1 Introduction 40

4.2 Methodology 42

4.2.1 Data Structure and DeepONet 44
4.2.2 Full Likelihood and Discretized Loss 46
4.2.3 Details in Network Architectural and Hyperparameter Tuning 47
4.3 Simulations 49
4.4 Discussion 55

5 Conditional Distribution Estimation Given Functional Covariates Using Deep Operator Neural Networks 56
5.1 Introduction 57
5.2 Methodology 59
5.2.1 Continuous Response 59
5.2.2 Discrete Response 62
5.3 Hyperparameters 64
5.4 Simulations 65
5.4.1 Continuous Response 65
5.4.2 Discrete Response 71
5.5 Real Data 71

Bibliography 75

LIST OF FIGURES

Page

2.1 An example of fully connected feed forward neural network with 2 hidden layers. In this example, the input dimension is 4 plus an intercept term, each hidden layer contains 10 nodes plus an intercept node and the output is a single value. 12
2.2 Conditional survival curves for 9 different sets of covariates when the Cox model is corrected specified (censoring rate around 20%). 17
2.3 Conditional survival curves for 9 different sets of covariates when the Cox proportional hazards assumption is violated (censoring rate around 50%). . . 18

3.1 Conditional distribution functions given 9 different sets of covariate values for uncensored data generated in Setup 1 with c=0.5. 28
3.2 Conditional distribution functions given 9 different sets of covariate values for uncensored data generated in Setup 2 with c=0.5. 29
3.3 Boxplots of MADEs for the NW method and our proposed method. (a) 1-D covariate, n = 1000, (b) 2-D covariates, n = 1000. 32
3.4 Estimated conditional distribution functions for 4 individuals. (a)(b): Fish Toxicity, (c)(d): Airfoil. The three vertical lines illustrate locations of the predicted mean, the predicted median and the observed value. 34
3.5 The memory and time cost vs. sample size for ODE model and our previous model 39

4.1 The unstacked DeepONet Structure. u, y, $G(u)(y)$ in this figure correspond to x, s, $h(x)(s)$ respectively. 45
4.2 Conditional survival curves for 9 different sets of covariates using feed forward neural networks with sample size $n = 2000$. 51
4.3 Conditional survival curves for 9 different sets of covariates using Cox-PH model with sample size $n = 2000$. 52
4.4 Conditional survival curves for 9 different sets of covariates using CNN with sample size $n = 2000$. 53
4.5 Conditional survival curves for with 9 different sets of covariates. 54

5.1 DeepONet structure with CNN layers for branch net 61
5.2 Conditional distribution functions for 9 different sets of functional covariates (setup 1). 68

5.3 Conditional distribution functions for 9 different sets of functional covariates (setup 2). 69
5.4 Conditional distribution functions for 9 different sets of functional covariates (setup 3). 70
5.5 Conditional distribution functions for 9 different sets of functional covariates (Discrete Case) 72
5.6 Estimated conditional distribution functions for 3 individuals. The vertical lines illustrate locations of the observed values. 74

LIST OF TABLES

Page

2.1 The expanded data set for survival problem with time varying covariate, where $(t_1, \ldots, t_n)$ are sorted values of $(y_1, \ldots, y_n)$ from the training set. 10
2.2 Comparisons of different methods (a: Cox-MLP (CC); b: Cox-Time; c: our new method) for analyzing four real data sets. The result of our method is marked with ** when it outperforms both Cox-MLP(CC) and Cox-Time, and is marked with * when it outperforms one of the models. 20

3.1 The expanded data set for estimating the conditional distribution function of a continuous response variable, where $(t_1, \ldots, t_n)$ are sorted values of $(y_1, \ldots, y_n)$ from the training set. 23
3.2 Average mean/median squared errors of both methods and the prediction coverage rate of the new method over 100 replications. 30
3.3 Average empirical cumulative probabilities of 100 repeated test sets at several estimated percentiles ($n = 1000$, $c = 0.5$). 30
3.4 Conditional mean prediction results of the L_2 method and the new method . 34

4.1 The data structure with time varying covariate. 44

5.1 The data structure for functional data. 62
5.2 Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 1) . . 67
5.3 Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 2) . . 68
5.4 Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 3) . . 69
5.5 The prediction coverage rate and mean/median squared errors for mean estimation. 73

ACKNOWLEDGMENTS

This work was supported in part by NIH R01 AG056764, NSF DMS 1915711, NIA RF1 AG075107 and NIH 2R01 AG053555.

I would like to express my deepest gratitude to my advisor, Dr. Bin Nan, for his tremendous support throughout these years. In our weekly meetings, he guided me through the process of conducting rigorous research step by step as a statistician. His integrity and dedication to the field have greatly influenced my own approach to research and have shaped me as a professional.

I would also like to extend my heartfelt thanks to my parents and parents-in-law for their unwavering support and help during this journey.

Finally, I would like to thank my friends, especially Ziqi Zhang, Tong Zou, Crystal Zhang, Yue Wang, Xinyue Cao, and Chiheng Zhou. I have learned so much from each of them, and their friendship has been invaluable throughout this experience.

VITA

Bingqing Hu

EDUCATION

Doctor of Philosophy in Statistics **2019–2024**
University of California, Irvine *Irvine, California, USA*

Master of Science in Applied Statistics **2017–2018**
University of Michigan, Ann Arbor *Ann Arbor, Michigan, USA*

Bachelor of Science in Physics **2013–2017**
Fudan University *Shanghai, China*

RESEARCH EXPERIENCE

Graduate Research Assistant **2020–2024**
University of California, Irvine *Irvine, California, USA*

TEACHING EXPERIENCE

Teaching Assistant **2022, 2024**
University of California, Irvine *Irvine, California, USA*

PUBLICATIONS

Hu, B. and Nan, B. Conditional Distribution Function Estimation Using Neural Networks for Censored and Uncensored Data **2023**
Journal of Machine Learning Research 24, 1-26

ABSTRACT OF THE DISSERTATION

Conditional Distribution Function Estimation Using Neural Networks

By

Bingqing Hu

Doctor of Philosophy in Statistics

University of California, Irvine, 2024

Professor Bin Nan, Chair

This dissertation presents novel neural network-based methods for estimating conditional distribution functions across diverse settings. We introduce a robust approach for survival analysis with right-censored and time-varying covariates, estimating the conditional hazard function and deriving the conditional survival function. This method outperforms traditional Cox proportional hazards models and recent neural network models in both simulated and real-world data scenarios. Expanding the methodology to uncensored data, we propose a unified approach for estimating conditional distribution functions for continuous responses with multiple covariates. Our method shows improved robustness, accuracy, and computational efficiency compared to kernel-based and mean regression neural network methods. Further, we address limitations of traditional models by using DeepONet to capture complex, long-term effects of covariates on hazard functions, demonstrating enhanced flexibility and adaptability. We also develop a deep operator neural network framework for functional data analysis, where covariates are functions, paving the way for advanced analysis of complex data structures.

Chapter 1

Introduction

Estimating a conditional distribution function given covariates is of great interest in many fields such as medicine, finance, and engineering. Accurate estimation of these functions allows researchers and practitioners to make informed decisions and predictions based on the relationships between variables. Neural networks have traditionally been employed in classification tasks and conditional mean estimation due to their flexibility and capacity to model complex, non-linear relationships. In this dissertation, we extend the application of neural networks to the estimation of conditional distribution functions given multiple, different types of covariates.

In Chapter 2, we address the survival analysis problem with right-censored data and time-varying covariates. Survival analysis often involves incomplete data due to censoring, which presents unique challenges. We propose a novel method using neural networks to estimate the instantaneous conditional hazard function. From this, we derive the conditional survival function, providing a comprehensive framework for handling censored data. We demonstrate the effectiveness of our method through simulations and applications to real-world datasets, showing superior performance compared to the traditional Cox-PH model and the latest neural network models.

Chapter 3 extends the methodology to uncensored data. Here, we adapt the neural network-based approach to estimate the conditional distribution function for continuous response variables with multiple covariates. This chapter bridges the gap between survival analysis and general regression problems, offering a unified approach to estimate conditional distributions. We compare our method against kernel-based approaches and mean regression neural nets, showcase its advantages in terms of robustness.

In Chapter 4, we expand our survival analysis framework to accommodate arbitrary effects of covariates. Traditional survival models often assume instantaneous effect of covariates on hazard function, which can be restrictive. Our approach leverages the flexibility of neural networks to capture complex, long term effect of covariates on hazard. We introduce the use of deep operator neural network, a deep learning architecture designed for operator learning, to model these arbitrary effects. Through simulations, we validate the robustness and adaptability of our method.

Chapter 5 further extends our approach to the realm of functional data analysis. Functional data, where covariates are functions rather than scalar or vectors, pose significant challenges for traditional statistical methods. We develop a deep operator network framework to estimate conditional distribution functions of both continuous and discrete outcome given functional covariates. This innovative approach opens new avenues for analyzing complex data structures in various applications. We demonstrate our method through both simulations and real-world datasets.

Overall, this dissertation provides a comprehensive exploration of neural network-based methods for estimating conditional distribution functions in diverse settings. By addressing both censored and uncensored data, incorporating arbitrary covariate effects, and extending to functional data analysis, we offer a versatile toolkit for researchers and practitioners.

Our main contribution is the development of an approach that can effectively handle high-dimensional covariates and incorporate censored data. Our method is the most robust non-parametric approach with least assumptions. While it may be less efficient compared to the machine learning and deep learning methods when their model assumptions hold, our approach remains consistently valid regardless of the data distribution, ensuring broad applicability and reliability. The methods developed herein have the potential to significantly impact various applied fields.

Chapter 2

Conditional Survival Function Estimation Using Neural Networks for Right Censored Data with Time-Varying Covariates

2.1 Introduction

Neural networks are widely used in prediction. Depending on the nature of the outcome (or response) variable of interest, most of the current work falls into two categories: classification for a categorical outcome variable and mean prediction for a continuous outcome variable. For the classification problem, one approach is to estimate the conditional distribution function of a categorical response variable given a set of predictors, also called covariates, then to classify a new data point into one of the categories of the response variable based on the estimated predicting probabilities given the new covariate values. A typical example is to use the logistic model for predicting a binary outcome. For a continuous response variable, however, the focus has been primarily on the center of its conditional distribution function which is used to predict a new data point. In fact, estimating the conditional distribution function of a continuous response variable nonparametrically is of greater interest [24]. It is a challenge, however, to estimate the conditional distribution function given multiple continuous covariates [25]. Nevertheless, with continuous predictors, the conditional distribution can provide a richer characterization of the relationship between the predictors and the response and allows for the construction of prediction intervals [24]. We propose to estimate the conditional distribution function of a continuous random variable given multiple covariates using neural networks.

Our work is inspired by the estimation of the conditional survival function for censored failure time data with time-dependent covariates. In the survival analysis literature, a conditional survival function is usually estimated by fitting the semiparametric Cox proportional hazards model [13]. Although it has been widely used, particularly in health studies, the Cox model requires strong modeling assumptions that can be violated in practice. To alleviate the modeling assumptions, kernel smoothing methods have been applied for estimating the hazard function [55, 53]. In [55], for example, a nonparametric estimator for the conditional hazard function is obtained by taking a ratio of the local linear estimators for the conditional density and the survival function. For multi-dimensional covariates, however, kernel smoothing methods suffer poor accuracy, reflected in slow convergence rates [25]. Researchers have been exploring the application of neural networks in survival analysis since the 1990s. One line of research uses neural networks to replace the linear component in a Cox model and the negative (logarithm of) partial likelihood as the loss function, see e.g., [16] and [64].

Recently, [33] revisited the Cox model and applied modern deep learning techniques (standardizing the input, using new activation functions, new optimizers and learning rate scheduling) to improve training. Their neural network model, called DeepSurv, outperforms Cox regressions evaluated via the C-index [26] on several real data sets. In simulation studies, DeepSurv outperforms Cox regression when the proportional hazards assumption is violated and the two methods perform similarly when the proportional hazards assumption is satisfied.

Following a similar modeling strategy, [11] developed the neural network model Cox-nnet for high-dimensional genetics data. Building upon the methodology of nested case-control studies, [40] proposed to use a loss function that modifies the partial likelihood by subsampling the risk sets. Their neural network models, Cox-MLP(CC) and Cox-Time, are computationally efficient and scale well to large data sets. Both models are relative risk models with the same modified partial likelihood but Cox-Time further removes the proportionality constraint by including time as a covariate in the relative hazard. Cox-MLP(CC) and Cox-Time were compared to the classical Cox regression, DeepSurv and a few other models on five real-world data sets and found to be highly competitive. We will compare our method to Cox-MLP(CC) and Cox-Time on four of these data sets (those in which the the number of tied survival times are negligible). In all of the aforementioned methods, the semi-parametric nature of the model is retained, hence the baseline hazard function needs to be estimated in order to estimate the conditional survival function, whereas our method estimates the conditional hazard function directly without specifying any baseline hazard function.

Another commonly used approach is to partition the time axis into a set of fixed intervals so that the survival probabilities are estimated on the set of discrete time points (discrete-time methods). [4] proposed a model called PLANN in which the neural networks contain a input vector of covariates and a discrete time point, and an output of the hazard rate at this time point. They used the negative logarithm of a Bernoulli likelihood as the loss function. [56] proposed a model that outputs a vector with each element representing the survival probability at a predefined time point, and used a modified relative entropy error [54] as the loss function. [6] also discretized the time, but used an ad hoc loss function that is the sum of squared errors, where for each subject at each time interval, the error is the difference between 1 (if a failure is observed in the interval) or 0 (otherwise) and the corresponding hazard component. More recent work includes Nnet-survival [18] and RNN-SURV [19]. Both models require fixed and evenly partitioned time intervals, where Nnet-survival includes a convolutional neural network structure (CNN) and RNN-SURV uses a recurrent neural network structure (RNN). They all use Bernoulli type loss functions with differently created binary variables, where RNN-SURV adds an upper bound of the negative C-index into the loss function.

There are also deep learning methods based on parametric models. They try to estimate the parameters in mixture parametric distributions (Weibull, log-normal, etc.) [2] [3] [46] [45]. A comprehensive systematic review of deep learning based survival methods can be found in [63]. There are several limitations in the current literature of survival analysis using neural networks. Cox-based neural network models are relative risk models with baseline hazards, which retain certain model structure, and the networks only output the relative risks. Parametric models have more specific model assumptions. Discrete-time methods and some of the latest cox-based methods such as [47] use loss functions that are constructed by heuristic criteria other than the true likelihood function. Moreover, most of these methods only deal with time-independent covariates.

To overcome these limitations, we propose a new method to estimate the conditional hazard function directly using neural networks. In particular, inspired by the standard data-expansion approach for the Cox regression with time-varying covariates, we input time-varying covariates together with observed time points into a simple feed-forward network and output the logarithm of instantaneous hazard. We build the loss function from the logarithm of the full likelihood function, in which all functions, including the conditional hazard function, the conditional cumulative hazard function, and covariate processes, are evaluated only at the observed time points. Compared to existing methods, our new method has a number of advantages. First, we can handle time-varying covariates. Second, we make the least number of assumptions that only include conditional independent censoring and that the instantaneous hazard given entire covariate paths only depends on values of covariates observed at the current time. Third, estimating the (logarithm of) hazard function without imposing any constraints to the optimization automatically leads to a valid survival function estimator that is always monotone decreasing and bounded between 0 and 1. Furthermore, because our loss function does not need to identify the risk set, scalable methods (e.g., training with batches in stochastic gradient descent) can be easily implemented to avoid blowing up the computer memory.

2.2 The Estimating Method Using Neural Networks

In this section, we start with the likelihood-based loss function for estimating the conditional survival function given a set of time-varying covariates and then provide an estimating procedure using neural networks.

2.2.1 Data and Notation

For subject i, we denote the time-varying covariate vector as $X_i(t)$, the underlying failure time as T_i, and the underlying censoring time as C_i, where T_i possesses a Lebesgue density. Let the observed time be $Y_i = \min\{T_i, C_i\}$ and the failure indicator be $\Delta_i = I(T_i \leq C_i)$. We have n independent and identically distributed (i.i.d.) observations $\{Y_i, \Delta_i, \widetilde{X}_i(\cdot) : \ i = 1, \ldots, n\}$, where $\widetilde{X}_i(t)$ denotes the covariate history up to time t, that is, $\widetilde{X}_i(t) = \{X_i(s), 0 \leq s \leq t\}$. We assume each of the processes $X_i(t)$ has left continuous sample path with right limit. Let $f(t|\widetilde{X}_i(\infty))$ be the conditional Lebesgue density function of T_i, $f_C(t|\widetilde{X}_i(\infty))$ be the conditional density function of C_i, $S(t|\widetilde{X}_i(\infty))$ be the conditional survival function of T_i, and $S_C(t|\widetilde{X}_i(\infty))$ be the conditional survival function of C_i.

Noting that the conditional survival probability given time-varying covariate is not well-defined if there is an internal covariate, we assume that all covariates are external [32]. Specifically, the conditional hazard function of T_i is independent of future covariate values and, furthermore, only depends on the current values of covariate processes:

$$\lambda\left[t\left|\widetilde{X}_i(\infty)\right.\right] = \lambda\left[t\left|\widetilde{X}_i(t)\right.\right] = \lambda\left[t\left|X_i(t)\right.\right]. \tag{2.1}$$

Let $h(t, X_i(t)) = \log \lambda(t|X_i(t))$. Then the conditional cumulative hazard function given covariate history has the following form:

$$\Lambda\left[t\left|\widetilde{X}_i(\infty)\right.\right] = \Lambda\left[t\left|\widetilde{X}_i(t)\right.\right] = \int_0^t \lambda\left[s\left|\widetilde{X}_i(s)\right.\right] ds = \int_0^t e^{h[s,X_i(s)]} ds,$$

and the conditional survival function is given by

$$S\left[t\left|\widetilde{X}_i(\infty)\right.\right] = S\left[t\left|\widetilde{X}_i(t)\right.\right] = \exp\left\{-\Lambda\left[t\left|\widetilde{X}_i(t)\right.\right]\right\} = \exp\left\{-\int_0^t e^{h[s,X_i(s)]} ds\right\}. \tag{2.2}$$

Using h instead of λ in the above expression removes the positivity constraint for λ, hence simplifies the optimization algorithm for estimating the conditional survival function (2.2). Furthermore, equation (2.2) is always a valid survival function for any function h.

i	start time	stop time	δ_{ij}	covariates
1	$t_0 = 0$	t_1	0	$x_1(t_1)$
1	t_1	t_2	0	$x_1(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$\cdots$	y_1	δ_1	$x_1(y_1)$
2	$t_0 = 0$	t_1	0	$x_2(t_1)$
2	t_1	t_2	0	$x_2(t_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
2	$\cdots$	y_2	δ_2	$x_2(y_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.1: The expanded data set for survival problem with time varying covariate, where $(t_1, \ldots, t_n)$ are sorted values of $(y_1, \ldots, y_n)$ from the training set.

2.2.2 Likelihood

Assume censoring time is independent of failure time given covariates. Then given observed data $\{y_i, \delta_i, x_i(\cdot)\}$, $i = 1, \ldots, n$, the full likelihood function becomes

$$\begin{aligned} L_n &= \prod_{i=1}^{n} \{f[y_i|\widetilde{x}_i(y_i)]S_C[y_i|\widetilde{x}_i(y_i)]\}^{\delta_i} \{f_C[y_i|\widetilde{x}_i(y_i)]S[y_i|\widetilde{x}_i(y_i)]\}^{1-\delta_i} \\ &\propto \prod_{i=1}^{n} \lambda[y_i|x_i(y_i)]^{\delta_i} S[y_i|\widetilde{x}_i(y_i)] \\ &= \prod_{i=1}^{n} \exp\{h[y_i, x_i(y_i)]\delta_i\} \exp\left\{-\int_0^{y_i} e^{h[t,x_i(t)]}dt\right\}. \end{aligned}$$

Thus, the log likelihood is given by

$$\ell_n = \sum_{i=1}^{n} \left\{ h[y_i, x_i(y_i)]\delta_i - \int_0^{y_i} e^{h[t,x_i(t)]}dt \right\}. \tag{2.3}$$

2.2.3 Data Structure and Discretized Loss

When fitting the Cox model with time-varying covariates, the data set is usually expanded to the structure given in Table 2.1 so each row is treated as an individual observation, where $(t_1, \ldots, t_n)$ are sorted values of observed times $(y_1, \ldots, y_n)$ and $\delta_{ij} = \delta_i I(t_j = y_i)$. Specifically, the time axis is partitioned naturally by observed times. The same data structure can be applied to maximizing the log likelihood function (2.3) at the grid points $(t_1, \ldots, t_n)$, or equivalently, minimizing the following loss function:

$$loss(h) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} I(t_j \leq y_i) \left\{ e^{h[t_j, x_i(t_j)]}(t_j - t_{j-1}) - h[t_j, x_i(t_j)]\delta_{ij} \right\}, \tag{2.4}$$

where $t_0 = 0$. It becomes clear that the expanded data set in Table 2.1 provides a natural way of implementing numerical integration in the negative log likelihood $-n^{-1}\ell_n$ based on empirical data. Once an estimator of h is obtained using neural networks (see Subsection 2.2.4 for details), the conditional survival function can be estimated by plugging the estimated h into Equation 2.2.

2.2.4 Neural Networks, Hyperparameters and Regularization

We propose to estimate the arbitrary function $h(t, x_i(t))$ by minimizing the respective loss function (2.4) using neural networks. We then obtain the estimated conditional survival curve from Equation (2.2). The input of neural networks is $(t_{j-1}, t_j, x_i(t_j))$ in each row of Table 2.1, and the output is $\widehat{h}(t, x_i(t))$.

We use tensorflow.keras [12] to build and train the neural networks. The network structure is a fully connected feed forward neural network with two hidden layers and a single output value. The input layer consists of t_{j-1}, t_j, and covariates. An intercept term is included in each layer (see Figure 2.1). The relu function is used as the activation function between input

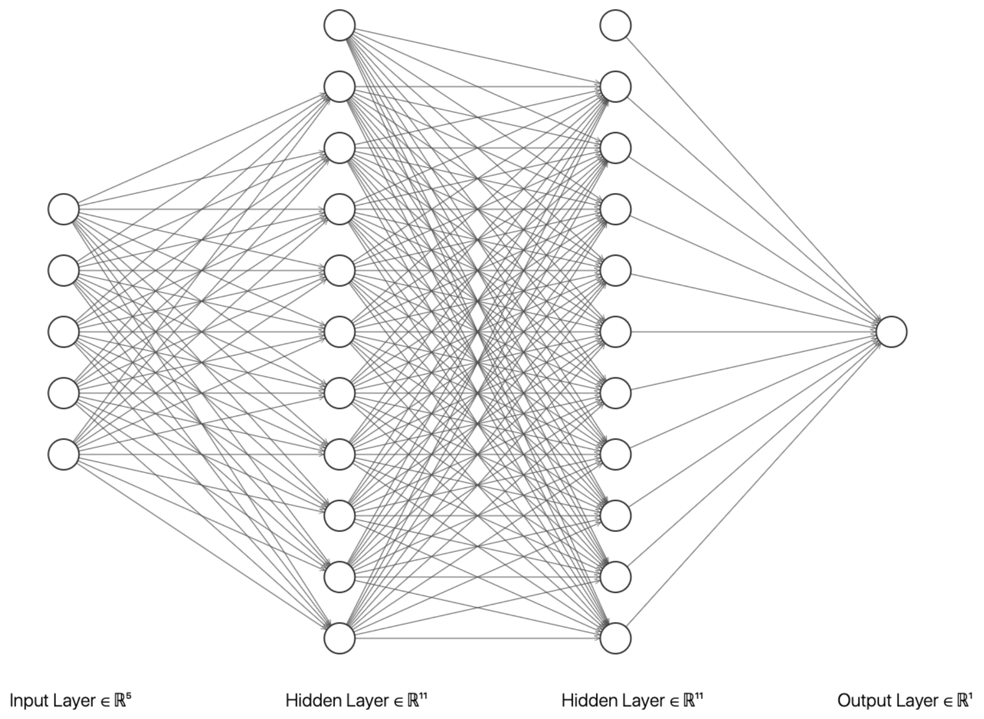

Input Layer $\in \mathbb{R}^5$
Hidden Layer $\in \mathbb{R}^{11}$
Hidden Layer $\in \mathbb{R}^{11}$
Output Layer $\in \mathbb{R}^1$

Figure 2.1: An example of fully connected feed forward neural network with 2 hidden layers. In this example, the input dimension is 4 plus an intercept term, each hidden layer contains 10 nodes plus an intercept node and the output is a single value.

and hidden layers, and the linear function is used for the output so that the output value is not constrained. We use Adam [34] as the optimizer. Other hyperparameters include the number of nodes in each layer, the batch size and the initial learning rate. In our simulations, the number of nodes in each hidden layer is 64, the batch size is 100, and the initial learning rate is 0.001. To have a fair comparison in real-world data examples, we tune the hyperparameters from the set of all combined values with the number of nodes in each hidden layer taken from $\{64, 128, 256\}$, the initial learning rate from $\{0.1, 0.01, 0.001, 0.0001\}$, and the batch size from $\{64, 128, 256\}$.

We use early stopping to avoid over-fitting. According to [20], early stopping has the advantage over explicit regularization methods in that it automatically determines the correct amount of regularization. Specifically, we randomly split the original data into training set and validation set with 1:1 proportion. When the validation loss is no longer decreasing in 10 consecutive steps, we stop the training. To fully use the data, we fit the neural networks again by swapping the training and the validation sets, then average both outputs as the final result.

2.3 Simulations

For censored survival data with time-varying covariates, we compare our method of using neural networks to the partial likelihood method for the Cox model under two different setups. In the first setup, we generate data following the proportional hazards assumption in which case the Cox model is the gold standard. In the second setup, we generate data from the model with a quadratic term and an interaction term in the log relative hazard function so the Cox model with original covariates as linear predictors is misspecified. Details are given below.

1. In both setups, we use the hazard function of a scaled beta distribution as the baseline hazard:

   $$\lambda_0(t) = \frac{f_0(t/\tau)}{1 - F_0(t/\tau)},$$

   where $f_0(.)$ and $F_0(.)$ are the density and the distribution functions of Beta $(8, 1)$, respectively. We use $\tau = 100$ so that $t \in [0, 100]$.

2. Generate time-varying covariates on a fine grid of $[0, \tau]$. For $t \in \{0, \Delta s, 2\Delta s, ..., \tau\}$ with $\Delta s = 0.01$, $i \in \{1, 2, ..., n\}$, we generate random variables $\alpha_{i1}, \dots, \alpha_{i5}$ independently from a Uniform $(0, 1)$ distribution, and q_i independently from a Uniform $(0, \tau)$ distribution, and construct two time-varying covariates as follows:

   $$\begin{aligned} x_{1i}(t) &= \alpha_{i1} + \alpha_{i2}\sin(2\pi t/\tau) + \alpha_{i3}\cos(2\pi t/\tau) + \alpha_{i4}\sin(4\pi t/\tau) + \alpha_{i5}\cos(4\pi t/\tau), \\ x_{2i}(t) &= \begin{cases} 0, & \text{if } t \le q_i; \\ 1, & \text{if } t > q_i. \end{cases} \end{aligned}$$

   The sample paths of both covariates are left-continuous step functions with right limit. We also generate three time-independent covariates:

   $$\begin{aligned} x_{3i} &\sim \text{Bernoulli}\,(0.6), \\ x_{4i} &\sim \text{Poisson}\,(2), \text{truncated at } 5, \\ x_{5i} &\sim \text{Beta}\,(2, 5). \end{aligned}$$

3. In *Setup 1*, the conditional hazard function is

   $$\lambda[t|x_i(t)] = \lambda_0(t) e^{2x_{1i}(t) + 2x_{2i}(t) + 2x_{3i} + 2x_{4i} + 2x_{5i}}, \tag{2.5}$$

and in *Setup 2*,

$$\lambda[t|x_i(t)] = \lambda_0(t)e^{2x_{1i}(t)^2+2x_{2i}(t)+2x_{3i}x_{4i}+2x_{5i}}. \tag{2.6}$$

Clearly, fitting the Cox model $\lambda[t|x_i(t)] = \lambda_0(t)\exp\{\beta_1 x_{1i}(t)+\beta_2 x_{2i}(t)+\beta_3 x_{3i}+\beta_4 x_{4i}+\beta_5 x_{5i}\}$ with data generated from (2.6) in Setup 2 will not yield desirable results.

4. Once covariates are generated, we numerically evaluate the conditional cumulative hazard function and the conditional survival function on the fine grid of survival time. Specifically, for $s \in \{0, \Delta_s, 2\Delta s, \dots, \tau\}$,

$$\begin{aligned} \Lambda[t|\widetilde{x}_i(t)] &= \Delta s \sum_{s\le t} \lambda_i[s|x_i(s)], \\ S[t|\widetilde{x}_i(t)] &= \exp\left\{-\Lambda_i[t|\widetilde{x}_i(t)]\right\}. \end{aligned}$$

5. For $i \in \{1, 2, ..., n\}$, we generate random variable u_i from a Uniform $(0, 1)$ distribution, then obtain the failure time by $t_i = \sup\{t : S_i[t|\widetilde{x}_i(t)] \ge u_i\}$.

6. We generate the censoring time c_i from an exponential distribution. Then we have $y_i = t_i \wedge c_i$ and $\delta_i = I(t_i \le c_i)$. The parameter of the exponential distribution is chosen to yield a censoring rate around 20% in setup (2.5), and about 50% in setup (2.6).

For each simulation setup, we independently generate a training set and a validation set with equal sample size, then fit our model using neural networks. We refit the model by swapping training and validation sets, and take the average as our estimator. For the Cox regression, we maximize the partial likelihood using all data. We repeat the process for N independent data sets, and calculate the average and sample variance of these N estimates at each time point on the fine grid for another set of randomly generated covariates. Finally, we plot the sample average of conditional survival curves estimated by neural networks together with the empirical confidence band, the average conditional survival curves estimated from the Cox regression, and the true conditional survival curve for a comparison.

The simulation results illustrated in both Figure 2.2 and Figure 2.3 are based on a sample size of $n = 3000$ (1500 for training and 1500 for validation) with 100 repetitions, where the curves for 9 different sets of covariates are presented. The green dashed line is the average estimated curve by using the partial likelihood method, the orange dash-dot line is the average estimated curve by our proposed neural networks method, and the black solid line is the truth curve. The dotted orange curves are the 90% confidence band obtained from the 100 repeated simulation runs using the proposed method. From Figure 2.2 we see that when the Cox model is correctly specified, both the partial likelihood method and our proposed neural networks method perform well, with estimated survival curves nicely overlapping with the corresponding true curves. When the Cox model is misspecified, Figure 2.3 shows that the partial likelihood approach yields severe biases, whereas the proposed neural network method still works well with a similar performance to that in Setup 1 shown in Figure 2.2 even with a higher censoring rate.

2.4 Real-World Data Sets

There are five real-world data sets analyzed by [40]. We re-analyze all these data sets using our method and compare with [40], except one data set that contains too many ties for which a discrete survival model would be more appropriate. Theses four data sets are: the Study to Understand Prognoses Preferences Outcomes and Risks of Treatment (SUPPORT), 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).

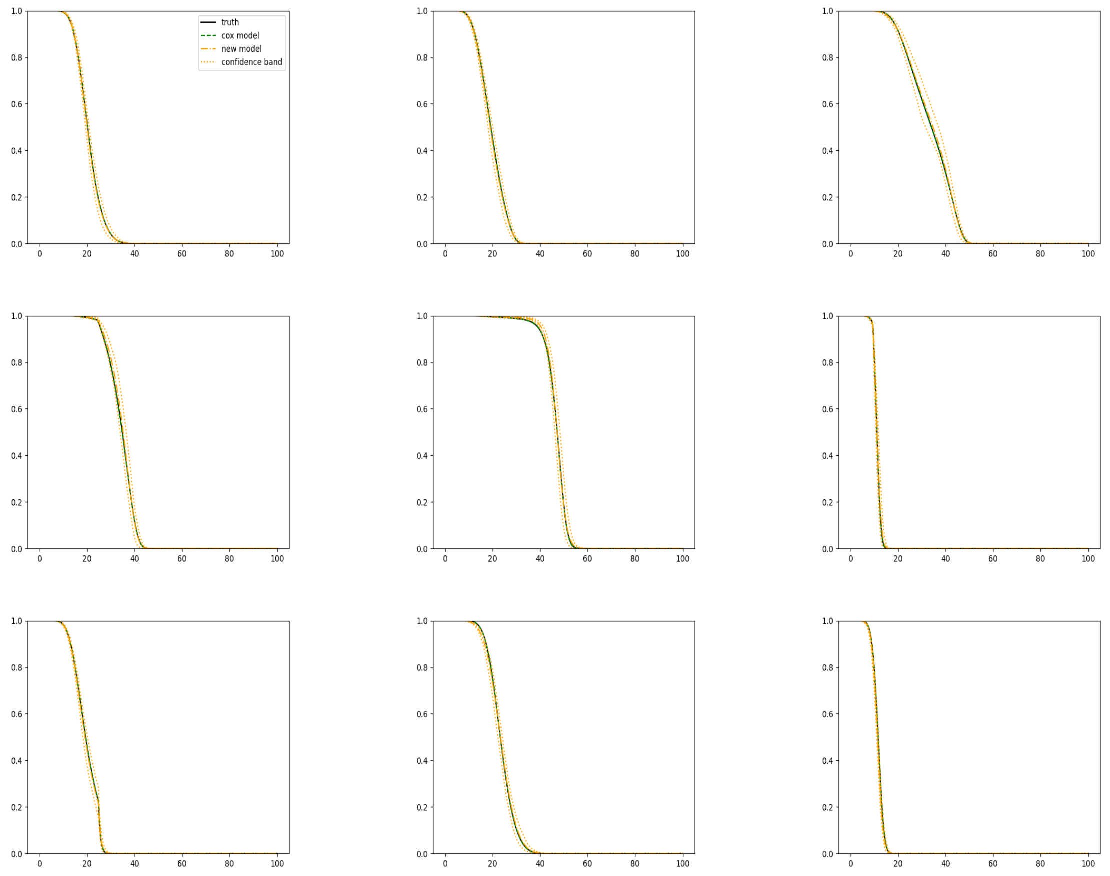


Figure 2.2: Conditional survival curves for 9 different sets of covariates when the Cox model is corrected specified (censoring rate around 20%).

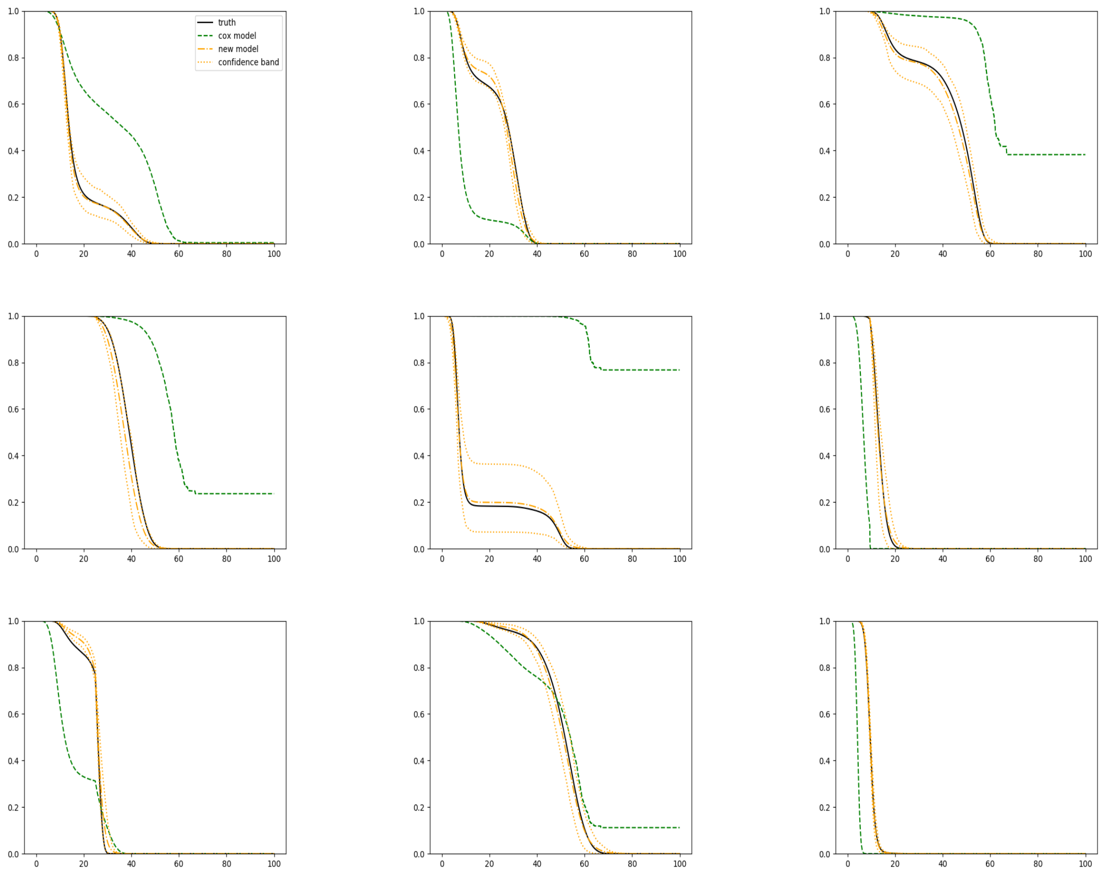


Figure 2.3: Conditional survival curves for 9 different sets of covariates when the Cox proportional hazards assumption is violated (censoring rate around 50%).

The first three data sets are introduced and preprocessed by [33]. The fourth data set is from the survival package of R ([58]) and preprocessed by [40]. These four data sets have sample sizes of a few thousand and the covariate numbers range from 7 to 14. The covariates in these data sets are all time-independent.

To compare with their method, we use the same 5-fold cross-validated evaluation criteria described in [40], including concordance index (C-index), integrated Brier score (IBS) and integrated binomial log-likelihood (IBLL). The time-dependent C-index [1] estimates the probability that the predicted survival times of two comparable individuals have the same ordering as their true survival times,

$$\text{C-index} = P[\widehat{S}(T_i|x_i) < \widehat{S}(T_i|x_j)|T_i < T_j, \Delta_i = 1].$$

The generalized Brier score [22] can be interpreted as the mean squared error of the probability estimates. To account for censoring, the scores are weighted by inverse censoring survival probability. In particular, for a fixed time t,

$$BS(t) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{\widehat{S}(t|x_i)^2 I(Y_i \leq t, \Delta_i = 1)}{\widehat{G}(Y_i)} + \frac{\left[1-\widehat{S}(t|x_i)\right]^2 I(Y_i > t)}{\widehat{G}(t)}\right\}.$$

where $\widehat{G}(t)$ is the Kaplan-Meier estimate of the censoring time survival function. The binomial log-likelihood is similar to the Brier score,

$$BLL(t) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{\log\left[1-\widehat{S}(t|x_i)\right] I(Y_i \leq t, \Delta_i = 1)}{\widehat{G}(Y_i)} + \frac{\log\left[\widehat{S}(t|x_i)\right] I(Y_i > t)}{\widehat{G}(t)}\right\}.$$

The integrated Brier score IBS and the integrated binomial log-likelihood score IBLL are calculated by numerical integration over the time duration of the test set.

The results of our method are summarized in Table 2.2, together with the results of [40] for

	C-Index			IBS			IBLL		
	a	b	c	a	b	c	a	b	c
SUPPORT	0.613	0.629	0.609	0.213	0.212	0.195**	-0.615	-0.613	-0.574**
METABRIC	0.643	0.662	0.652*	0.174	0.172	0.166**	-0.515	-0.511	-0.496**
Rot.&GBSG	0.669	0.677	0.680**	0.171	0.169	0.176	-0.509	-0.502	-0.524
FLCHAIN	0.793	0.790	0.788	0.093	0.102	0.102*	-0.314	-0.432	-0.336*

Table 2.2: Comparisons of different methods (a: Cox-MLP (CC); b: Cox-Time; c: our new method) for analyzing four real data sets. The result of our method is marked with ** when it outperforms both Cox-MLP(CC) and Cox-Time, and is marked with * when it outperforms one of the models.

a comparison. For SUPPORT and METABRIC data, our model yields the best integrated brier score and integrated binomial log-likelihood.

For Rot.&GBSG data, our model has the best C-index. The other results are comparable to that from the [40]. Note that in the 5-fold cross validation procedure, we use the set-aside data only as the test set for evaluation of the criteria and the rest of the data for training and validation of the neural networks, whereas [40] use the set-aside data as both the test set and the validation set which would lead to more favorable evaluations.

Chapter 3

Conditional Distribution Function Estimation Using Neural Networks for Continuous Response with Multiple Covariates

3.1 Introduction

In Chapter 2, we discuss the method of using neural network to estimate the conditional hazard function for censored survival data. We can see that an estimator of the conditional hazard function yields an estimator of the conditional survival function, hence equivalently the conditional distribution function, on the support of censored survival time given covariates. When there is no censoring, the problem naturally reduces to a general regression analysis, where the conditional distribution function is of interest. With slight modifications, our method for censored survival data can be adapted to uncensored data in a straightforward way. This expands the scope of the current literature on neural networks that primarily focuses on certain characteristic of the conditional distribution, for example, the conditional mean that corresponds to the mean regression. Once we obtain an estimator of the conditional distribution function, we can easily calculate the conditional mean given covariates, which provides a robust alternative approach to the mean regression using the L_2 loss. Note that the mean regression requires a basic assumption that the error term is uncorrelated with any of the covariates, which can be easily violated if some important covariate is unknown or unmeasured and correlated with some measured covariate. Our likelihood estimating approach, however, does not need such an assumption.

3.2 The Generalized Estimating Method

If there is no censoring, then $\delta_i = 1$ for all $i \in \{1, \ldots, n\}$ in the log likelihood function (2.3). Now consider an arbitrary continuous response variable $Y \in (-\infty, \infty)$ that is no longer "time." Note that the time variable $T \in [0, \infty)$. We are interested in estimating $F(y|x)$, the conditional distribution function of Y given covariates $X = x$, where X is a random vector. Since there is no time component, covariates are no longer "time-varying."

Assume $\{Y_i, X_i\}$, $i = 1, \ldots, n$, are i.i.d. Denote the observed data as $\{y_i, x_i\}$, $i = 1, \ldots, n$. We generalize the idea of using hazard function in survival analysis to estimate $F(y|x)$ for an arbitrary continuous random variable Y. Again, let $\lambda(t|x_i) = e^{h(t,x_i)}$. Then the conditional cumulative hazard function becomes

$$\Lambda(t|x_i) = \int_{-\infty}^{t} \lambda(s|x_i)ds = \int_{-\infty}^{t} e^{h(s,x_i)}ds,$$

i	start	stop	δ_{ij}	covariates
1	$t_0 = -\infty$	t_1	0	x_1
1	t_1	t_2	0	x_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	...	y_1	1	x_1
2	$t_0 = -\infty$	t_1	0	x_2
2	t_1	t_2	0	x_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
2	...	y_2	1	x_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 3.1: The expanded data set for estimating the conditional distribution function of a continuous response variable, where $(t_1, \dots, t_n)$ are sorted values of $(y_1, \dots, y_n)$ from the training set.

which gives the conditional distribution function

$$F(t|x_i) = 1 - \exp\left[-\Lambda(t|x_i)\right]. \tag{3.1}$$

Hence, the log likelihood function is given by

$$\ell_n = \sum_{i=1}^{n} \left[h(y_i, x_i) - \int_{-\infty}^{y_i} e^{h(t,x_i)} dt \right]. \tag{3.2}$$

Note that the above log likelihood has the same form as (2.3) except that the covariates are not time-varying, $\delta_i = 1$ for all i, and integrals start from $-\infty$. As a way of evaluating integrals in the log likelihood, the expanded data structure in Table 2.1 can be useful in estimating $h(y, x)$ with slight modifications given in Table 3.1.

To be numerically tractable, we assign $1/n$ to be the value of the distribution function at t_1, in other words, we make $F(t_1|x_i) = 1/n$, which is the empirical probability measure of Y at t_1. Thus $\Lambda(t_1|x_i) = -\log(1 - 1/n)$. Letting $\delta_{ij} = I(t_j = y_i)$ and evaluating the integrals in (3.2) on grid points $(t_1, \dots, t_n)$ that are sorted values of $(y_1, \dots, y_n)$, we obtain the following loss function:

$$\begin{aligned} loss(h) &= \frac{1}{n}\sum_{i=1}^{n}\left\{-\log(1-1/n)+\sum_{j=2}^{n} I(t_j \le y_i)\left[e^{h(t_j,x_i)}(t_j-t_{j-1})-h(t_j,x_i)\delta_{ij}\right]\right\} \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=2}^{n} I(t_j \le y_i)\left[e^{h(t_j,x_i)}(t_j-t_{j-1})-h(t_j,x_i)\delta_{ij}\right] + \text{Constant}. \end{aligned} \tag{3.3}$$

Once an estimator of h, denoted by $\widehat{h}$, is obtained, the conditional distribution function (3.1) can be estimated by

$$\widehat{F}(y|x) = I(t_1 \le y)\left\{1-\frac{n-1}{n}\exp\left[-\sum_{j=2}^{n} I(t_j \le y)e^{\widehat{h}(t_j,x)}(t_j-t_{j-1})\right]\right\}. \tag{3.4}$$

Remark: If the support of the continuous response variable has a fixed finite lower bound, then the integration for the conditional cumulative hazard function is the same as that for survival data. In other words, there is no need to assign a point mass of $1/n$ at t_1.

Similar to what we discuss in Chapter 2, we can estimate the arbitrary function $h(t, x_i)$ by minimizing the loss function (3.3) using neural networks. We then obtain the estimated conditional distribution function from Equation (3.4). The input of neural networks is (t_{j-1}, t_j, x_i) in each row of Table 3.1, and the output is $\widehat{h}(t, x_i)$. Note that the first row for each i in Table 3.1 is excluded from the calculation (since we manually assign a point mass of $1/n$ at t_1). The network structure and the hyperparameters setup are the same as discussed in Chapter 2: In the simulations, the number of nodes in each hidden layer is 64, the batch size is 100, and the initial learning rate is 0.001. In real-world data examples, we tune the hyperparameters from the set of all combined values with the number of nodes in each hidden layer taken from $\{64, 128, 256\}$, the initial learning rate from $\{0.1, 0.01, 0.001, 0.0001\}$, and the batch size from $\{64, 128, 256\}$.

3.3 Simulations

For uncensored continuous outcomes, the traditional neural network method with the commonly used L_2 loss function gives the conditional mean estimator. Then the conditional distribution function given a set of covariate values can be estimated by shifting the center of the empirical distribution of training set residuals to the estimated conditional mean. This would yield a valid estimator under the assumption that the errors (outcomes subtract their conditional means) are i.i.d. and uncorrelated with conditional means. We will evaluate the impact of this widely imposed condition for the mean regression via simulations. On the other hand, an estimator of the conditional distribution function gives a conditional mean estimator as follows:

$$\int_{-\infty}^{\infty} t d\widehat{F}(t|x) = \sum_{i=1}^{n} t_i \left(\widehat{F}(t_i|x) - \widehat{F}_k(t_{i-1}|x) \right).$$

Thus, we will compare our method to the method with L_2 loss on the estimation of the conditional distribution function as well as the estimation of the conditional mean.

In the following simulation studies, we consider i.i.d. data generated from the following model:

$$y_i = x_{1i}^2 + x_{2i}x_{3i} + x_{3i}x_{4i} + x_{5i} + \epsilon_i g(x_i),$$

$i = 1, \ldots, n$, where x_i denotes the i-th vector of all covariates, ϵ_i is mean-zero given all covariates, and g is a function of covariates, so $\epsilon_i g(x_i)$ is the i-th error term with zero-mean. We consider two simulation setups. In the first setup, the error is uncorrelated with the mean and has constant variance. In the second setup, the error is correlated with the mean and has non-constant variance. We would expect our new method either performs similarly or outperforms the method with L_2 loss since our loss function is based on the nonparametric likelihood function that is free of any model assumption. Specifically, covariate values $x_{1i}, \ldots, x_{5i}$ are generated from the following distributions:

$$\begin{aligned}
x_{1i} &\sim \mathrm{N}\,(0,1),\ \text{truncated at}\ \pm 3,\\
x_{2i} &\sim \mathrm{Uniform}\,(0,1),\\
x_{3i} &\sim \mathrm{Beta}\,(0.5,0.5),\\
x_{4i} &\sim \mathrm{Bernoulli}\,(0.5),\\
x_{5i} &\sim \mathrm{Poisson}\,(2),\ \text{truncated at}\ 5.
\end{aligned}$$

The two setups are:

Setup 1 (uncorrelated error with constant variance): generate another covariate $x_{6i} \sim \mathrm{N}\,(1,1)$ independently, then generate $\epsilon_i \sim$ a mixture distribution of $\mathrm{N}\,(-2,1)$, $\mathrm{N}\,(0,1)$, and $0.5x_{6i}^2$ with mixture probabilities $(0.1, 0.7, 0.2)$, and let $g(x_i) = c$, where c is a constant.

Setup 2 (correlated error with non-constant variance): generate another covariate $x_{6i} \sim \mathrm{N}\,(1 + 0.5x_{1i}, 0.75)$, such that

$$\begin{pmatrix} x_{1i} \\ x_{6i} \end{pmatrix} \sim \mathrm{N}\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}\right),$$

then generate $\epsilon_i \sim$ a mixture distribution of $\mathrm{N}\,(-2,1)$, $\mathrm{N}\,(0,1)$, and $0.5x_{6i}^2$ with mixture probabilities $(0.1, 0.7, 0.2)$, and let $g(x_i) = cx_{1i}^2$, where c is a constant.

Note that different values of constant c yield different signal to noise ratios in both setups.

For each setup, we generate independent training and validation data sets with equal sample size, then fit both models with our general loss given in (3.3) and the L_2 loss using the same neural network architecture. Figure 3.1 and Figure 3.2 illustrate the comparisons of estimated conditional distribution functions given 9 different sets of covariates between these two methods with a sample size of 5000 and 100 replications. In these figures, the black solid curve represents the true conditional distribution function, the green dashed curve represents the estimated conditional distribution function using L_2 loss, and the orange dash-dot curve represents the estimated conditional distribution using our method. The orange dotted curves are the 90% confidence band estimated using our method from the 100 repeated experiments. Figure 3.1 illustrates that when the error is uncorrelated with the covariates and has constant variance (Setup 1), both methods perform well in estimating the conditional distribution functions. When the error becomes correlated with the covariates and has non-constant variance (Setup 2), the traditional neural network method using L_2 loss fails, which is illustrated in Figure 3.2.

Further more, we compare the conditional mean estimates of both methods under two different sample sizes ($n = 1000$ and $n = 5000$) and two different magnitudes of noises ($c = 0.5$ and $c = 1$). We evaluate the performance of both methods by averaging the mean and median squared prediction errors, respectively, of 500 independently generated test data points over 100 replications, and summarize the results in Table 3.2. The averaged coverage rates of 90% and 95% predictive intervals obtained using our method are also presented in Table 3.2. These coverage rates demonstrate the performance of our distribution estimation, as they reflect the proportion of times that the outcome values in the test set fall within their corresponding predictive intervals. In Setup 1, both methods have similar mean squared prediction error, and our method yields slightly smaller median squared prediction error. In Setup 2, our model yields slightly better mean squared error, and significantly better median squared error. Our model provides reasonable prediction coverage rates in both setups, with improved performance as the sample size increases.

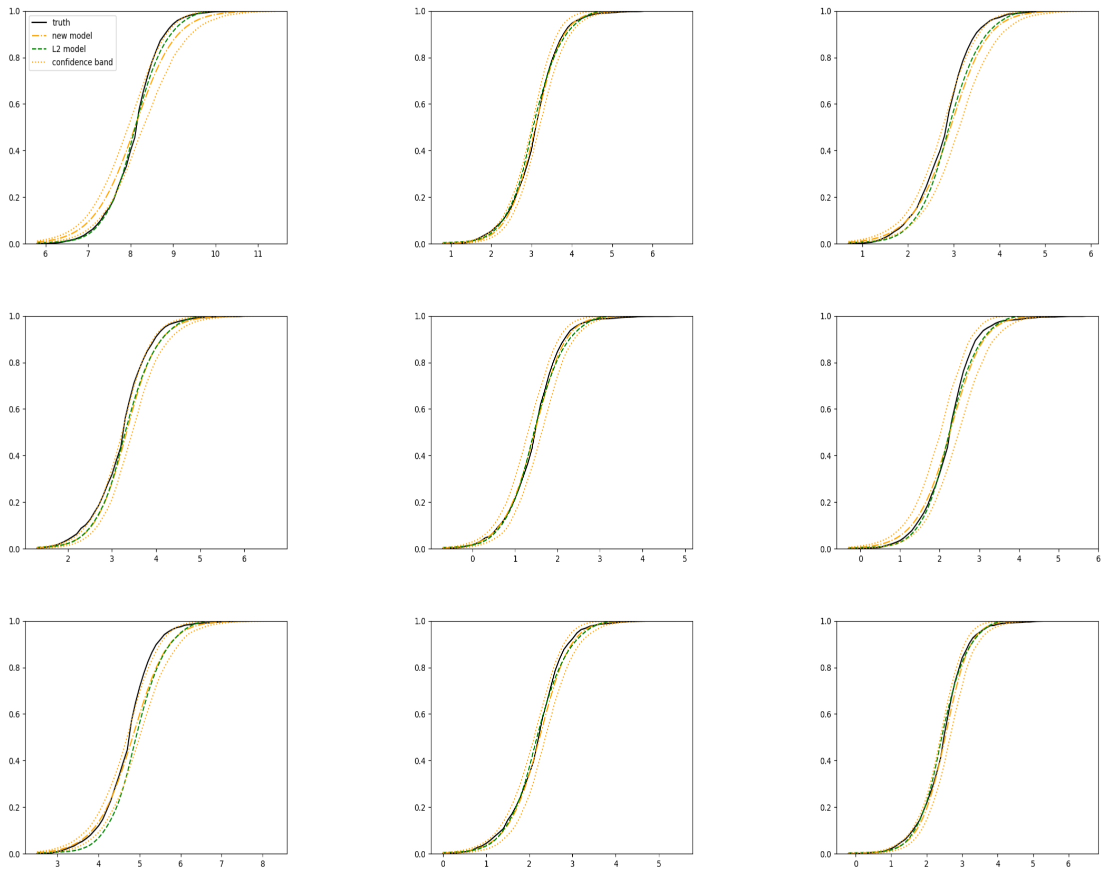


Figure 3.1: Conditional distribution functions given 9 different sets of covariate values for uncensored data generated in Setup 1 with c=0.5.

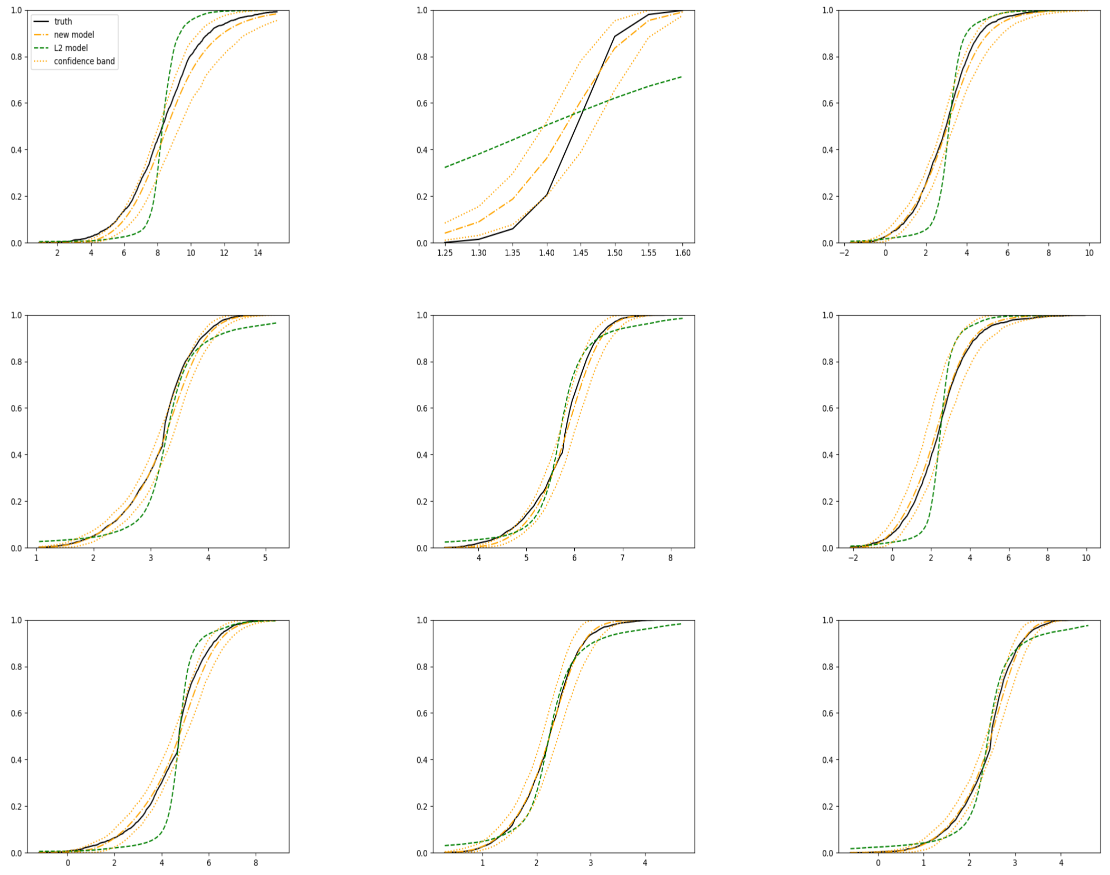


Figure 3.2: Conditional distribution functions given 9 different sets of covariate values for uncensored data generated in Setup 2 with c=0.5.

Sample size n	1000		5000	
Setup 1				
c	0.5	1	0.5	1
L_2 method mean squared error	0.50	1.83	0.45	1.73
new method mean squared error	0.51	1.84	0.44	1.73
L_2 method median squared error	0.17	0.59	0.14	0.52
new method median squared error	0.16	0.56	0.13	0.51
new method 90% coverage rate	0.90	0.90	0.90	0.90
new method 95% coverage rate	0.95	0.95	0.95	0.96
Setup 2				
c	0.5	1	0.5	1
L_2 method mean squared error	1.79	6.90	1.65	6.47
new method mean squared error	1.74	6.84	1.58	6.34
L_2 method median squared error	0.09	0.27	0.04	0.13
new method median squared error	0.02	0.08	0.01	0.05
new method 90% coverage rate	0.87	0.87	0.91	0.91
new method 95% coverage rate	0.91	0.91	0.95	0.94

Table 3.2: Average mean/median squared errors of both methods and the prediction coverage rate of the new method over 100 replications.

Target probability	0.10	0,20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
Setup 1	0.11	0.20	0.30	0.40	0.51	0.62	0.72	0.82	0.91
Setup 2	0.09	0.18	0.29	0.40	0.52	0.63	0.74	0.83	0.92

Table 3.3: Average empirical cumulative probabilities of 100 repeated test sets at several estimated percentiles ($n = 1000$, $c = 0.5$).

Additionally, we provide more detailed evaluation of the performance of conditional distribution estimates using test data by comparing sample proportions (empirical cumulative probabilities) at several estimated percentiles to corresponding population cumulative probabilities. We consider two cases in the above simulations: Setup 1 and Setup 2 when $n = 1000$ and $c = 0.5$. In each case, for a considered (target) probability we obtain the estimated percentile for each individual in the test set, then calculate the frequency of individuals with response values less than or equal to their estimated percentiles. We average the results over 100 independent replications, which are summarized in Table 3.3. From Table 3.3 we see that our method provides reasonably accurate estimates of the conditional distribution functions for a moderate sample size.

3.4 Comparisons to Kernel Methods for Uncensored Data

It is advantageous of using neural networks for estimating the conditional distribution when there are several continuous covariates. For low dimensional continuous covariates, especially 1-D or 2-D, kernel methods can be applied for estimating conditional distributions [24]. We first consider the following example of [24] with 1-D covariate to compare our method with the Nadaraya-Watson (NW) estimator:

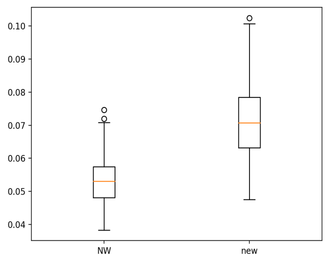
$$Y_i = 2\sin(3.1416X_i) + \epsilon_i,$$

where $\{X_i\}$ and $\{\epsilon_i\}$ are two independent sequences of independent random variables having a common distribution with density $1 - |x|$ on $[-1, 1]$. Following [24], we estimate the conditional distribution function of Y given X on a regular grid defined by steps .067 and .054 in x- and y-axes, and evaluate the performance of estimators via mean absolute deviation error (MADE), which is defined as follows:

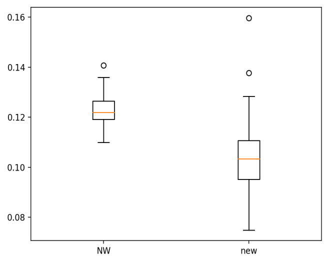
$$\text{MADE} = \frac{\sum_i |\hat{F}(y_i|x_i) - F(y_i|x_i)| I(.001 \leq F(y_i|x_i) \leq .999)}{\sum_i I(.001 \leq F(y_i|x_i) \leq .999)}$$

For the NW approach, we select bandwidth via a 5-fold cross-validation using mean square errors. For the neural network method, we use the fixed hyperparameters discussed in Section 2.2.4. Other combinations of hyperparameters yield similar results. The boxplots of MADEs in Figure 3.3a illustrates the results of simulations with a sample size of 1000 over 100 replications. We can see that NW method yields slightly better MADEs.

We further examine the case with two continuous covariates. Specifically, we add another



(a)



(b)

Figure 3.3: Boxplots of MADEs for the NW method and our proposed method. (a) 1-D covariate, n = 1000, (b) 2-D covariates, n = 1000.

random covariate to the above mean structure:

$$Y_i \;=\; 2\sin(3.1416X_{1i}) + X_{2i} + \epsilon_i,$$

where $\{X_{1i}\}$ and $\{\epsilon_i\}$ are still from a distribution with density $1-|x|$ on $[-1,1]$, and $\{X_{2i}\}$ is from uniform(-1,1). We estimate the conditional distribution function on a grid defined by steps .067, .067 and .054 in x_1-, x_2- and y-axes. The boxplots of MADEs obtained from 100 simulations with a sample size of 1000 are presented in Figure 3.3b, which show that the NW method loses its edge when the covariate dimension increases. It is well-known that going to any higher dimension will be difficult for kernel methods.

3.5 Real-World Data Sets

We use QSAR Fish Toxicity data set [8] and Airfoil Self-Noise data set [5] to illustrate our method for uncensored data. QSAR Fish Toxicity data set is collected for developing quantitative regression models to predict acute aquatic toxicity towards the fish Pimephales promelas (fathead minnow) on a set of 908 chemicals. Six molecular descriptors (representing the structure of chemical compounds) are used as the covariates and the concentration that causes death in 50% of test fish over a test duration of 96 hours, called LC_{50} 96 hours (ranges from 0.053 to 9.612 with a mean of 4.064) was used as model response. The 908 data points are curated and filtered from experimental data. The six molecular descriptors come from a variable selection procedure through genetic algorithms. In their original research article, the authors used a k-nearest-neighbours (kNN) algorithm to estimate the mean. The data set can be obtained from the machine learning repository of the University of California, Irvine (`https://archive.ics.uci.edu/ml/datasets/QSAR+fish+toxicity`).

Airfoil Self-Noise data set is collected for developing a noise prediction model. Self-noise is defined as the noise generated when an airfoil passes through smooth non-turbulent inflow conditions. The data set contains 1503 samples and six variables. The response variable is scaled sound pressure level (dB) and the covariates are, frequency (Hz), angle of attack ($^\circ$), chord length (m), free-stream velocity (m/s), and suction side displacement thickness (m). The data set can also be obtained from the machine learning repository of the University of California, Irvine (`https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise#`).

We use 5-fold cross-validated R^2, mean squared error and median squared error to evaluate our method and the neural networks with L_2 loss on these two real-world data sets and summarize the results in Table 3.4. Our new method yields better prediction in all three criteria for both data sets. The large advantage of our method for the Airfoil Self-Noise data may be due to a very skewed distribution of the response variable. For each data set, predicted conditional distribution functions given two different sets of covariate values obtained by our method are presented in Figure 3.4 for an illustration.

	Fish		Airfoil	
	L_2 method	New method	L_2 method	New method
Mean squared error	0.86	0.82	19.76	8.52
Median squared error	0.22	0.20	8.24	2.38
R^2	0.59	0.61	0.60	0.82

Table 3.4: Conditional mean prediction results of the L_2 method and the new method

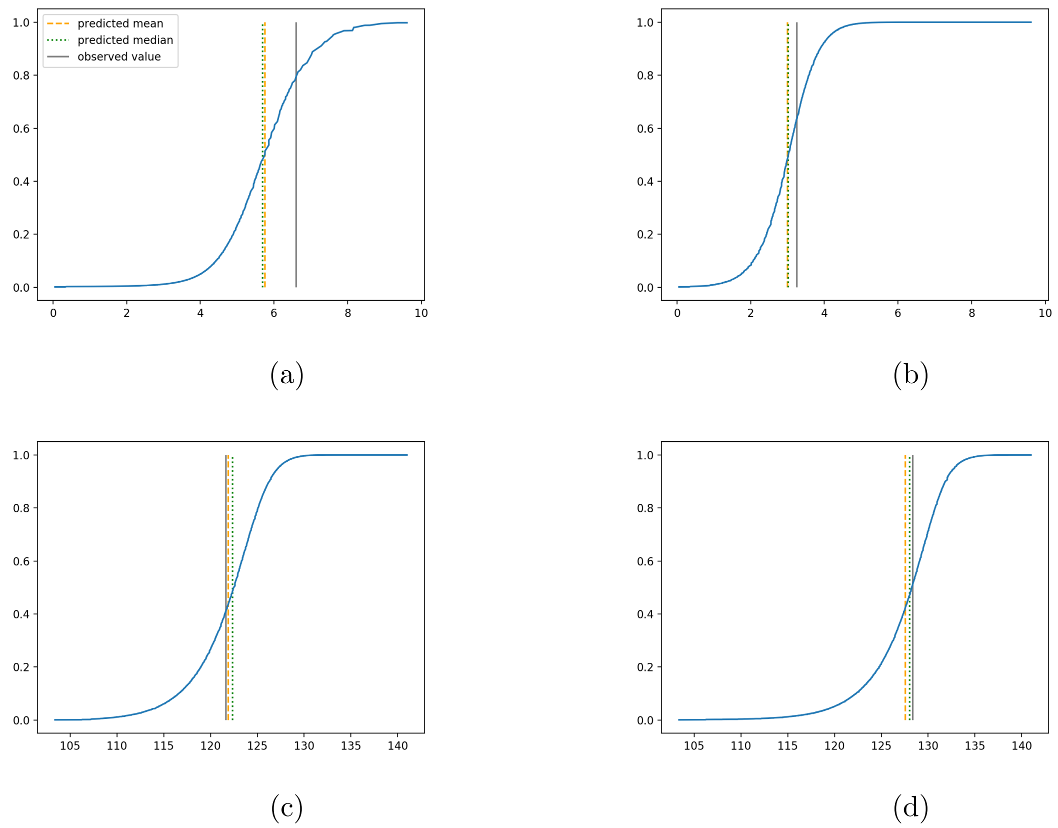


Figure 3.4: Estimated conditional distribution functions for 4 individuals. (a)(b): Fish Toxicity, (c)(d): Airfoil. The three vertical lines illustrate locations of the predicted mean, the predicted median and the observed value.

3.6 Numerical Comparison to ODE Neural Networks

The data expansion technique used in this work provides a simple and natural way of numerically evaluating the full likelihood based loss function. However, it seems that the data expansion would increase the effective sample size from n to n^2. This is not a concern for the survival problem with time-varying covariates (Chapter 2) because each covariate process needs to be observed at least at all distinct time points, leading to an order of n^2 number of distinct data points. When covariates are random variables other than stochastic processes, the sample size is indeed n, thus there should be a large room for developing more efficient numerical approaches. In particular, a recent work by [57] comes to our attention, which combines neural networks with an ordinary differential equation (ODE) framework to estimate the conditional survival function given a set of baseline covariates, in other words, time-independent covariates. They use ODE solver to integrate the cumulative hazard function from an initial value and its derivative (the hazard function). In the following part of the section, we would discuss how to use the Neural ODE model to estimate the conditional distribution function for uncensored data and make a comparison of the ODE approach to our method.

The Neural ODE model is proposed by [9]. For a variable z, given its initial state $z(t_0)$ and derivative (continuous dynamics), any later state of this variable $z(t_1)$ can be calculated using an "ODE solver" such as Euler's method. The key idea is that the derivative can be parameterized by a neural network:

$$z'(t) = f(z(t), t; \theta).$$

When the loss function L is a function of the final state, it can be written as:

$$L(z(t_1)) = L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t; \theta)dt\right) = L(\text{ODESolver}(z(t_0), f, t_0, t_1, \theta)).$$

Then by minimizing the loss, the parameters θ can be learned. Based on adjoint sensitivity analysis [48], the Neural ODE model is able to avoid going into the "ODE solver" in back propagation, but use another ODE to calculate the gradient and update the parameters.

[57] apply this model to the conditional survival function estimation and name their approach the Survival Ordinary Differential Equation Networks (SODEN). Specifically, they consider the conditional hazard function and the cumulative hazard function as an ODE with fixed initial value:

$$\begin{aligned} &\Lambda_x'(t) = f(\Lambda_x(t), t; x, \theta), \\ &\Lambda_x(0) = 0, \end{aligned} \tag{3.5}$$

where the function f is modeled by a feed-forward neural network. The loss function is the negative log-likelihood:

$$l(\theta) = \sum_{i=1}^{n} \{\Lambda_{x_i}(y_i) - \delta_i \log f(\Lambda_{x_i}(y_i), y_i; x_i, \theta)\}. \tag{3.6}$$

Note that the loss function is based on the likelihood, which is similar to Equation (2.3). The only difference is that they calculate the cumulative hazard from the "ODE solver" instead of the direct numerical integration using the observed time partition. Note that SODEN of [57] only applies to the survival time that has a fixed lower limit at zero.

In order to apply ODE solver, observation time is rescaled so that all the individuals have the same time window. Let $H_i(t) = \Lambda_{x_i}(ty_i)$, then $H_i(1) = \Lambda_{x_i}(y_i)$. For each individual, Equation set (3.5) can be transformed to:

$$
\begin{aligned}
&H_i'(t) = f(H_i(t), ty_i; x_i, \theta) y_i, \\
&H_i(0) = 0.
\end{aligned}
\tag{3.7}
$$

The input of the neural network f is ty_i, $H_i(t)$ and x_i (the time point t is determined by the "ODE Solver"). Assume the covariate dimension is d, and denote

$$
M(t) = \begin{bmatrix} H_1(t) & y_1 & x_1 \\ H_2(t) & y_2 & x_2 \\ \dots & \dots & \dots \end{bmatrix}_{n*(d+2)},
$$

$$
\tilde{f}_M(t) = \begin{bmatrix} f(H_1(t), ty_1; x_1, \theta) y_1 & 0 & 0 \\ f(H_2(t), ty_2; x_2, \theta) y_2 & 0 & 0 \\ \dots & \dots & \dots \end{bmatrix}_{n*(d+2)}.
$$

Then,

$$
M(1) = \text{ODESolver}(M(0), \tilde{f}_M(t), t_0 = 0, t_1 = 1; \theta). \tag{3.8}
$$

From $M(1)$ and $\tilde{f}_M(1)$, we obtain $\Lambda_{x_i}(y_i)$ and $f(\Lambda_i(y_i), y_i; x_i, \theta)$ in the loss function (3.6).

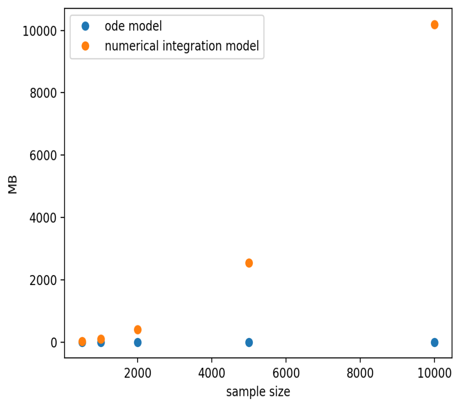
We can see that, the way SODEN deals with the covariates is to set the derivative of the covariates to be 0 in $\tilde{f}_M(t)$, so that the same covariate values pass into the neural network at each time step. If a covariate is time-varying, it is difficult to input the covariate history into this framework.

As for the general regression case discussed in this Chapter, the ODE framework in SODEN model can be adapted. Recall that we assign $1/n$ to be the value of the distribution function at the smallest observation point t_0, we can modify Equation Set (3.5) to

$$\begin{aligned} \Lambda_x'(t) &= f(\Lambda_x(t), t; x, \theta), \\ \Lambda_x(t_0) &= -\log(1 - 1/n), \end{aligned}$$

and rescale Λ_{x_i} by $H_i(t) = \Lambda_{x_i}(t(y_i - t_0) + t_0)$.

We have observed that the ODE method performs similarly comparing to our approach in terms of the mean/median square errors in the simulations described in Section 3.3. In terms of the memory usage, the ODE model is more efficient as it does not require data expansion. The time complexity of the ODE model is close to $O(n)$ based on experiments. However, the numerical comparisons show that our method is faster than the ODE method for a broad range of sample sizes (see Figure 3.5), indicating a memory-speed trade-off of our method and a large coefficient of the ODE complexity. For much larger sample sizes, we would expect the ODE method beats our method in speed (results not obtained due to the memory limitation). On the other hand, the ODE method may loss its advantage in estimating the survival function with time-varying covariates because of the needed data expansion to a size of n^2.



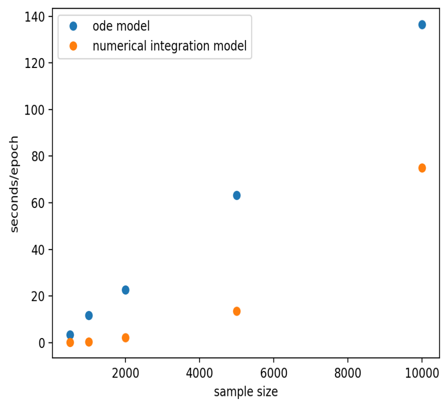


Figure 3.5: The memory and time cost vs. sample size for ODE model and our previous model

Chapter 4

Survival Estimation with Arbitrary Effects of Time-varying Covariates Using Neural Networks

4.1 Introduction

Survival prediction is of increasing importance and estimating the conditional survival function from censored data nonparametrically is of great interest. Recent advances in machine learning and deep learning have overcome certain limitations of traditional semiparametric survival analysis methods [13, 62]. These approaches include Gradient Boosting Machines (GBM) [28, 50] and Random Survival Forests (RSF) [30], which are ensemble methods based on survival trees [41]; DeepSurv [33] and DeepHit [19], which are based on neural networks. Some of them [28, 50, 33], although in a more flexible non-linear form, still keep the Cox model or the accelerated failure time model framework, thus still assume certain model structures. Also, they estimate a risk score that summarizes the features rather than estimating the conditional survival function directly. DeepHit divides the continuous time into a series of evenly partitioned time intervals and output the probability of an event occurring in each time interval. RSF is a flexible approach, which uses log-rank test to split the data and directly output the survival probabilities on observed time points.

The above methods are developed for time-independent covariates. Time-varying covariates, however, are also common in practice. To fit the Cox-PH model with time-varying covariates, the dataset needs to be expanded with each row representing a subject's records during a specific time interval formed by all the observed time points. The machine learning and deep learning methods mentioned above can potentially be extended to incorporate time-varying covariates using the same data expansion technique, that is, to treat each time interval as a separate observation with constant covariates. Such extensions for time-varying covariates, however, are still in their early stages and are relatively limited. In particular, [14] found that the computational cost is extremely high for RSF when dealing with time-varying covariates.

Following the same data expansion technique, we have proposed a non-parametric approach using neural networks to directly estimate the conditional survival function given time-varying covariates [29]. It takes time-varying covariates, baseline covariates and a corresponding time interval as the input of a feed forward neural network and outputs the logarithm of the conditional hazard function. The method uses a loss function determined by the full nonparametric likelihood, thus does not assume any specific model structure.

To the best of our knowledge, all the known methods for time-varying covariates, including those discussed above, assume that the effect of covariates on the conditional hazard function is instantaneous. That is, the instantaneous hazard given entire covariate paths only depends on values of covariates observed at the current time point. Such an assumption, however, can be rather limited and considering non-instantaneous effects of covariates on the hazard function can be interesting and important in addressing some realistic problems, for example, when hazard is related to an arbitrary cumulative exposure or certain delayed covariate effects. A nonparametric approach for non-instantaneous covariate effects provides the most flexible way of estimating how covariates influence survival, captures complex temporal dynamics and eliminates modeling biases in survival prediction.

In this work, we further generalize the method of [29] to account for arbitrary effects of time-varying covariates based on the entire covariate history. The time-varying covariates are functional inputs and the conditional survival function (or equivalently the conditional hazard function) becomes an operator. We propose to apply the recently developed deep operator network, the DeepONet [42], to estimate the unknown operator. To the best of our knowledge, our new method is the first to model arbitrary effects of time-varying covariates in estimating the conditional survival function with the fewest assumptions.

4.2 Methodology

Consider survival times of n subjects, which can be right-censored. We are interested in estimating the survival time ditribution given the covariates. For subject i, we denote the time-varying covariate vector as $X_i(t)$. We further denote the underlying failure time as T_i, and the underlying censoring time as C_i. We assume that the failure time possesses a Lebesgue density and that the censoring time is independent of failure time given covariates. Let the observed time be $Y_i = \min\{T_i, C_i\}$ and the failure indicator be $\Delta_i = I(T_i \leq C_i)$. The observations are independent and identically distributed (i.i.d.), and can be written as $\{Y_i, \Delta_i, \widetilde{X}_i(\cdot) : \; i = 1, \ldots, n\}$, where $\widetilde{X}_i(t)$ denotes the covariate history up to time t, that is, $\widetilde{X}_i(t) = \{X_i(s), 0 \leq s \leq t\}$. We assume each of the processes $X_i(t)$ has left continuous sample path with right limit. Let $f[t|\widetilde{X}_i(\infty)]$ be the conditional Lebesgue density function of T_i, $f_C[t|\widetilde{X}_i(\infty)]$ be the conditional density function of C_i, $S[t|\widetilde{X}_i(\infty)]$ be the conditional survival function of T_i, and $S_C[t|\widetilde{X}_i(\infty)]$ be the conditional survival function of C_i.

We further assume that future covariate values do not influence the current risk. In other words, the risk of an event occurring at time t only depends on the current and past values of covariates, not on future values, so the covariates are external [32]. The conditional hazard function of T_i can be written as:

$$\lambda\left[t\left|\widetilde{X}_i(\infty)\right.\right]=\lambda\left[t\left|\widetilde{X}_i(t)\right.\right]. \tag{4.1}$$

To meet the positivity constraint for the hazard function, we log transform the hazard function to ease the numerical implementations. Denote $h[t,\widetilde{X}_i(t)]=\log\lambda[t|\widetilde{X}_i(t)]$. Then the conditional cumulative hazard function given covariate history has the following form:

$$\Lambda\left[t\left|\widetilde{X}_i(\infty)\right.\right]=\Lambda\left[t\left|\widetilde{X}_i(t)\right.\right]=\int_0^t\lambda\left[s\left|\widetilde{X}_i(s)\right.\right]ds=\int_0^t e^{h[s,\widetilde{X}_i(s)]}ds.$$

The conditional survival function is given by

$$S\left[t\left|\widetilde{X}_i(\infty)\right.\right]=S\left[t\left|\widetilde{X}_i(t)\right.\right]=\exp\left\{-\Lambda\left[t\left|\widetilde{X}_i(t)\right.\right]\right\}=\exp\left\{-\int_0^t e^{h[s,\widetilde{X}_i(s)]}ds\right\}. \tag{4.2}$$

Equation (4.2) is always a valid survival function for any function $h[s,\widetilde{X}_i(s)]$. Once the function h is estimated, we can easily determine the conditional survival function estimator.

Remark: Kalbfleisch and Prentice [32] clearly distinguishes two different types of time-varying covariates: external and internal. An external covariate is the one that its future path is not affected by the occurrence of a failure at the current time. Any non-external covariate is an internal covariate. An internal covariate is typically taken on the individual subject over time, and its path is affected by the survival status. We assume that the covariates are external in this work. Note that the conditional survival probability is not well-defined if there is an internal covariate.

i	time point	covariates
1	t_1	$x_1(t_0), 0, \ldots, 0$
1	t_2	$x_1(t_0), x_1(t_1),\ 0,\ \ldots,\ 0$
$\vdots$	$\vdots$	$\vdots$
1	t_m	$x_1(t_0), x_1(t_1), ..., x_1(t_{m-1})$
2	t_1	$x_2(t_0), 0, \ldots, 0$
2	t_2	$x_2(t_0), x_2(t_1), 0, \ldots, 0$
$\vdots$	$\vdots$	$\vdots$
2	t_m	$x_2(t_0), x_2(t_1), ..., x_2(t_{m-1})$
$\vdots$	$\vdots$	$\vdots$

Table 4.1: The data structure with time varying covariate.

4.2.1 Data Structure and DeepONet

To estimate the function $h[s, \widetilde{X}_i(s)]$ nonparametrically using the DeepONet [42], we first need to discretize the time. We partition the time axis evenly into m intervals on grid points $(t_0 \equiv 0, t_1, t_2, ...t_m \equiv \tau)$, where τ is a finite upper bound of the considered support of the survival time, oftentimes taken to be the largest observed failure time in a data set in practice. Given observed data $\{y_i, \delta_i, x_i(\cdot)\}$, the input of the DeepONet is $[s, x_i(t_0), x_i(t_1), ..., x_i(t_{m-1})]^T$ where $s \in \{t_1, t_2, ..., t_m\}$. Note that $h[s, \widetilde{X}_i(s)] = h[s, \widetilde{X}_i(\tau)]$ for external covariates $X_i(\cdot)$, we aim to only input the history value of the covariates up to the time point s, and the future values are masked with 0. See equation (4.3) for the validity of such an approach. As shown in Table 4.1, each subject is expanded into m rows for input.

As for the neural network structure, the most straightforward way is to directly employ a classical network, such as fully connected feed forward neural networks (FNN) or convolutional neural networks (CNN), and concatenate $\widetilde{x}(s)$ and s together as the network input. Most of the related work uses such simple concatenatation [4, 19, 29, 31] as they only consider the covariate value at the current time point, i.e., $[s, x_i(s)]$. While we are dealing with the entire covariate history, treating each element in $[x_i(t_0), x_i(t_1), ...x_i(t_{m-1})]$ and s equally is not reasonable. Recently, using neural networks to approximate operators has drawn

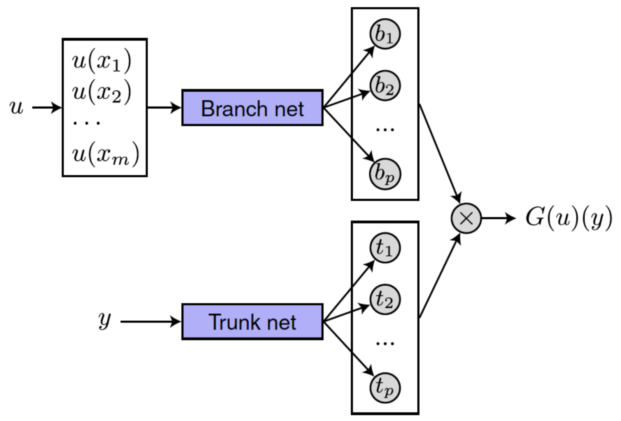


Figure 4.1: The unstacked DeepONet Structure. u, y, $G(u)(y)$ in this figure correspond to x, s, $h(x)(s)$ respectively.

increasing attention in the research literature [42, 38]. It is shown in [42] that the deep operator network can learn various explicit operators, such as integrals and fractional Laplacians, as well as implicit operators that represent deterministic and stochastic differential equations. Our problem can be seen as an operator learning problem if we treat $h[s, \widetilde{X}_i(s)]$ as an operator $h[\widetilde{X}_i(s)](s)$. Thus the neural operator structure can be applied. Specifically, we use an unstacked DeepONets structure. A DeepONet consists of two sub-networks, one for encoding the input function at a fixed number (m) of sensors $[x(t_0), x(t_1), ..., x(t_{m-1})]^T$ (branch net), and another for encoding the location s for the output function (trunk net) [42]. See Figure 4.1, where a different set of notation is used.

The branch and trunk architecture is inspired by the Universal Approximation Theorem for Operators [10]. The theorem states that two fully connected neural networks with a single hidden layer, combined by a vector dot product of the outputs, are able to approximate any continuous nonlinear operator with arbitrary accuracy. Specifically, suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$ and $K_2 \subset \mathbb{R}^d$ are compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, and G is a nonlinear continuous operator which maps V into $C(K_2)$. Here $C(K)$ is the Banach space of all continuous functions defined on K equipped with the uniform norm. For any $\epsilon > 0$, there are positive integers n, p, and m, constants c_i^k, ξ_{ij}^k, θ_i^k, $w_k \in \mathbb{R}$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, and $j = 1, \dots, m$, such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon \tag{4.3}$$

holds for all $u \in V$ and $y \in K_2$.

A generalized version of the approximation theorem (Theorem 2 in [42]) states that for any nonlinear continuous operator G and any $\epsilon > 0$, there exist positive integers m, p, continuous vector functions $g : \mathbb{R}^m \to \mathbb{R}^p$, $f : \mathbb{R}^d \to \mathbb{R}^p$, and $x_1, x_2, \dots, x_m$, such that,

$$|G(u)(y) - \langle g[u(x_1), u(x_2), ..., u(x_m)], f(y) \rangle| < \epsilon \tag{4.4}$$

holds for all u and y, where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathbb{R}^p$. The functions g and f can be chosen as diverse classes of neural networks, for example, fully connected neural networks, residual neural networks and convolutional neural networks.

4.2.2 Full Likelihood and Discretized Loss

We propose to use the full likelihood to build the loss function. Given the observations $\{y_i, \delta_i, x_i(\cdot)\}$, the full likelihood function is

$$\begin{aligned} L_n &= \prod_{i=1}^{n} \{f[y_i|\widetilde{x}_i(y_i)] S_C[y_i|\widetilde{x}_i(y_i)]\}^{\delta_i} \{f_C[y_i|\widetilde{x}_i(y_i)] S[y_i|\widetilde{x}_i(y_i)]\}^{1-\delta_i} \\ &\propto \prod_{i=1}^{n} \lambda[y_i|\widetilde{x}_i(y_i)]^{\delta_i} S[y_i|\widetilde{x}_i(y_i)] \\ &= \prod_{i=1}^{n} \exp\{h[\widetilde{x}_i(y_i)](y_i)\delta_i\} \exp\left\{ -\int_0^{y_i} e^{h[\widetilde{x}_i(t)](t)} dt \right\}. \end{aligned}$$

The log likelihood is given by

$$\ell_n = \sum_{i=1}^{n} \left\{ h[\widetilde{x}_i(y_i)](y_i)\delta_i - \int_0^{y_i} e^{h[\widetilde{x}_i(t)](t)} dt \right\}. \tag{4.5}$$

By calculating the above integrals numerically using the summation on a partition of $[0, \tau]$ with m subintervals, we obtain the loss function that is the discretized negative log likelihood, scaled by the sample size:

$$loss(h) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} I(t_{j-1} \leq y_i) \left\{ e^{h[\tilde{x}_i(t_{j-1})](t_{j-1})}(t_j - t_{j-1}) - h[\tilde{x}_i(t_{j-1})](t_{j-1})\delta_{ij} \right\}, \tag{4.6}$$

where $\delta_{ij} = I(t_j \geq y_i)\delta_i$. Note that δ_{ij} is always 0 until the event happens.

4.2.3 Details in Network Architectural and Hyperparameter Tuning

The implementation of DeepONet is provided in the python package DeepXDE [43]. The package is originally used for solving ordinary and partial differential equations. It supports five tensor libraries as backends. To adapt the method to our problem, we modified the code in DeepXDE using its TensorFlow 2.x backend. Specifically, we use our customized loss function and modify the branch net structure for the CNN method. We trained the neural networks on a system equipped with an AMD Ryzen 9 8945HS processor with Radeon 780M Graphics at 4.00 GHz and 32GB of RAM.

For trunk net, since we only have one dimensional time point input, we use the simple one layer FNN structure. For branch net, we try both FNN and CNN to summarize the time-varying covariates. The FNN is simply composed of two fully connected hidden layers (a.k.a. dense layer). The CNN structure is composed of two blocks of 1D convolutional layers [36] and max pooling layers, followed by one fully connected layer. In simulations, we have one time-varying covariate and two extra time-independent covariates. When we use FNN structure, we concatenate the time-independent covariates to the time-varying ones directly, so the input dimension is $m + 2$. When we use CNN structure, it is not reasonable to concatenate two types of covariates. So we first input the time-varying covariates into CNN and then concatenate the time-independent ones with the CNN output before going through the final fully connected layer. Since the branch net requires fix length of the input, we mask the future value of the time-varying covariates with 0 (see Table 4.1). The masked values will not contribute to the output of the FNN branch net because the multiplication with those future time input-weights will be 0, regardless of the weight values, see equation (4.3). To ensure that it does not contribute to the output in the case of using CNN, we use casual padding [60] that automatically neglects the future time steps. We expect the CNN branch net to perform better because it uses temporal neighborhood information of the time-varying covariates.

We use a validation dataset to decide when to stop during training and use another independent test dataset to tune the hyperparameters. Specifically, in each training, we stop when the validation loss no longer decreases, and we choose the hyperparameter combination with the smallest test loss. The hyperparameters are chosen from the following sets:

number of nodes in each dense layer: $[32, 64, 128, 256]$
number of filters in each Conv1D layer: $[16, 32, 64]$
pool size: $[4, 8]$
learning rate: $[0.01, 0.001, 0.0001]$
batch size: $[100, 500, 1000]$

In the following simulations, the number of nodes in each dense layer (except for the last layer in branch and trunk net) is 128, number of filters in each Conv1D layer is 16, pool size is 8 with the same stride, learning rate is 0.001 and batch size is 1000. "Relu" function is used as the activation function between the hidden layers, and linear function is used for the final output so that the output value is not constrained. "Adam" is used as the optimizer. In the last layer before doing dot product, we use $p = 10$ nodes as suggested in [42].

When we use FNN as the branch net, the number of partition m is crucial to the model performance. In the following simulations we try $m = 100$, 200, 300, 400, 500 with the other hyperparameters fixed. When sample size $n = 500$, in 5 out of 10 experiments, we obtain the minimum test loss at $m = 200$, and the median m value for the best test loss is 250. We further increase the sample size to 2000, and observe that the optimal m value is still around 250. Thus, we use $m = 250$ in all the simulations in the next subsection.

4.3 Simulations

We generate data from a model where covariate history has a cumulative effect on the conditional hazard function. The Cox-PH model that assumes $\lambda[t|\widetilde{X}_i(t)] = \lambda[t|X_i(t)]$ is misspecified in this setup.

First, we generate a time-varying covariate on a fine grid of $[0, 100]$. For $t \in \{0, \Delta s, 2\Delta s, ..., 100\}$ with $\Delta s = 0.1$, $i \in \{1, 2, ...n\}$, we generate random variables $\alpha_{i1}, \dots, \alpha_{i5}$ independently from a Uniform $(0, 1)$ distribution and construct a time-varying covariate as follows:

$$x_i(t) \quad = \quad \alpha_{i1} + \alpha_{i2}\sin(2\pi t/\tau) + \alpha_{i3}\cos(2\pi t/\tau) + \alpha_{i4}\sin(4\pi t/\tau) + \alpha_{i5}\cos(4\pi t/\tau).$$

The sample paths of the covariate is left-continuous step functions with right limit. Then we generate two time-independent covariates $z_i \sim$ Bernoulli(0.5), and $w_i \sim$ Normal$(0, 1)$. The binary covariate z can be seen as a treatment assignment variable.

The conditional hazard function is generated by

$$\lambda\left(t\,|\widetilde{x}_i(t)\,,z_i,w_i\right)=0.05*\exp\left(w_i+z_i+0.01*\sum_{0}^{t}x_i(s)\Delta s+0.01*\sum_{0}^{t}x_i^2(s)z_i\Delta s\right).$$

We numerically evaluate the conditional cumulative hazard function and the conditional survival function on the fine grid of survival time. Specifically, for $s\in\{0,\Delta s,2\Delta s,\ldots,100\}$,

$$\begin{aligned}\Lambda(t|\widetilde{x}_i(t)) &= \Delta s\sum_{s\le t}\lambda_i(s|x_i(s)),\\ S(t|\widetilde{x}_i(t)) &= \exp\left\{-\Lambda_i(t|\widetilde{x}_i(t))\right\}.\end{aligned}$$

Note that, due to the discrete nature of the time variable, the exact survival function determined by the above hazard function should be expressed as a product integral. We find using the survival function as an exponential function of the cumulative hazard to approximate the exact survival function is adequate for simulation studies.

For $i\in\{1,2,...n\}$, we generate random variable u_i from a Uniform$\,(0,1)$ distribution, then obtain the failure time by $t_i=\sup\{t:S_i(t|\widetilde{x}_i(t))\ge u_i\}$. We generate the censoring time c_i from an Exponential$\,(50)$ distribution. Then we have $y_i=t_i\wedge(c_i\wedge 99)$ and $\delta_i=I(t_i\le c_i)$. The censoring rate is around 20% in this setup.

We independently generate training sets and validation sets, with the time-varying covariate taking value on m equispaced grid points. We fit our model using neural networks and then use the fitted model to estimate the conditional survival curves given 9 different sets of newly generated covariates. We repeat the process for $N=200$ times, plot the sample average and 90% confidence band of the estimated conditional survival curves. Figure 4.2

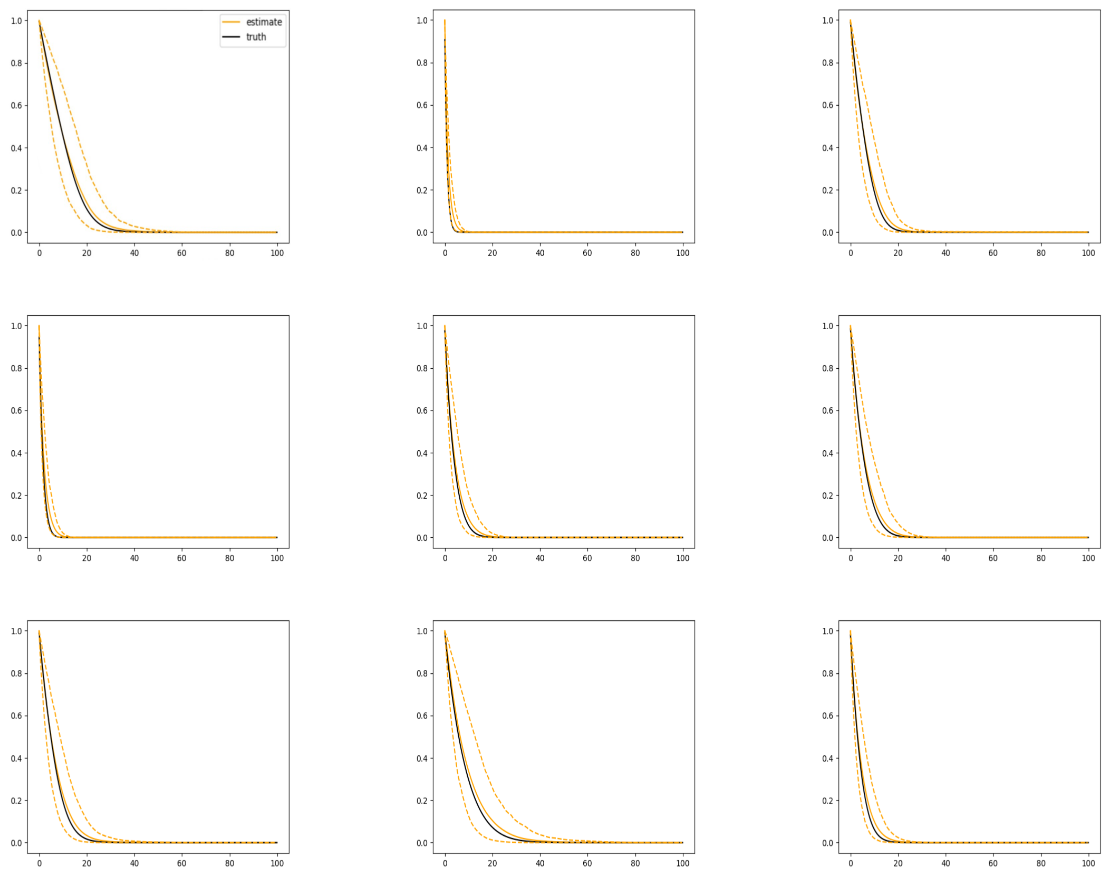


Figure 4.2: Conditional survival curves for 9 different sets of covariates using feed forward neural networks with sample size $n = 2000$.

shows the results from DeepONet with FNN branch net. We see that the estimated curves well overlap with the ground truth (black solid lines). Figure 4.3 shows the results from the Cox-PH model. We only shows the sample average of the Cox-PH model since the variance is small. We can see a large bias from the Cox model. This is expected because the Cox model assumptions are violated in this simulation setup.

Figure 4.4 shows the result with CNN based branch net. We can see that, using CNN structure with the same number of partition (m = 250) gives slightly better survival estimates with smaller variances (narrower confidence bands). Both Figures are based on a sample size of 2000. The results with sample size of 500 are slightly more biased in some cases.

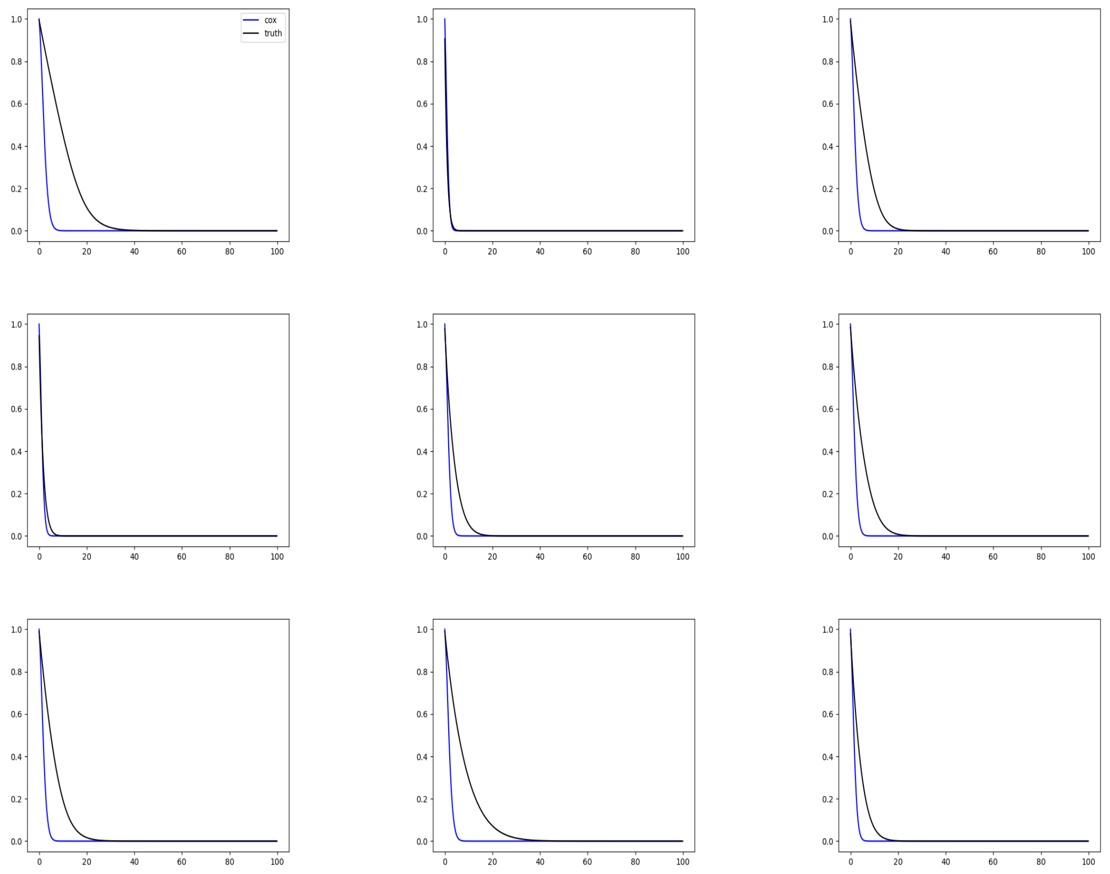


Figure 4.3: Conditional survival curves for 9 different sets of covariates using Cox-PH model with sample size $n = 2000$.

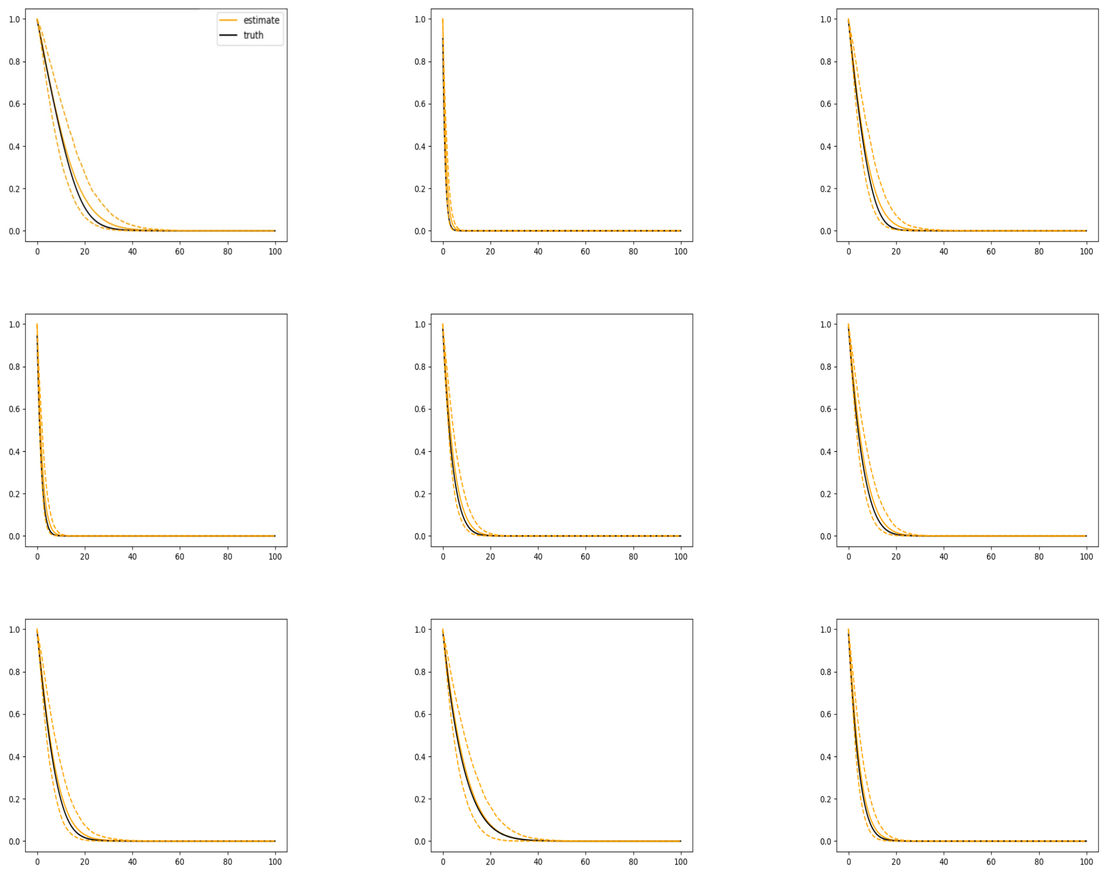


Figure 4.4: Conditional survival curves for 9 different sets of covariates using CNN with sample size $n = 2000$.

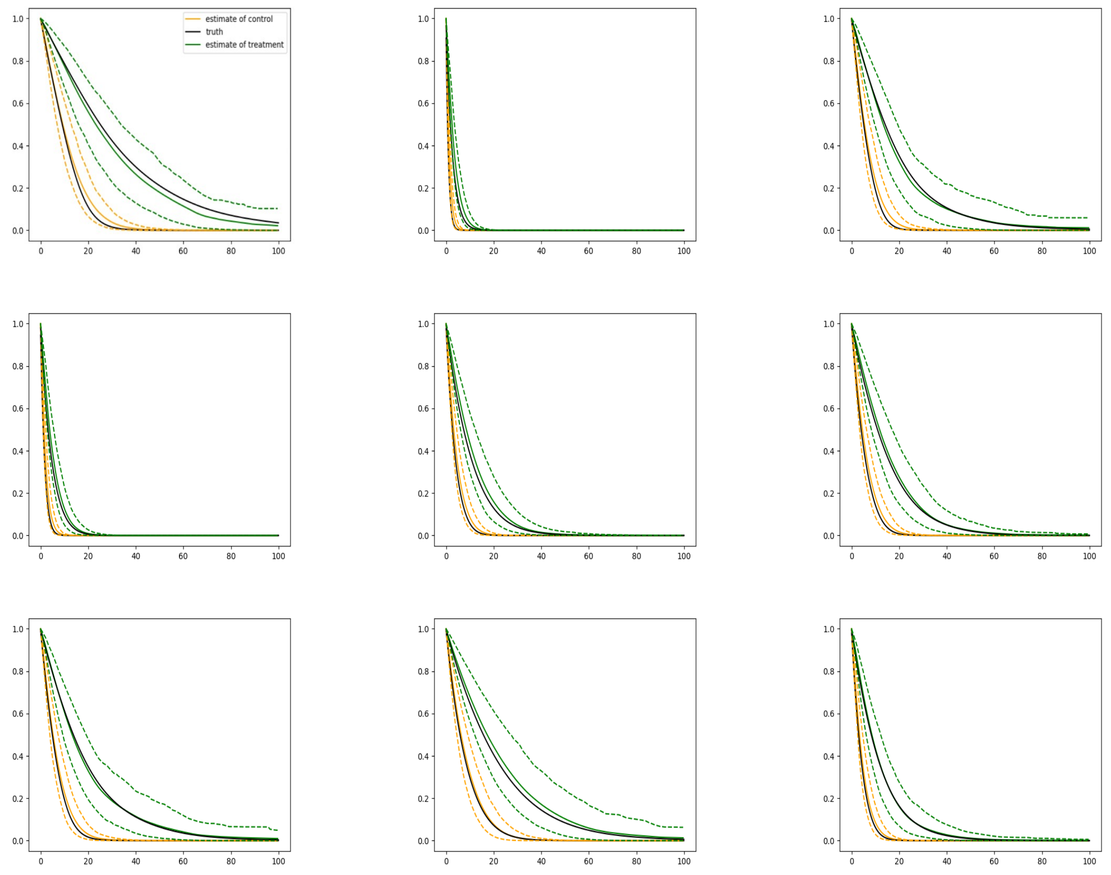


Figure 4.5: Conditional survival curves for with 9 different sets of covariates.

Given a specific set of $x_i(t)$ and w_i, we can estimate the survival curves with $z_i = 0$ and $z_i = 1$ respectively and illustrate the curve difference to mimic the treatment effect for a new individual. This provides a framework of understanding how treatment affects an individual's survival probability adjusted by potential confounders without imposing any model assumption, making the evaluation of personalized treatment possible. Figure 4.5 provides illustrations of such estimates. The upper curves are the ground truth (solid black) and estimates (solid green) for $z_i = 1$, while the lower curves are the ground truth (solid black) and estimates (solid yellow) for $z_i = 0$. We can see the effect variation among different individuals.

4.4 Discussion

It is worth noting that directly using the full likelihood approach enables our method to be the most non-parametric, providing maximum robustness with the least number of assumptions. This makes our approach always valid across all data distributions, ensuring reliability in diverse situations. Other methods may be more efficient, but only when their specific model assumptions hold true.

In the simulations, we numerically evaluate the conditional cumulative hazard function and the conditional survival function at discrete time points. Despite this discrete evaluation, the relationship between survival and hazard functions is treated continuously. This approach may introduce bias because the discrete evaluation may not perfectly align with the continuous model, potentially leading to discrepancies in the estimated functions. To mitigate this bias, it would be prudent to use the exact survival function at discrete time points. This adjustment could improve the accuracy of the estimates by aligning the evaluation more closely with the actual data structure, thereby reducing any potential bias introduced by the continuous approximation.

Additionally, when incorporating covariate history into the neural network, we construct an expanded dataset where each time point for every subject is paired with the entire covariate history. This approach, while comprehensive, significantly increases memory requirements due to the large volume of data involved. Improving the efficiency of data handling and storage is crucial, which is of great interest for future work. It is also of interest to apply our method to real world data with external time-varying covariates.

Chapter 5

Conditional Distribution Estimation Given Functional Covariates Using Deep Operator Neural Networks

In Chapter 3, we show how to use neural networks to estimate the conditional distribution function for a continuous response variable given covariates in an Euclidean space. We know that when the number of the covariates is greater than two, the traditional kernel method becomes difficult to apply. In this chapter, we investigate the estimation of a conditional distribution given functional covariates, which is even more challenging. In this case, the conditional distribution function becomes an operator. We propose to use deep operator network for its estimation.

5.1 Introduction

Functional Data Analysis (FDA, refer to Ramsay & Silverman's work [49] for a comprehensive discussion on FDA methods and see [61] for a recent review) is a branch of statistics that deals with data that are in the form of curves or functions. Different from traditional data, functional data are intrinsically infinite dimensional and generated by smooth underlying processes. In reality, the functional data can only be collected discretely over time or space. It might be tempting to consider functional data as classical multivariate data. Unfortunately, multivariate manipulation does not take into account smoothness or more general structure of the underlying functions so that the curse of dimensionality happens. Numerous practical studies have shown that a direct functional approach gives better results [52]. In this work, we consider a regression setting in which the response is a scalar and at least one of the predictors is a random function. Specifically, let Y be a scalar response variable and $X(t)$, $t \in \mathcal{T}$, be the functional covariate. The conventional FDA focuses on a functional linear model (FLM) [7, 23] that has the following form:

$$E(Y \mid X) = \beta_0 + \int_{\mathcal{T}} X(s)\beta(s)ds.$$

FLM can be fitted by first expanding the covariate and the coefficient function in the same functional basis, i.e., $X(t) = \sum_{k=1}^{\infty} A_k \varphi_k(t)$, $\beta(t) = \sum_{k=1}^{\infty} \beta_k \varphi_k(t)$. Then the model becomes a linear model of the form

$$E(Y \mid X) = \beta_0 + \sum_{k=1}^{\infty} \beta_k A_k,$$

where the infinite sum is replaced by a finite sum that is truncated at the first K terms in implementations.

The conventional FLM is a linear model considering the cumulative linear effect of $X(t)$ on

Y. One direct way to generalize it to nonlinear models is to add a nonlinear link function between the linear predictor and the response [44]. To further remove the model's structural constraints, some nonparametric models have been considered [17].

In recent years, as modern deep learning models have shown competitive performance across various fields, researchers have started to apply them to FDA. Most work uses basis function expansion to convert functional inputs into a finite-dimensional vector before feeding them into a standard feed forward neural network. For example, both [52] and [51] use functional neurons in the first neural network layer. Specifically, they use B-splines to summarize the functional input. [59] also consider using the Fourier basis functions to summarize the functional covariate into a scalar in the first layer of a neural network. The basis coefficients would be updated as the neural network learns. They simply call their neural net model Functional Neural Networks. [65] propose an alternative neural network architecture called AdaFNN that consists of a novel Basis Layer implemented via micro-networks, where the basis functions themselves do not need to be a priori and can be learned from the full data.

On the other hand, convolutional neural networks (CNN) [39] are designed to handle high-dimensional inputs and naturally pool neighboring information. It is shown that, under suitable assumptions, the convergence rate of CNN is independent of the input dimension [37]. CNN-based methods have achieved great success in imaging data analysis and many other fields, revolutionizing the processing and the analysis of complex datasets across various domains. Unlike the methods that integrate basis function expansion, directly using CNNs is more straightforward, allowing for automatic learning of hierarchical representations from raw data. As shown in [59], CNNs achieve performance comparable to that of basis expansion methods on some functional datasets.

In this work, we investigate the potential of applying CNNs to functional data. Instead of mean regression, we aim to estimate the conditional distribution function of a scalar response variable Y given functional covariates X. The idea of using neural networks to estimate conditional distribution functions has been discussed in [29], where covaraites are Euclidean variables. For functional covariates X, we take them together with a grid point as the input of our model, and output the value of the conditional distribution function at that grid point. In order to treat the functional covariates and the grid point differently, we use a neural operator structure called DeepONet [42] to estimate the conditional distribution function of Y given the functional X.

Our method is different from the well-known generative deep learning models. Generative models, such as Generative Adversarial Networks (GANs) [21], Variational Autoencoders (VAEs) [35], and Diffusion Models [27], provide ways to draw new samples from the empirical data distribution, whereas our method directly estimates an arbitrary conditional distribution function. It would be an interesting future work to compare our method with those generative deep learning methods within the context of FDA.

5.2 Methodology

Given a dataset $\{(x_i(\cdot), y_i)\}_{i=1}^n$ of i.i.d. samples of functional covariates $X(\cdot)$ and the associated scalar response variable Y. We assume that $X(t)$ is a stochastic process (or in general a set of stochastic processes and Euclidean variables) defined over a bounded interval $t \in [a, b]$. We then partition $[a, b]$ by grid points $a \leq t_1 < t_2 < \cdots < t_m \leq b$ for numerical considerations and assume $x_i(t)$ is observed on these grid points.

5.2.1 Continuous Response

Since $P(Y \leq y \mid X)$ is non-decreasing and bounded between 0 and 1, in order to estimate the conditional cumulative distribution function (CDF) without any constraint, we use hazard function to construct the likelihood. The hazard function of a continuous Y given X is defined as:

$$\lambda(s \mid X) = \lim_{\Delta s \to 0} \frac{P(s \leq Y < s + \Delta s \mid Y \geq s, X)}{\Delta s}$$

Let $h(s \mid X) = \log \lambda(s \mid X)$. Then h can take any real value without any constraint. The conditional CDF can be expressed in terms of the hazard function as:

$$F(y \mid X) = 1 - \exp\left(-\int_{-\infty}^{y} e^{h(s|X)}\, ds\right). \tag{5.1}$$

Once an estimator of h becomes available, an estimator of the conditional CDF can be obtained from the above equation (5.1) given a set of covariates X.

Clearly $h(s|X)$ is an operator for functional covariates X. Hence we propose to apply the recently developed deep operator neural network approach, the so-called DeepONet method [42], for the estimation of $h(s|X)$. A DeepONet consists of two sub-networks, one for encoding the functional input at the discrete time grid $\{t_1, t_2, ...t_k\}$ (branch net), and another for encoding the locations s for the output functions (trunk net) [42]. The branch and trunk architecture is inspired by the Universal Approximation Theorem for operators [10], which states that two fully connected neural networks with a single hidden layer, combined by a vector dot product of the outputs, are able to approximate any continuous nonlinear operator with arbitrary accuracy. In practice, people usually use multiple hidden layers and different neural network architectures. For trunk net, since we only have one dimensional input, we use simple one layer fully connected neural network. For branch net, we use 1D convolutional layers [36] followed by max pooling layers to summarize the functional covariates. See Figure 5.1 for an illustration. When there exist Euclidean covariates W, we concatenate them with X at each time grid as the branch net input.

Following the notation for operators, we denote $h(s \mid X)$ by $h(X)(s)$ from now on. We use the full likelihood to build the loss function given the dataset $\{(x_i(\cdot), y_i)\}_{i=1}^n$, where $x_i(t)$ is

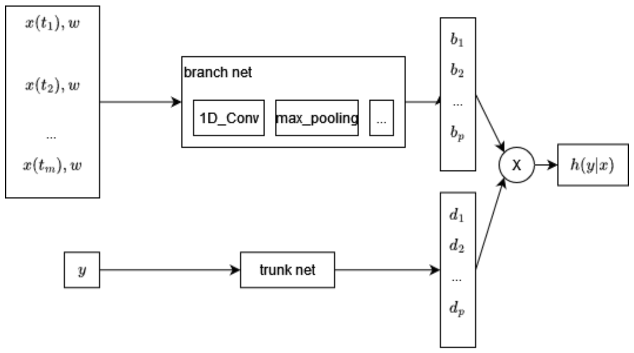


Figure 5.1: DeepONet structure with CNN layers for branch net

observed on the grid points $\{t_1, t_2, ...t_m\}$. Similar to [29], the log likelihood is given by

$$\ell_n = \sum_{i=1}^{n} \left[h(x_i)(y_i) - \int_{-\infty}^{y_i} e^{h(x_i)(s)} ds \right]. \tag{5.2}$$

We evaluate the above integrals using Riemann summation. Specifically, we evaluate $h(x_i)(s)$ on an equal-spaced partition, i.e., $s \in \{s_1, s_2, ..., s_k\}$, where s_1 is the first order statistic of $\{y_i\}_{i=1}^n$, and s_k is the maximum order statistic. Then, we obtain the following discretized loss function:

$$loss(h) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=2}^{k} I(s_{j-1} \le y_i) \left[e^{h(x_i)(s_j)} (s_j - s_{j-1}) - h(x_i)(s_j)\delta_{ij} \right]. \tag{5.3}$$

where $\delta_{ij} = I(s_j \ge y_i)$. Note that the above loss function is obtained by assigning a point mass of $1/n$ at s_1 for the case that Y has unbounded lower support. If the support of Y has a finite lower bound s_0, then the inner summation in (5.3) starts from $j = 1$. Since $F(s_1|x_i) = 1/n$, we have $\int_{-\infty}^{s_1} e^{h(x_i)(s)} ds = -\log(1 - 1/n)$. Once an estimator of h, denoted by $\widehat{h}$, is obtained, the conditional distribution function (5.1) can be estimated by

$$\widehat{F}(y|x) = I(s_1 \le y) \left\{ 1 - (1 - \frac{1}{n}) \exp\left[- \sum_{j=2}^{k} I(s_j \le y) e^{\widehat{h}(x)(s_j)} (s_j - s_{j-1}) \right] \right\}. \tag{5.4}$$

i	evaluation point	covariates
1	s_1	$x_1(t_1), x_1(t_2), \ldots, x_1(t_m)$
1	s_2	$x_1(t_1), x_1(t_2), \ldots, x_1(t_m)$
$\vdots$	$\vdots$	$\vdots$
1	s_m	$x_1(t_1), x_1(t_2), \ldots, x_1(t_m)$
2	s_1	$x_2(t_1), x_2(t_2), \ldots, x_2(t_m)$
2	s_2	$x_2(t_1), x_2(t_2), \ldots, x_2(t_m)$
$\vdots$	$\vdots$	$\vdots$
2	s_m	$x_2(t_1), x_2(t_2), \ldots, x_2(t_m)$
$\vdots$	$\vdots$	$\vdots$

Table 5.1: The data structure for functional data.

Note that in (5.4), $\widehat{F}(y \mid x) = 0$ when $y < s_1$ and $\widehat{F}(y \mid x) = 1/n$ when $y \in [s_1, s2)$.

For better illustration, the data structure is shown in Table 5.1. Each subject is expanded into m rows for input.

5.2.2 Discrete Response

When the response variable Y is discrete, assume $Y \in \{s_1, s_2, \ldots, s_k\}$, the full log likelihood function is

$$\ell_n = \sum_{i=1}^{n} \log f(y_i|x_i),$$

where $f(y|x) = P(Y = y|X = x)$, $y \in \{s_1, s_2, \ldots, s_k\}$. The hazard function of Y given covariates $X = x$ is defined as

$$\lambda(s_j|x) = \frac{f(s_j|x)}{P(T \geq s_j|x)} \in (0, 1].$$

Thus we have

$$f(s_j|x) = \lambda(s_j|x)P(T \geq s_j|x) = \lambda(s_j|x)\prod_{r=1}^{j-1}(1-\lambda(s_r|x)) = \lambda(s_j|x)\prod_{r=1}^{k}(1-\lambda(s_r|x))^{I(s_r<s_j)}.$$

Let $h(x)(s_j) = \log \frac{\lambda(s_j|x)}{1-\lambda(s_j|x)} \in (-\infty, \infty)$, we have

$$\begin{aligned} \ell_n &= \sum_{i=1}^{n}\left\{\log\lambda(y_i|x_i) + \sum_{j=1}^{k} I(s_j < y_i)\log[1-\lambda(s_j|x_i)]\right\} \\ &= \sum_{i=1}^{n}\left\{\log\frac{e^{h(x_i)(y_i)}}{1+e^{h(x_i)(y_i)}} + \sum_{j=1}^{k} I(s_j < y_i)\log\frac{1}{1+e^{h(x_i)(s_j)}}\right\}. \end{aligned}$$

The loss function becomes

$$loss(h) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} I(s_j \leq y_i)\left[\log(1+e^{h(x_i)(s_j)}) - h(x_i)(s_j)\delta_{ij}\right], \tag{5.5}$$

where $\delta_{ij} = I(s_j \geq y_i)$.

Once an estimator of h is obtained, the conditional distribution function can be estimated by

$$\begin{aligned} \hat{F}(s_j|x) &= \sum_{r=1}^{j}\hat{f}(s_r|x) \\ &= \sum_{r=1}^{j}\left\{\hat{\lambda}(s_r|x)\prod_{t=1}^{r-1}[1-\hat{\lambda}(s_t|x)]\right\} \\ &= \sum_{r=1}^{j}\frac{e^{\hat{h}(x)(s_r)}}{\prod_{t=1}^{r}[1+e^{\hat{h}(x)(s_t)}]}. \end{aligned}$$

5.3 Hyperparameters

For the trunk net of DeepONet, we use a simple one layer fully connected neural network. For the branch net, we use CNN to summarize the functional covariates. The CNN structure is composed of 2 blocks of 1D convolutional layers [36] and max pooling layers, followed by 1 fully connected layer.

“Relu” function is used as the activation function between the hidden layers and linear function is used for the final output so that the output value is not constrained. “Adam” is used as the optimizer. In last layer before doing dot product, we use $p = 10$ nodes as suggested in [42]. For simulations in Section 5.4, we choose a commonly used hyperparameter combination. Specifically, we use the following optimal hyperparameters: number of nodes in each dense layer is 128, number of filters in each Conv1D layer is 32, kernel size is 3, pool size is 8 with the same stride, learning rate is 0.0001 and batch size is 1000. We use the same set of hyperparameters for our new method and the conventional mean regression neural networks. This approach allows for a straightforward comparison. For real world data in Section 5.5, we choose the hyperparameters that yield smallest validation loss for both methods. Specifically, for our new method, number of nodes in the last dense layer is 128, number of filters in each Conv1D layer is 64, kernal size is 3, pool size is 2, number of nodes in trunk net dense layer is 64, batch size is 2000, learning rate is 0.0001. For L_2 method, number of filters in each Conv1D layer is 64, kernal size is 3, pool size is 2, number of nodes in each dense layer is 256, batch size is 500, learning rate is 0.001.

The implementation of DeepONet is provided in python package DeepXDE [43]. We modified the code in DeepXDE using its TensorFlow 2.x backend. We trained the neural networks on a system equipped with an AMD Ryzen 9 8945HS processor with Radeon 780M Graphics at 4.00 GHz and 32GB of RAM.

5.4 Simulations

For $t \in \{0, \Delta t, 2\Delta t, ..., 1\}$ with $\Delta t = 0.01$, $i \in \{1, 2, ...n\}$, we generate a functional covariate, $x_i(t)$ from a standard Brownian motion, $\{W_k\}_{k\in[0,1]}$, The increment $W_{k+s} - W_s$ has the $N(0, k)$ distribution. We generate another functional covariate, $z_i(t)$ from a Poisson Process with a rate of 10. We also generate a time independent covariate w_i from a standard normal distribution.

5.4.1 Continuous Response

We consider three simulation setups,

- Setup 1:

$$y_i = 0.01 * \sum_t \{\sin[x_i(t)] + w_i\}[-\log(t)] + \epsilon_i,$$

  and $\epsilon_i \sim N(0, 1)$.

- Setup 2:

$$\log \mu_i = 0.001 * \sum_t \{\sin[x_i(t)] * z_i(t) + z_i(t) + w_i\}[-\log(t)],$$

  and $y_i \sim \text{Exponential}(1/\mu_i)$.

- Setup 3:

$$\log \mu_i = 0.001 * \sum_t \{\sin[x_i(t)] * z_i(t) + z_i(t) + 10 * (w_i + v_i)\}[-\log(t)],$$

  where $v_i \sim N(1 + 0.5w_i, 0.75)$ is an unmeasured confounder. $y_i \sim \text{Exponential}(1/\mu_i)$.

In Setup 1, the model for y_i is relatively straightforward. Here, y_i is generated using a sinusoidal function of $x_i(t)$ with an additive noise term w_i and a time-dependent factor involving the logarithm of t. The error term ϵ_i follows a normal distribution, meaning the errors are uncorrelated with the mean of the response variable y_i. This setup assumes a simple linear relationship with normally distributed errors. In Setup 2, the model is more complex. Instead of modeling y_i directly, it models the logarithm of the expected mean μ_i. The model includes multiplicative and additive interactions with another time-dependent term $z_i(t)$. The final response variable is assumed to follow an Exponential distribution. In Setup 3, we further include an unmeasured confounder v_i into the simulation model.

We independently generate training sets and validation sets. For each simulation run, when the validation loss no longer decreases, we stop training to avoid overfitting. Once the neural net model is trained, we use the fitted model to estimate the conditional CDF curves given newly generated covariates. We repeat the process for $N = 500$ times, then plot the sample average and 90% empirical confidence band of the estimated conditional CDF curves.

The traditional neural network method with the commonly used L_2 loss function gives the conditional mean estimator. The conditional distribution function given a set of covariate values can be estimated by shifting the center of the empirical distribution of training set residuals to the estimated conditional mean. This would yield a valid estimator under the assumption that the errors (outcomes subtract their conditional means) are i.i.d. and uncorrelated with conditional means. On the other hand, in our method, an estimator of the conditional distribution function gives a conditional mean estimator as follows:

$$\int_{-\infty}^{\infty} t d\widehat{F}(t|x) = \sum_{i=1}^{n} t_i \left(\widehat{F}(t_i|x) - \widehat{F}_k(t_{i-1}|x) \right).$$

Thus we can compare our method to the widely imposed mean regression method with the same CNN neural network structure on the estimation of the conditional distribution

	L_2 method	new method
mean squared error for mean estimation	1.0018	1.0325
median squared error for mean estimation	0.4242	0.4418
mean squared error for distribution estimation	0.0016	0.0025
median squared error for distribution estimation	0.0003	0.0004
90% coverage rate	0.843	0.896
95% coverage rate	0.881	0.951

Table 5.2: Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 1)

function as well as the estimation of the conditional mean.

Conditional CDF curves for 9 different sets of functional covariates are shown in Figure 5.2, Figure 5.3 and Figure 5.4. We can see that, in Setup 1, both methods estimate CDF well but the estimated curves by the proposed method shows slight bias in some cases. This is not surprising because L_2 method should have the best performance in this setup. In Setup 2 and Setup 3, L_2 method using the empirical distribution of the residuals can not estimate the distribution curve correctly since there is a mean-variance relationship and the true CDF is not smooth at 0. In Setup 3, the performance of L_2 method is slightly worse because of not accounting for the confounder v_i.

We also evaluate the performance of both methods by averaging the mean and median squared errors for the mean estimation and distribution estimation, respectively, of 200 independently generated test data points over 500 replications. The results are shown in Table 5.2, Table 5.3 and Table 5.4. Coverage rates of 90% and 95% predictive intervals are also presented. For Setup 1, we can see that the new method has larger mean and median squared errors for mean estimation and similar mean and median squared errors for CDF estimation. But for Setup 2 and Setup 3 where L_2 model assumption is violated, the new method has better performance in CDF estimation, which leads to better 90% and 95% coverage rates.

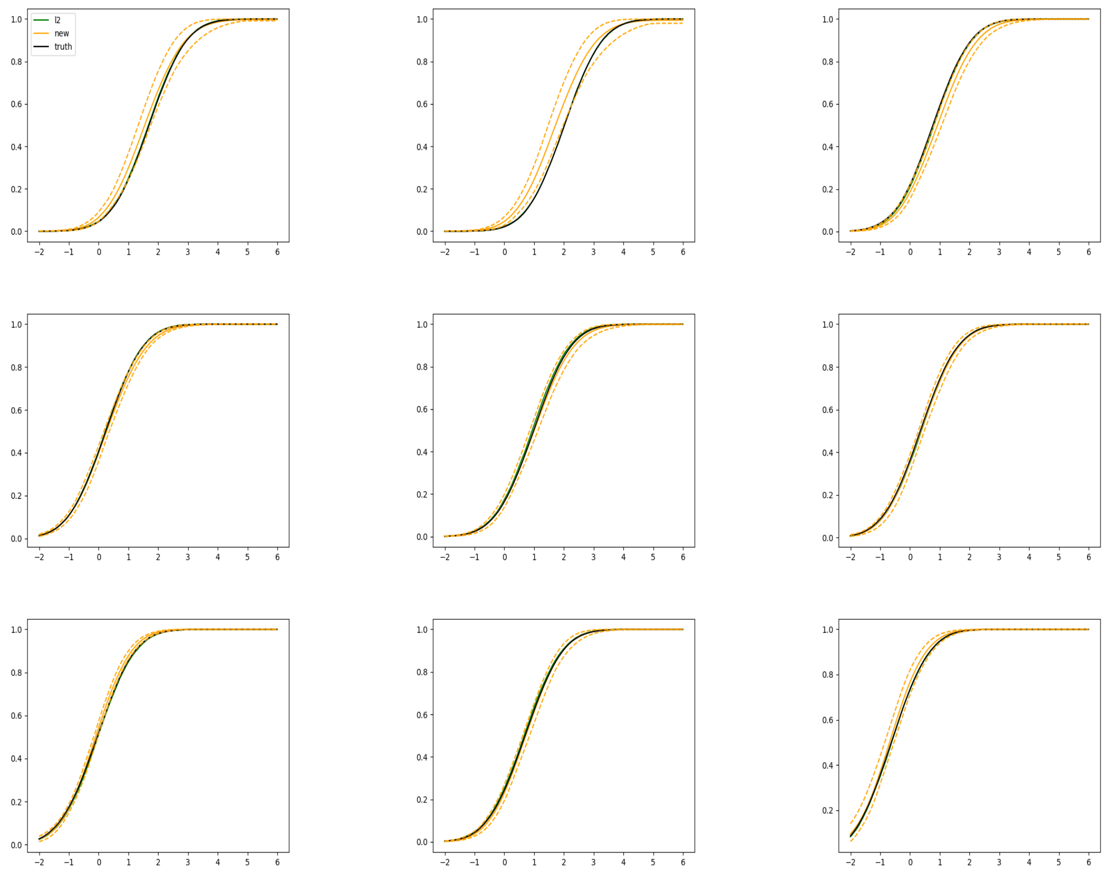


Figure 5.2: Conditional distribution functions for 9 different sets of functional covariates (setup 1).

	L_2 method	new method
mean squared error for mean estimation	0.7679	0.7680
median squared error for mean estimation	0.2203	0.2718
mean squared error for distribution estimation	0.0018	0.0007
median squared error for distribution estimation	0.0009	0.0004
90% coverage rate	0.908	0.893
95% coverage rate	0.961	0.945

Table 5.3: Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 2)

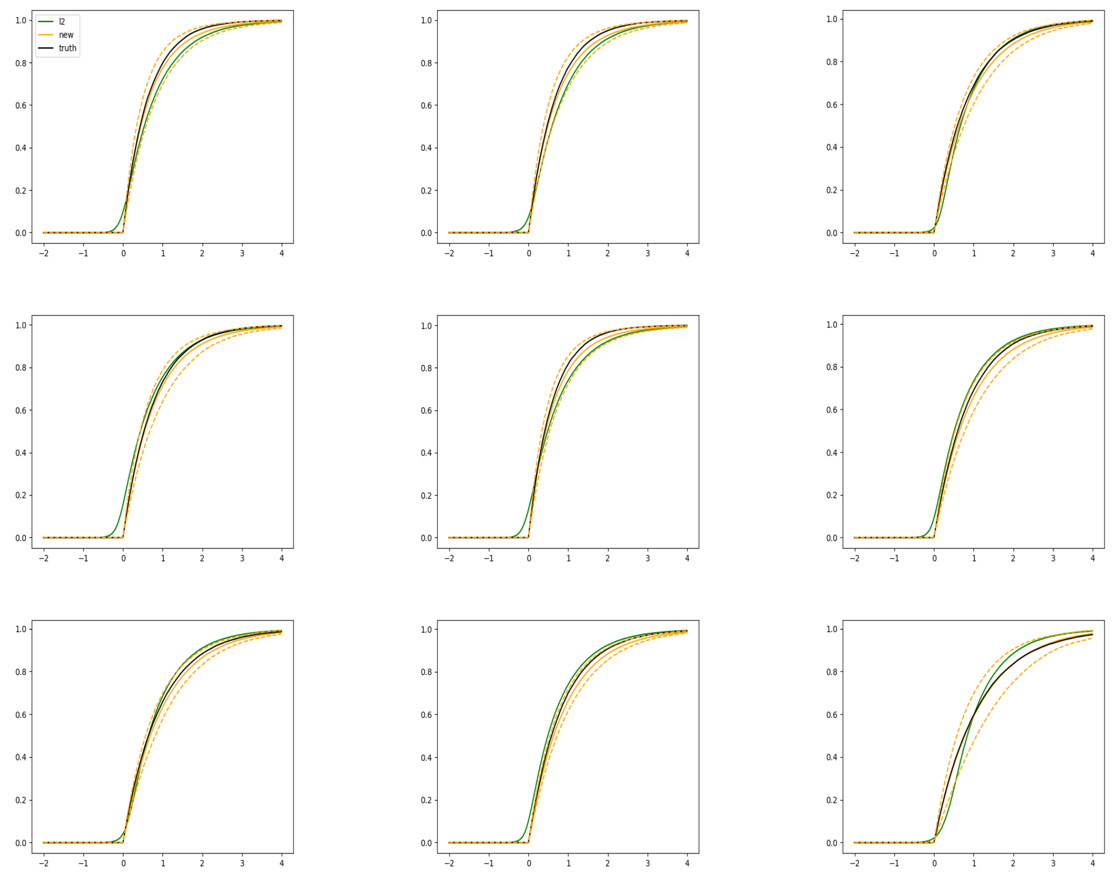


Figure 5.3: Conditional distribution functions for 9 different sets of functional covariates (setup 2).

	L_2 method	new method
mean squared error for mean estimation	0.6369	0.6418
median squared error for mean estimation	0.1763	0.2219
mean squared error for distribution estimation	0.0020	0.0009
median squared error for distribution estimation	0.0009	0.0005
90% coverage rate	0.911	0.896
95% coverage rate	0.960	0.941

Table 5.4: Average mean/median squared errors for mean estimation and distribution estimation, and prediction coverage rates over 500 replications. (Setup 3)

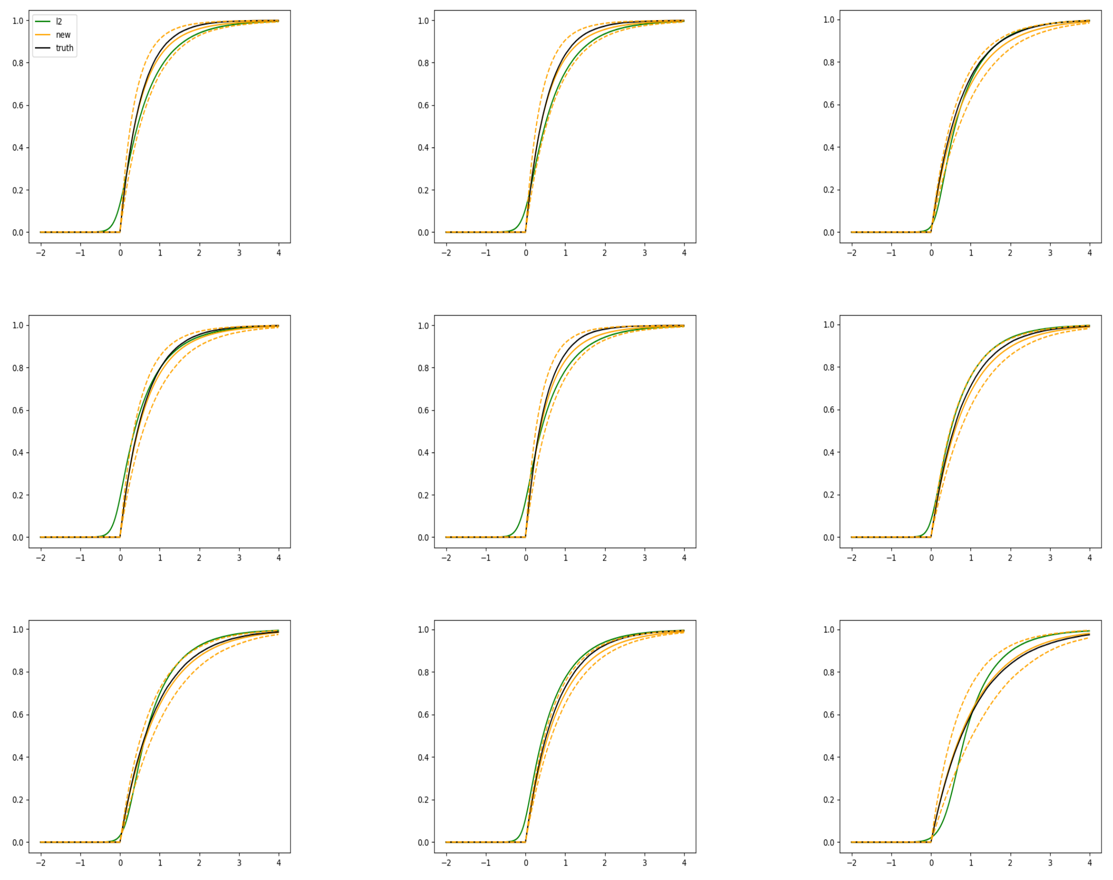


Figure 5.4: Conditional distribution functions for 9 different sets of functional covariates (setup 3).

5.4.2 Discrete Response

For discrete response, we consider the following setup:

$$\mu_i = 0.01 * \sum_t \{\sin[x_i(t)] + w_i\}[-\log(t)],$$

$$y_i \sim \text{Poisson}(\mu_i).$$

In this setup, we are dealing with a discrete response variable y_i that follows a Poisson distribution. The mean parameter μ_i of the Poisson is determined by the time dependent variable $x_i(t)$ and time-independent variable w_i, all modulated by a time-dependent logarithmic factor $-\log(t)$.

Again, we independently generate training sets and validation sets, and train our neural networks with the loss function described in Section 5.2.2. We use the fitted model to estimate the conditional CDF curves given newly generated covariates. We repeat the process for $N = 100$ times, then plot the sample average and 90% confidence band of the estimated conditional CDF curves. The estimated conditional CDF for 9 sets of functional covariates are shown in Figure 5.5. Our method provides a novel way to estimate the stepped CDF of a discrete variable. The estimated step functions (orange) closely align with the truth (black).

5.5 Real Data

In this section, we apply our method on a public real world data set discussed in [59]. The Bike Sharing data [15] contains the hourly and daily count of rental bikes between years 2011 and 2012 in Capital bike share system with the corresponding weather and seasonal information. The goal is to predict the number of daily rentals given hourly temperature (functional, 24 points) in each day. We also include a binary variable representing working

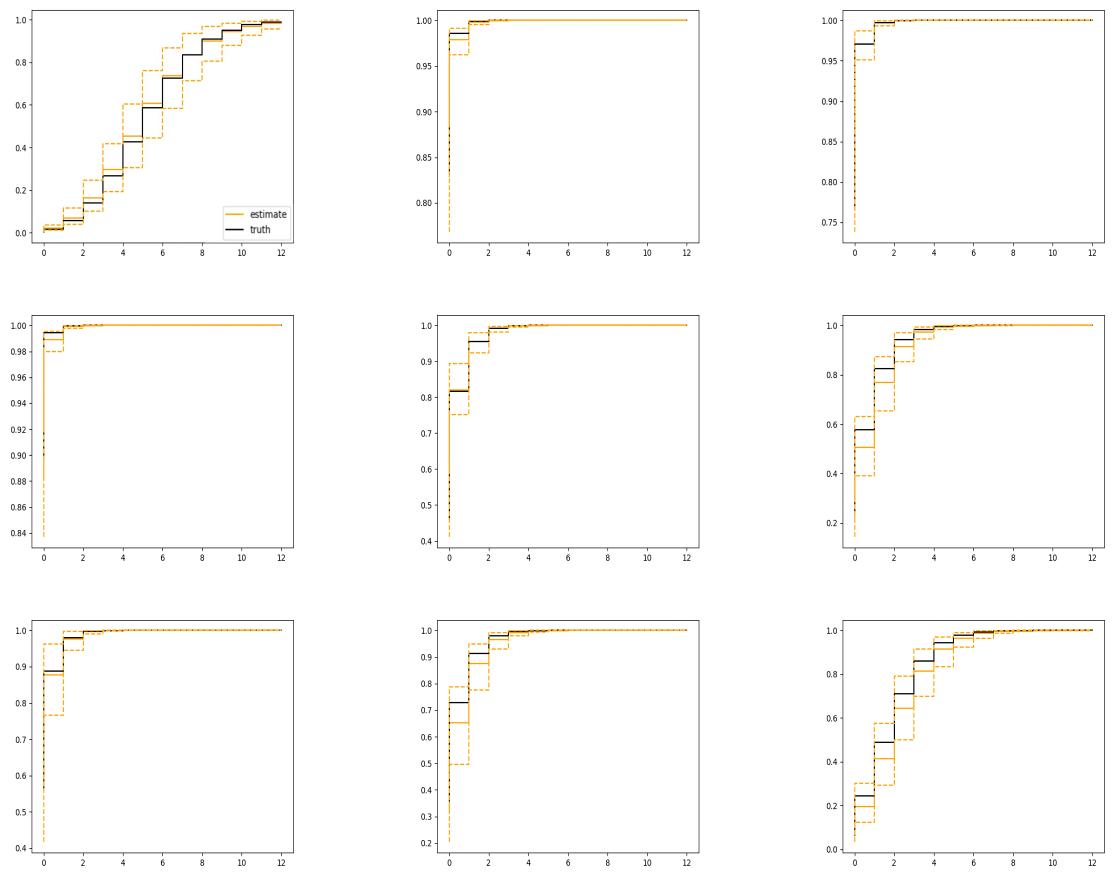


Figure 5.5: Conditional distribution functions for 9 different sets of functional covariates (Discrete Case)

	L_2 method	new method
mean squared error	0.491	0.528
median squared error	0.261	0.323
50% coverage rate	0.447	0.471
90% coverage rate	0.835	0.900
95% coverage rate	0.892	0.956

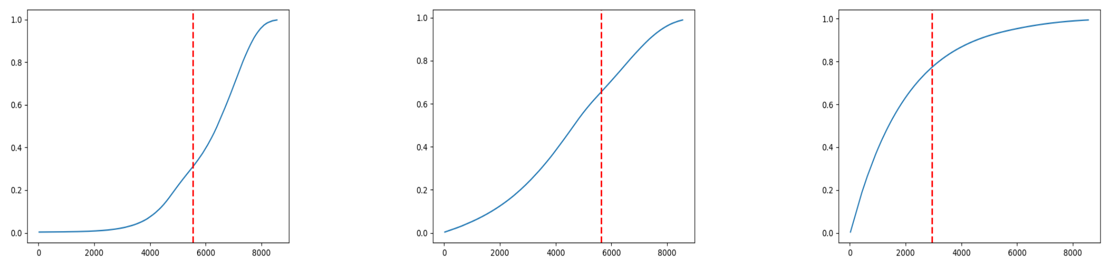
Table 5.5: The prediction coverage rate and mean/median squared errors for mean estimation.

day or not in the prediction model because this could be a crucial factor to the bike rental. The sample size is 730. Although the number of daily rentals is discrete, it ranges from a few hundreds to more than 8000. So, it would be reasonable to treat the response as continuous.

Although the Bike dataset has been previously analyzed using other FDA methods, such as in [59], the data usage differs, and the preprocessing details were not clearly presented in earlier works. For instance, [59] only uses Saturday records, to eliminate the effect of different week days, resulting in a very small sample size. These factors make it challenging to directly compare different methods with existing results.

In [59], it was demonstrated that conventional mean regression neural networks with a CNN structure perform well in predicting mean outcomes compared to kernel methods. Therefore, we compare our method, which predicts the conditional CDF, with the mean regression neural networks (L_2 method). Specifically, we employ the same CNN structure in the branch net of our method as used in the conventional mean regression neural networks.

We use 5-fold cross-validation to train the models and make predictions. In real-world datasets, where the underlying conditional CDF is unknown, we assess the quality of the CDF estimate by examining the coverage rate. Additionally, as described in Section 5.4.1, we calculate the conditional mean from the estimated conditional CDF. The performance of the methods is then evaluated using 5-fold cross-validated mean squared error (MSE) and median squared error for mean estimation. The results are summarized in Table 5.5.



(a) (b) (c)

Figure 5.6: Estimated conditional distribution functions for 3 individuals. The vertical lines illustrate locations of the observed values.

Since the L_2 method directly minimizes the mean squared error, it is expected that the MSE of the predicted mean would be smaller. Our new method provides reasonable estimates of the mean. Regarding coverage rates, our CDF method provides better results. Our coverage rates are very close to the expected values, while L_2 method has lower coverage rates than expected, indicating that the prediction intervals produced by L_2 model are narrower than the truth.

Figure 5.6 illustrates how our method estimates the conditional CDF (represented by the blue curve) based on the functional and scalar covariates. The observed values of the response are indicated by the vertical red lines.

Bibliography

[1] L. Antolini, P. Boracchi, and E. M. Biganzoli. A time-dependent discrimination index for survival data. *Statistics in medicine*, 24 24:3927–44, 2005.

[2] A. Avati, T. Duan, S. Zhou, K. Jung, N. H. Shah, and A. Y. Ng. Countdown regression: Sharp and calibrated survival predictions. In R. P. Adams and V. Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 145–155. PMLR, 22–25 Jul 2020.

[3] A. Bennis, S. Mouysset, and M. Serrurier. Dpwte: A deep learning approach to survival analysis using a parsimonious mixture of weibull distributions. In I. Farkaš, P. Masulli, S. Otte, and S. Wermter, editors, *Artificial Neural Networks and Machine Learning – ICANN 2021*, pages 185–196, Cham, 2021. Springer International Publishing.

[4] E. Biganzoli, P. Boracchi, L. Mariani, and E. Marubini. Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach. *Statistics in Medicine*, 17(10):1169–1186, 1998.

[5] T. F. Brooks, D. S. Pope, and M. A. Marcolini. Airfoil self-noise and prediction. *NASA reference publication 1218*, 1989.

[6] S. F. Brown, A. J. Branford, and W. Moran. On the use of artificial neural networks for the analysis of survival data. *IEEE Transactions on Neural Networks*, 8(5):1071–1077, 1997.

[7] H. Cardot, F. Ferraty, and P. Sarda. Functional linear model. *Statistics Probability Letters*, 45(1):11–22, 1999.

[8] M. Cassotti, D. Ballabio, R. Todeschini, and V. Consonni. A similarity-based qsar model for predicting acute toxicity towards the fathead minnow (pimephales promelas). *SAR and QSAR in Environmental Research*, 26(3):217–243, 2015.

[9] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint*, 2018.

[10] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its applications to dynamic systems. *Neural Networks, IEEE Transactions on*, pages 911 – 917, 08 1995.

[11] T. Ching, X. Zhu, and L. X. Garmire. Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data. *PLOS Computational Biology*, 14(4):1–18, 04 2018.

[12] F. Chollet et al. Keras, 2015.

[13] D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

[14] S. Cygu, H. Seow, J. Dushoff, and B. M. Bolker. Comparing machine learning approaches to incorporate time-varying covariates in predicting cancer survival time. *Scientific Reports*, 13(1370), 2023.

[15] H. Fanaee-T. Bike Sharing. UCI Machine Learning Repository, 2013. DOI: https://doi.org/10.24432/C5W894.

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

[17] F. Ferraty and P. Vieu. Nonparametric functional data analysis: Theory and practice. 51, 01 2006.

[18] M. F. Gensheimer and B. Narasimhan. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257–e6257, 01 2019.

[19] E. Giunchiglia, A. Nemchenko, and M. van der Schaar. Rnn-surv: A deep recurrent model for survival analysis. pages 23–32, 2018.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[22] E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in medicine*, 18 17-18:2529–45, 1999.

[23] P. Hall and J. L. Horowitz. Methodology and convergence rates for functional linear regression. *The Annals of Statistics*, 35(1):70 – 91, 2007.

[24] P. Hall, R. C. L. Wolff, and Q. Yao. Methods for estimating a conditional distribution function. *Journal of the American Statistical Association*, 94(445):154–163, 1999.

[25] P. Hall and Q. Yao. Approximating conditional distribution functions using dimension reduction. *The Annals of Statistics*, 33(3):1404–1421, 2005.

[26] F. E. Harrell, K. L. Lee, R. M. Califf, D. B. Pryor, and R. A. Rosati. Regression modelling strategies for improved prognostic prediction. *Statistics in medicine*, 3(2):143—152, 1984.

[27] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020.

[28] T. Hothorn, P. Bühlmann, S. Dudoit, A. Molinaro, and M. J. Van Der Laan. Survival ensembles. *Biostatistics*, 7(3):355–373, 12 2005.

[29] B. Hu and B. Nan. Conditional distribution function estimation using neural networks for censored and uncensored data. *Journal of Machine Learning Research*, 24(223):1–26, 2023.

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

[31] J. Islam, M. Turgeon, R. Sladek, and S. Bhatnagar. Case-base neural network: Survival analysis with time-varying, higher-order interactions. *Machine Learning with Applications*, 16:100535, 2024.

[32] J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data (2nd ed.)*. Hoboken, NJ:Wiley, 2002.

[33] J. L. Katzman, U. Shaham, A. Cloninger, J. Bates, T. Jiang, and Y. Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1):24, 2018.

[34] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014.

[35] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[36] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021.

[37] M. Kohler, A. Krzyżak, and B. Walter. On the rate of convergence of image classifiers based on convolutional neural networks. *Annals of the Institute of Statistical Mathematics*, 74(6):1085–1108, December 2022.

[38] N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

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

[41] M. Leblanc and J. Crowley. Survival trees by goodness of split. *Journal of the American Statistical Association*, 88(422):457–467, 1993.

[42] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, 03 2021.

[43] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[44] H.-G. Müller and U. Stadtmüller. Generalized functional linear models. *The Annals of Statistics*, 33(2):774 – 805, 2005.

[45] C. Nagpal, V. Jeanselme, and A. Dubrawski. Deep parametric time-to-event regression with time-varying covariates. In R. Greiner, N. Kumar, T. A. Gerds, and M. van der Schaar, editors, *Proceedings of AAAI Spring Symposium on Survival Prediction - Algorithms, Challenges, and Applications 2021*, volume 146 of *Proceedings of Machine Learning Research*, pages 184–193. PMLR, 22–24 Mar 2021.

[46] C. Nagpal, X. R. Li, and A. Dubrawski. Deep survival machines: Fully parametric survival regression and representation learning for censored data with competing risks, 2021.

[47] C. Nagpal, S. Yadlowsky, N. Rostamzadeh, and K. Heller. Deep cox mixtures for survival regression, 2022.

[48] L. Pontrjagin, V. Boltyanskii, R. Gamkrelidze, E. Mishchenko, and D. Brown. *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics. Wiley, 1962.

[49] J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer, 2005.

[50] G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 1999.

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

[52] F. Rossi, B. Conan-Guez, and F. Fleuret. 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 vol.3, 2002.

[53] I. Selingerova, S. Katina, and I. Horova. Comparison of parametric and semiparametric survival regression models with kernel estimation. *Journal of Statistical Computation and Simulation*, 91(13):2717–2739, 2021.

[54] S. A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Syst.*, 2, 1988.

[55] L. Spierdijk. Nonparametric conditional hazard rate estimation: A local linear approach. *Computational Statistics & Data Analysis*, 52(5):2419–2434, 2008.

[56] N. W. Street. A neural network model for prognostic prediction. In *ICML*, pages 540–546. Citeseer, 1998.

[57] W. Tang, J. Ma, Q. Mei, and J. Zhu. Soden: A scalable continuous-time survival model through ordinary differential equation networks. *Journal of Machine Learning Research*, 23(34):1–29, 2022.

[58] T. M. Therneau. *A Package for Survival Analysis in R*, 2021. R package version 3.2-13.

[59] B. Thind, K. Multani, and J. Cao. Deep learning with functional inputs. *Journal of Computational and Graphical Statistics*, 0(0):1–10, 2022.

[60] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[61] J.-L. Wang, J.-M. Chiou, and H.-G. Müller. Functional data analysis. *Annual Review of Statistics and its application*, 3:257–295, 2016.

[62] L. J. Wei. The accelerated failure time model: A useful alternative to the cox regression model in survival analysis. *Statistics in Medicine*, 11(14-15):1871–1879, 1992.

[63] S. Wiegrebe, P. Kopper, R. Sonabend, B. Bischl, and A. Bender. Deep learning for survival analysis: a review. *Artificial Intelligence Review*, 57(3):65, 2024.

[64] A. Xiang, P. Lapuerta, A. Ryutov, J. Buckley, and S. Azen. Comparison of the performance of neural network methods and cox regression for censored survival data. *Computational Statistics & Data Analysis*, 34(2):243 – 257, 2000.

[65] J. Yao, J. Mueller, and J.-L. Wang. Deep learning for functional data analysis with adaptive basis layers. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11898–11908. PMLR, 18–24 Jul 2021.