JOURNAL OF COMPUTATIONAL AND GRAPHICAL STATISTICS
2024, VOL. 33, NO. 1, 181–191
https://doi.org/10.1080/10618600.2023.2233581

Taylor & Francis
Taylor & Francis Group

Check for updates

Functional Nonlinear Learning

Haixu Wang[a] and Jiguo Cao[b]

[a]Department of Mathematics and Statistics, University of Calgary, Calgary, Canada; [b]Department of Statistics & Actuarial Science, Simon Fraser University, Burnaby, Canada

ABSTRACT

Using representations of functional data can be more convenient and beneficial in subsequent statistical models than direct observations. These representations, in a lower-dimensional space, extract and compress information from individual curves. The existing representation learning approaches in functional data analysis usually use linear mapping in parallel to those from multivariate analysis, for example, functional principal component analysis (FPCA). However, functions, as infinite-dimensional objects, sometimes have nonlinear structures that cannot be uncovered by linear mapping. Linear methods will be more overwhelmed by multivariate functional data. For that matter, this article proposes a functional nonlinear learning (FunNoL) method to sufficiently represent multivariate functional data in a lower-dimensional feature space. Furthermore, we merge a classification model for enriching the ability of representations in predicting curve labels. Hence, representations from FunNoL can be used for both curve reconstruction and classification. Additionally, we have endowed the proposed model with the ability to address the missing observation problem as well as to further denoise observations. The resulting representations are robust to observations that are locally disturbed by uncontrollable random noises. We apply the proposed FunNoL method to several real datasets and show that FunNoL can achieve better classifications than FPCA, especially in the multivariate functional data setting. Simulation studies have shown that FunNoL provides satisfactory curve classification and reconstruction regardless of data sparsity. Supplementary materials for this article are available online.

ARTICLE HISTORY
Received June 2022
Accepted June 2023

KEYWORDS
Curve classification; Feature mapping; Functional data analysis; Neural networks

1. Introduction

Representation learning can be a powerful tool to extract features for functional data. Functional data are curves, images, or any objects defined over time, location, and other continuums (Ramsay and Silverman 2005; Vieu and Ferraty 2006; Kokoszak and Reimherr 2017). Functional data are more prone to having nonlinear structures which would require substantial effort to process for further utilization. Directly using observed functional data for supervised models can be inefficient and even more so when observations are contaminated with uncontrollable random errors. Hence, we often use a low-dimensional latent feature space to represent the original functional data. The representations can be used for denoising functional data for recovering individual functions. In addition, the representations can convey more information than interpolating the observed data. For example, the feature space may contain grouping or label information of observed trajectories. Building a classification model is thus more efficient by using such representations than original observations. The aforementioned matters motivate us to construct a representation learning model for functional data.

Many representation learning methods have been developed for scalar variables, but there are only a few methods dedicated to functional data. The most used approach is the functional principal component analysis (FPCA), which aims to find major sources of variation in functional data. The major sources of variation are represented by the functional principal components (FPCs). Individual functional data can be expressed by a linear combination of FPCs. The coefficients to the FPCs are called FPC scores. The scores are the representations produced by FPCA, and they are linear projections in the low dimensional space defined by selected FPCs. We can then establish functional regression models based on FPC scores so that the model estimation is less burdened by the dimensionality of functional data. FPCA also provides an inherent way to reconstruct underlying functional data observed with random errors. Ramsay and Silverman (2005) provided a comprehensive introduction to FPCA. The theories of FPCA were well established by Dauxois et al. (1982), Hall and Hosseini-Nasab (2005), and Hall et al. (2006). Several extensions to standard FPCA have been proposed to adjust the representations for different scenarios. For instance, FPCs can be more interpretable when using a sparsity assumption as illustrated in Chen and Lei (2015), Lin et al. (2016), and Nie and Cao (2020). Furthermore, Dai and Müller (2018) considered the case where functional data are defined on some Riemannian manifolds than the usual Euclidean space. As the representations of functional data, the FPC scores are used for subsequent statistical models. FPC scores can be used for establishing functional linear models (Yao et al. 2005b) or generalized functional linear models (Müller and Stadtmüller 2005). Moreover, the scores can be used for either clustering (Peng and Müller 2008) or classification (Müller 2005) of individual curves.

CONTACT Jiguo Cao *jiguo_cao@sfu.ca* Department of Statistics & Actuarial Science, Simon Fraser University, Burnaby, Canada.
Supplementary materials for this article are available online. Please go to *www.tandfonline.com/r/JCGS*.
© 2023 American Statistical Association and Institute of Mathematical Statistics

Curve classifications by FPC scores were further investigated in Delaigle and Hall (2012) and Zhou and Sang (2021). In real applications, functional data are not always ideal. James et al. (2000), Yao et al. (2005a), and Nie et al. (2022) addressed a common issue where functional data are not fully observed. In some cases, functional data contain excessive noises that would undermine the usual estimation process. Kong et al. (2018), James et al. (2000), and Shi and Cao (2022) demonstrated how to introduce robustness into the estimation for establishing a functional regression model or applying FPCA. Shi et al. (2021) proposed an FPCA method for longitudinal data with informative dropout. Nie et al. (2018), Jiang et al. (2023a), and Jiang et al. (2023b) developed supervised FPCA methods by considering the correlation of functional predictors with responses. Shi et al. (2022) proposed an FPCA method for image data in a new regression framework.

FPCA has been well developed for univariate functional data but cannot directly address multivariate data. Explaining the variation of multivariate trajectories becomes more difficult. Representing multivariate functional data by vectors are more beneficial for subsequent models in comparison to univariate case. Ramsay and Silverman (2005) suggests one approach for dealing with multivariate functional data. That is, we can join multivariate trajectories in a univariate one, that is, connecting functions by endpoints from all dimensions. The resulting functional data becomes univariate, and standard FPCA can then be applied to the obtained univariate functions. Different functional variables have different measurement units and magnitudes, hence, Chiou et al. (2014) employs normalization to balance the degrees of variations across dimensions. In the meantime, Happ and Greven (2018) investigated the case that trajectories from different dimensions are observed over nonuniform domains. This approach circumvents the step to concatenate multivariate functions into univariate ones. It performs FPCA on each dimension in the multivariate functional data and then applies principal component analysis on the FPC scores from each dimension.

Linear projections have at least the following limitations for representing functional data. The foremost limitation is that linear projection cannot recognize the nonlinear structures in functional data. Projections in the linear feature space may not offer more effective information compared with original observations. Second, the conventional FPCA approach is to concatenate the multivariate variable into a univariate one and apply FPCA to the concatenation of functional data. As a result, the linear components have to consider the nonlinear structures compounded by all dimensions. Thus, we need to pursue a mapping that addresses the nonlinearity in functional data and is more flexible in dealing with multivariate data. The third limitation of FPCA is that it relies on the assumption of a common covariance function of all observed data. The associated functional principal space is a direct consequence of the assumption. However, if functional data contain labels for individual trajectories, then the assumption of a common eigenspace (Dai et al. 2017) may be violated. The violation undermines the validity of the functional principal component space which does not convey label information. Alternatively, we can separately apply FPCA to observations with different labels. But the projections in a different functional principal component space are not comparable and cannot be used for developing a classification model. Therefore, we need to design a mapping that does not depend on the estimation of covariance functions. This mapping can be applied to observed data regardless of labels. The corresponding representations convey necessary label information and are comparable in a shared feature space. As a result, the representations can be passed onto the subsequent classification model. Last but not least, after obtaining the functional principal components, we usually keep the leading few FPCs and discard the rest because the leading FPCs contain the most information of functional data with respect to variations. Therefore, the final FPC space is a truncated space and may not be optimal for subsequent models. The low-rank FPCs may also be essential in the classification of functional data and are sometimes more significant than the leading FPCs. This limitation motivates us to establish a mapping that does not discard any information in doing dimension reduction for functional data.

In this work, we propose a functional nonlinear learning (FunNoL) method for functional data to overcome the aforementioned limitations of linear projections. The FunNoL method employs a nonlinear mapping $\phi : \mathcal{X} \to \mathcal{F}$ for functional data $\boldsymbol{X}(t)$ in the functional space $\mathcal{X}$ and establishes a feature space $\mathcal{F}$. Assuming that the dimension of the representation $\boldsymbol{z}$ in $\mathcal{F}$ is a much lower dimensional object than the original functional data, we can use ϕ to compress information of any given individual curve. The nonlinear mapping ϕ is not subject to any specific form. It can attend to nonlinear structures of functional data that would be overlooked by linear methods. We use the most popular tool, neural networks, to implement and estimate nonlinear mappings. In real applications, functional data are always subject to excessive noises or missing observations. These can be thought of as examples of the data corruption problem. For that matter, we introduce robustness to the representation model for dealing with missing observations and excessive local disturbances of observed functional data. The obtained representations can focus on information from the underlying function.

Our proposed FunNoL method has at least the following advantages in comparison with FPCA. First, FunNoL addresses the nonlinear structures in functional data with a flexible nonlinear mapping estimated by neural networks. It alleviates the requirement of specifying any forms for the mapping functions. Second, multivariate functional data can be input into the networks without being changed to the univariate one. Third, FunNoL does not need to calculate covariance functions which require substantial computational efforts when the sample size or the number of observation points is large. Fourth, our proposed FunNoL method is a one-step model by which feature extraction and curve classification are done at the same time. The nonlinear mapping ϕ and its associated feature space are obtained under the criteria containing two objective functions for curve reconstruction and classification. Hence, the resulting feature space is optimal for both tasks. In comparison to the proposed FunNoL method, the conventional FPCA-based approaches are two-step methods, that is, they extract features (linear projections) in the first step and use them for the classification/clustering in the second step. Last but not least, we do not need to truncate the feature space which is an essential

step in FPCA, hence, the representations can preserve as much information as possible. In conclusion, FunNoL can provide reliable and efficient low-dimensional representations of functional data.

We consider two modeling tasks, classification and reconstruction of functional data, upon obtaining the representations in real data applications. Based on the representations, we can consider a classifier f, a linear or nonlinear function, to produce labels for observed curves. Establishing a classification model on vector-valued representations is much easier than that on functions. It is also straightforward to extend the classification model for multivariate response variables. The secondary task is to recover underlying individual curves given noisy and corrupted observations. FPCA offers an easy solution to this task given that individual curves are defined as a linear combination of FPCs. On the other hand, representations obtained through nonlinear mappings require an inverse mapping ψ to reconstruct the input curves. Acquiring ψ is hence included in the estimation process. In the later section, we will illustrate how to use neural networks to simultaneously estimate ϕ, ψ, and f. Obtaining ϕ under two objectives allows the feature space to contain and exchange information between two tasks. This can also help to solve the over-fitting problem where each modeling task acts as a regularizer on the other. As a result, the features have better generalization ability and are more task-oriented than using a single modeling task.

Real applications are carried out with a focus on the aforementioned two tasks. We will apply the FunNoL method to several datasets to fairly examine its performance. The datasets are retrieved from Dau et al. (2019). The primary assessment for the model is based on the label prediction of test data. One evaluation of representations is expressiveness indicating whether the following model can easily use the information in the feature space. For that matter, we can consider using a simple logistic regression to obtain predicted labels. Based on the comparison of prediction performance, the proposed FunNoL method establishes a feature space that is more effective than the FPC space with the same number of dimensions. We have also evaluated the performance of the FunNoL method under various sparsity settings of training and test data. The FunNoL method provides accurate classification and reconstruction results that do not suffer from missing observations.

In this article, we demonstrate the flexibility and usefulness of the FunNoL method for functional data. It addresses the nonlinear structures of observed curves and handles multivariate variables more naturally in comparison with FPCA. The introduced robustness of representations can overcome various challenges caused by data corruption. It is evident that the FunNoL method can be involved in more applications from FDA. The remainder of this article is organized as follows. Section 2 outlines the representation learning model. A bound of generalization errors is provided, and we will demonstrate that the smoothness of mapping functions is enforced with manually corrupted observations. Section 3 illustrates the estimation process of nonlinear mappings. Applications on real datasets are examined in Section 4. Section 5 contains simulation studies to examine the ability of FunNoL for dealing with data corruption problems. Section 6 concludes the article. The computing codes for some application studies are available at *https://github.com/caojiguo/FunNoL*.

2. Methodology

Representation learning of functional data entails the construction of a mapping ϕ, from the data space $\mathcal{X}$ to the associated feature space $\mathcal{F}$. In this work, we propose to establish a nonlinear mapping, more general and flexible, for producing representations of functional data. The proposed functional nonlinear learning (FunNoL) method includes three components: ϕ for learning features, ψ for reconstructing functional data, and f for classifying functional data. We will approximate these nonlinear functions with the following neural network implementation:

$$\begin{aligned} &\mathbf{h}_j = g_h(\mathbf{W}\mathbf{x}_j + \mathbf{U}\mathbf{h}_{j-1}) \quad \text{and} \quad \mathbf{z} = g_z(\mathbf{V}\mathbf{h}_J), \\ &\tilde{\mathbf{h}}_j = g_{\tilde{h}}(\tilde{\mathbf{W}}\mathbf{z} + \tilde{\mathbf{U}}\tilde{\mathbf{h}}_{j-1}) \quad \text{and} \quad \tilde{\mathbf{x}}_j = g_x(\mathbf{G}\tilde{\mathbf{h}}_j), \\ &\mathbf{y} = g_y(\mathbf{M}\mathbf{z}). \end{aligned} \tag{1}$$

All g's are activation functions to be chosen in applications. $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\tilde{\boldsymbol{W}}$, $\tilde{\boldsymbol{U}}$, $\boldsymbol{M}$, and $\boldsymbol{G}$ are parameter matrices of the recurrent neural networks. Let $\mathbf{x}_j$ be a D-dimensional vector corresponding to the evaluation of $\mathbf{X}(t_j) = (X_1(t_j), \ldots, X_D(t_j)$ at t_j where t_j is the jth discrete observation of any individual trajectory, where $j = 1, \ldots, J$, and J is the total number of observation points. Let the hidden state $\mathbf{h}_j$ and the feature space be L-dimensional, and the curve label $\mathbf{y}$ is a Q-dimensional vector of binary values, such that the element in $\mathbf{y}$ is 1 for the observed label and zero otherwise. We consider a multi-class classification problem where Q corresponds to the number of classes. Then, $\mathbf{W}$ is an L by D matrix, and $\mathbf{U}$ is an L by L matrix. All of $\mathbf{V}$, $\tilde{\mathbf{W}}$, and $\tilde{\mathbf{U}}$ are L by L matrices, whereas $\mathbf{G}$ and $\mathbf{M}$ are D by L and Q by L matrices, respectively. We choose the number of representations, L, by minimizing the K-fold cross-validation error in doing classification.

Figure 1 depicts the neural networks which estimate the representation model including the aforementioned three components.

Explicitly defining a nonlinear mapping can be aimless in searching for the most optimal one. For that matter, we can use neural networks to approximate nonlinear functions. This allows us to obtain mappings for functional data without specifying any particular form for them. Another advantage of neural networks is that we do not need to process data before doing the estimation. Multivariate functional data can be directly fed into the network without being flattened into univariate trajectories. In the latter part of the section, we can also demonstrate that neural networks can handle missing observation, or data corruption, in functional data.

First, we will focus on the estimation of ϕ which takes the input of trajectories and produces a representation vector. Functional data is distinct from multivariate data where the former is defined over a continuum. The dependence between values at different observation points prohibits us to use standard feed-forward neural networks (FNN). If we input functional data into FNN, then the dependency in trajectories gets lost after the first layer where we take the inner product between them and coefficients. To be more adequate for functional data, we can

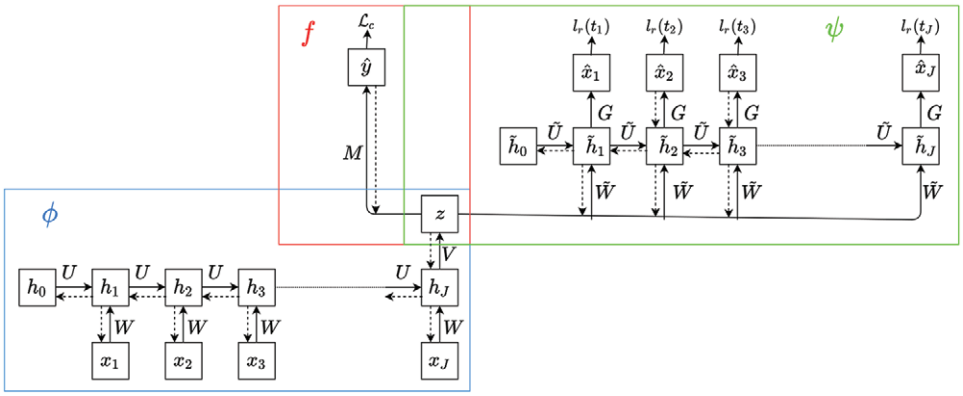

Figure 1. The network design of the proposed functional nonlinear learning (FunNoL) model (1). It includes three components ϕ, ψ, and f for learning features, reconstructing functional data, and classifying functional data, respectively. The bold arrow lines correspond to the direction of propagation, whereas the dashed arrow lines indicate the flow of gradients. Boxes enclose the networks estimating each component in the model.

use recurrent neural networks (RNN) which can maintain the continuity in observed trajectories.

Let $\mathbf{X}(t)$ be a D-dimensional functional variable. We assume that the variable is observed over a grid of equally spaced points $t_1, \ldots, t_J$ for all individuals. Then, any observed trajectory is denoted by the sequence $\mathbf{x}_j = \{\mathbf{x}(t_j)\} = (X_1(t_j), \ldots, X_D(t_j))$ for $j = 1, \ldots, J$. For simplicity of notation, we assume that all D functional variables are observed at the same time, but our method can apply to the case when the functional variables are observed at different times. One significant component of the RNN model is the hidden unit $\mathbf{h}(t)$ introduced to yield nonlinearity and abstraction in the network output. The hidden unit will be evaluated in parallel to $\mathbf{X}(t)$ at every observation point t_j. The role of $\mathbf{h}(t)$ is to create abstract dimensions to perform the nonlinear transformation on observations $\mathbf{x}(t)$ along t. It will also use a nonlinear function to propagate hidden information from $\mathbf{h}_{j-1} = \mathbf{h}(t_{j-1})$ to $\mathbf{h}_j = \mathbf{h}(t_j)$. We will use a general specification of an RNN to demonstrate how $\phi(\mathbf{X}(t))$ can be approximated.

A simple RNN model is defined as $\mathbf{h}_j = g_h(\mathbf{W}\mathbf{x}_j + \mathbf{U}\mathbf{h}_{j-1})$ where g_h is the activation. The hidden unit $\mathbf{h}$ is called the memory unit in the network. As illustrated in the above formulation, the hidden unit is responsible for handling the dependence between observations with a nonlinear autoregressive form. This also gives $\mathbf{h}_j$ the ability to memorize information up to t_j. In addition to tracking past information, the hidden unit uses abstract dimensions to process observations at each observation point. We can also observe that $\mathbf{h}$ is able to take multivariate observations. The dimension of $\mathbf{h}$, as the dimension of the feature space, will be treated as a tuning parameter. At the end, any observed trajectory can be summarized by $\mathbf{h}_J$. Applying a nonlinear mapping ϕ on any observed $\mathbf{x}(t)$ can be achieved by using $\mathbf{h}_J$ at an intermediate step. We generate the representation $\mathbf{z}$ as a function of $\mathbf{h}_J$ which can be approximated by a feed-forward neural network, that is, $\mathbf{z} = g_z(\mathbf{V}\mathbf{h}_J)$. Thus, the approximation of ϕ is based on the combination of two network models, from $\mathbf{x}$ to $\mathbf{h}_J$ and $\mathbf{h}_J$ to $\mathbf{z}$.

The first objective upon obtaining the representations is to reconstruct and denoise functional data. The proposed nonlinear representation model requires us to establish an inverse mapping ψ from the feature space to the original functional data space for completing the objective. Similar to the case of estimating ϕ, we propose to use RNNs for the approximation of ψ without defining an explicit form. The RNN model for approximating ψ is specified as $\tilde{\mathbf{h}}_j = g_{\tilde{h}}(\tilde{\mathbf{W}}\mathbf{z} + \tilde{\mathbf{U}}\tilde{\mathbf{h}}_{j-1})$ and $\tilde{\mathbf{x}}_j = g_x(\mathbf{G}\tilde{\mathbf{h}}_j)$ where $\tilde{\mathbf{x}}_j$ is the reconstruction at any given point t_j.

The second objective after obtaining the representations $\mathbf{z}$ is to predict the label $\mathbf{y}$, in one-hot format, for functional data. We establish a classification model based on the representation, that is, $\mathbf{y} = f(\mathbf{z})$. To align with the approximation of ϕ and ψ, we can use a feedforward neural network to estimate this nonlinear function, that is, $\mathbf{y} = g_y(\mathbf{M}\mathbf{z})$ where g_y is an activation function and $\mathbf{M}$ is the coefficient matrix. We will illustrate how to stack neural networks for approximating complex ϕ, ψ, and f in the supplementary document. In the remainder of this section, we provide the generalization ability of this model on predicting labels.

One concern of using neural networks for approximating nonlinear functions is the generalization ability. Given that the proposed model (1) is trained on one dataset, whether it can be generalized for any unseen data is of great interest. That is, we expect the representation model can provide reliable predictions in all three model objectives. This concern is partially caused by the over-parameterization in neural networks. For example, when we are inputting functional data into the model, the first RNN component introduces a hidden unit $\mathbf{h}_j$ in parallel to any observed trajectories. The dimension of $\mathbf{h}$ is usually larger than the dimension D in functional data. The correlated dimensions in $\mathbf{h}$ are used to extract as much information as possible. As we increase the dimension in the hidden unit, the dimensions of both coefficient matrices $\mathbf{W}$ and $\mathbf{U}$ are also increasing. The representation model and the classifier tends to over-fit the training data given the increasing model complexity as well as more stacked layers of networks.

To address such concerns, we need to quantify the generalization ability of the proposed representation model. We can use the generalization error which corresponds to the expected model errors over a data distribution. By providing a bound on the generalization error, we may have a better understanding of prediction performances with respect to the proposed method. In this work, we will prioritize the classification task over the reconstruction task and develop a generalization bound for it.

Let the classification loss function l_c be the margin loss, and the expected margin losses given a data distribution are given in the following definition.

Definition 1. `Expected margin loss`. Let $\mathcal{D}$ be the distribution of any data pair $(\boldsymbol{x}, \boldsymbol{y})$, and γ corresponds to the margin in classification. Then, the expected margin loss with respect to $\mathcal{D}$ is

$$l_\gamma(f_{\mathbf{w}}) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left[f_{\mathbf{w}}(\boldsymbol{x})[y] \le \gamma + \max_{j\neq y}\{f_{\mathbf{w}}(\boldsymbol{x})[j]\}\right], \quad (2)$$

where $f_{\mathbf{w}}(\boldsymbol{x})[y]$ is the prediction of class y in Q possible classes and $\mathbf{w} = (\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{U}}, \boldsymbol{M}, \boldsymbol{G})$ is the parameter vector. Let $f_{\mathbf{w}}(\mathbf{x}) : \mathcal{X} \to \mathbb{R}^Q$ be the classifier based on some covariates $\mathbf{x} \in \mathcal{X}$ and Q is the number of classes in $\mathbf{y}$. The curve label $\mathbf{y}$ is a Q-dimensional vector of binary values, such that the element in $\mathbf{y}$ is 1 for the observed label and zero otherwise. The output of $f_{\mathbf{w}}(\mathbf{x})$ is also a Q-dimensional vector. Each element in $f_{\mathbf{w}}(\mathbf{x})$ is the probability of $\mathbf{x}$ belonging to the corresponding class in $\mathbf{y}$. Given any training dataset $\mathcal{S}$, the empirical margin loss is calculated as

$$\hat{l}_\gamma(f_{\mathbf{w}}) = \frac{1}{|S|}\sum_{(x,y)\sim S}\left[f_{\mathbf{w}}(\boldsymbol{x})[y] \le \gamma + \max_{j\neq y}\{f_{\mathbf{w}}(\boldsymbol{x})[j]\}\right].$$

We can get the generalization bound on the classifier in model (1) as follows.

Theorem 1. (Generalization bound). For any $\delta, \gamma > 0$, with probability $\ge 1-\delta$ over a set of N data pairs, the following bound holds

$$l_0(f_{\mathbf{w}}) \le \hat{l}_\gamma(f_{\mathbf{w}}) + \mathcal{O}\left(\sqrt{\frac{E(\boldsymbol{w}) + \ln(N/\delta)}{N-1}}\right),$$

where $E(\boldsymbol{w}) = \frac{(H_{y,M}+H_{y,V}+H_{y,W}+H_{y,U})k\ln k}{\gamma^2}\left(\frac{||\boldsymbol{M}||^2}{||\boldsymbol{M}||_F^2} + \frac{||\boldsymbol{V}||^2}{||\boldsymbol{V}||_F^2} + \frac{||\boldsymbol{W}||^2}{||\boldsymbol{W}||_F^2} + \frac{||\boldsymbol{U}||^2}{||\boldsymbol{U}||_{\mathcal{F}}^2}\right)$.

For Theorem 1, we assume that the dimension of $\boldsymbol{h}$ and number of variables in $\boldsymbol{X}(t)$ are the same as k. The $||\cdot||$ is the Euclidean norm, and $||\cdot||_F$ is the Forbenius norm such that $||\mathbf{A}||_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^m A_{ij}^2}$ for a m by m matrix $\mathbf{A}$. The proof of the theorem is presented in the supplementary document.

The generalization bound presented in the article is to focus on the classification task of functional data. The generalization bound implies that the difference in the expected loss $l_0(f_{\mathbf{w}})$ and empirical loss $\hat{l}_\gamma(f_{\mathbf{w}})$ is bounded. The bound in this work implies that even though the model components are estimated by neural networks (potentially with many layers and parameters) with a finite set of data, the model can retain bounded generalization errors to any unseen data. Furthermore, we can see that the bound is determined by the norms of neural network parameter matrices rather than the number of parameters. It provides the rationale for using neural networks to estimate nonlinear mappings because we are very likely subject to overparameterization.

The next section will illustrate how to estimate the parameters of the networks. We also address an optimization obstacle, exploding/vanishing gradient, and provide solutions to the problem. Given that the observed trajectories may contain excessive noises, we introduce robustness into the estimation routine so that the representations are insensitive to local disturbances on the observations.

3. Model Estimation

The neural network parameters in $\boldsymbol{w} = (\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{M}, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{U}}, \boldsymbol{G})$ are estimated by minimizing the following objective function:

$$\begin{aligned}\hat{\boldsymbol{w}} &= \arg\min_{\mathbf{w}} \mathcal{L} = \arg\min_{\mathbf{w}} \mathcal{L}_c + \mathcal{L}_r, \\ &= \arg\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^N l_c(y_i, f(\boldsymbol{z}_i)) + \frac{1}{N}\sum_{i=1}^N l_r(x_{1:J}^i, \psi(\phi(x_{1:J}^i))),\end{aligned} \quad (3)$$

where l_c and l_r are the loss functions for assessing the classification and reconstruction results. The loss l_c can be the loss function of any classification method. The loss function l_r needs to be decomposed with respect to t_j, that is, $l_r(x_{1:J}^i, \psi(\phi(x_{1:J}^i))) = \frac{1}{J}\sum_{j=1}^J l_r^{(i)}(t_j)$. For the example of using a squared error loss function, $l_r^{(i)}(t_j) = ||\mathbf{x}_j^i - \tilde{\mathbf{x}}_j^i||^2$. Then, the optimization in (3) is done by updating parameters with $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot d\mathcal{L}/d\boldsymbol{w}$ for a learning rate α. For illustrating the estimation procedure, we will use the original observations. Adding a corruption step does not alter the derivation of parameter gradients. In case of missing observations, we can set the loss function to zero at their observed locations.

The gradient update is also termed backpropagation under the framework of neural networks. The gradients of parameters are calculated in the opposite direction of the network flow. The dashed lines in Figure 1 are showing the order of calculations. It is easy to backpropagate training errors in the standard feed-forward neural network which is estimating f. On the other hand, updating parameters in recurrent neural networks is more complicated. The training error is not only transferred through layers but time t_j's as well. Backpropagation through time (BPTT) is involved with the estimation of ϕ and ψ. Figure 1 illustrates both the network flow and direction of backpropagation. The detailed derivations of gradients are included in the supplementary document.

3.1. Vanishing/Exploding Gradient

The recurrent neural network and its variants are more prone to the vanishing or exploding gradient problem. That is, the gradients become too small or huge when applying updates on the current parameter estimates. After updating, the estimates are far away from the previous ones, and the optimization procedure becomes unstable. The issue is primarily caused by applying the chain rule over time, that is,

$$\frac{d\boldsymbol{h}_j}{d\boldsymbol{h}_s} = \prod_{k=s+1}^{j}\frac{d\boldsymbol{h}_k}{d\boldsymbol{h}_{k-1}} = \prod_{k=s+1}^{j}\boldsymbol{U}\mathrm{diag}(\sigma_h'(\boldsymbol{W}\boldsymbol{x}_k + \boldsymbol{U}\boldsymbol{h}_{k-1})).$$

The norm of $d\mathbf{h}_k/d\mathbf{h}_{k-1}$ is bounded by the norms of both terms in the product. As $j - s$ increases, $||d\mathbf{h}_k/d\mathbf{h}_{k-1}|| < 1$ leads to the vanishing gradients whereas $||d\mathbf{h}_k/d\mathbf{h}_{k-1}|| > 1$ leads to the

exploding case. This recursive term is unavoidable in recurrent networks and a major performance setback to estimations. The root of this problem is embedded in the long-term dependency between hidden units $\boldsymbol{h}_j$ and $\boldsymbol{h}_s$. From the specification in (1), $\boldsymbol{h}_j$ congregates all information from $\boldsymbol{h}_s$ for any $s < j$ regardless of the history significance. Given that we have a constant coefficient matrix $\mathbf{U}$ for weighting the previous hidden state, the product is $d\boldsymbol{h}_j/d\boldsymbol{h}_s$ is the result of multiplying the same coefficient over and over. This will lead to the gradient problem. One naive solution is to truncate the history dependence within a fixed threshold distance. If $j - s$ exceeds the threshold, we only calculate the recursive gradient for a fixed number of times. This approach fails to attend to important relations persisting through the distance in t. The rest of this section introduces remedies for the problem which do not truncate the history dependence.

The first approach to address the issue is the gradient clipping method. The clipping strategy is primarily focusing on the exploding gradient scenario (Pascanu et al. 2013) and can attend to the vanishing gradient problem with some modifications. The logic of this approach uses unstable gradients. Even though the gradient is either exploding or vanishing, it still points in the right direction of optimization. As a result, we need to control them from making the optimization procedure unstable. The clipping method is to impose a threshold on the norm of the gradient. If the norm of the gradient is larger than the threshold, then we scale down the gradient to a unit-norm vector.

Alternatively, it is more intuitive to enhance the hidden states with abilities to handle history dependence. In other words, the history dependence should be varying and be determined by the model. The main cause for gradient problems is from the constant coefficients $\mathbf{U}$ and $\tilde{\mathbf{U}}$ in (1). These describe a constant dependence between adjacent hidden states throughout the entire observation period. The motivation for this approach lies in replacing a constant coefficient matrix with a varying quantity. The solution is to introduce an additional hidden unit $\mathbf{m}$ as well as gating functions to process information from past hidden states. One popular model following this approach is the long short-term memory model (LSTM) as the following:

$$\begin{aligned}
\mathbf{i}_j &= g(\mathbf{W}_i\mathbf{x}_j + \mathbf{U}_i\mathbf{h}_{j-1}), \mathbf{o}_j = g(\mathbf{W}_o\mathbf{x}_j + \mathbf{U}_o\mathbf{h}_{j-1}), \mathbf{f}_j \\
&= g(\mathbf{W}_f\mathbf{x}_j + \mathbf{U}_f\mathbf{h}_{j-1}), \\
\mathbf{m}_j &= \mathbf{f}_j \circ \mathbf{m}_{j-1} + \mathbf{i}_j \circ \tilde{\mathbf{m}}_j \text{ where } \tilde{\mathbf{m}}_j = \tanh(\mathbf{W}_m\mathbf{x}_j + \mathbf{U}_m\mathbf{h}_{j-1}), \\
\mathbf{h}_j &= \mathbf{o}_j \circ \tanh(\mathbf{m}_j),
\end{aligned} \tag{4}$$

where g is the sigmoid function. LSTM defines three gates: the input gate $\mathbf{i}(t)$, the forget gate $\mathbf{f}(t)$ and the output gate $\mathbf{o}(t)$. These gating functions are bounded within $[0, 1]$ and act as a weighting function dependent on the observation point. They are responsible for handling external inputs and past information, updating the memory in $\boldsymbol{m}(t)$, and producing the current hidden state $\boldsymbol{h}(t)$. LSTM facilitates the propagation from $\boldsymbol{h}_{j-1}$ to $\boldsymbol{h}_j$ by $\boldsymbol{m}_j$ instead of direct connection between adjacent states. This replaces the recursive component from $d\boldsymbol{h}_j/d\boldsymbol{h}_{j-1}$ with $d\boldsymbol{m}_j/d\boldsymbol{m}_{j-1}$ in the calculation of gradients through time. The gating functions are present in the calculation of $\boldsymbol{m}_j$. Since the gating functions are varying for reflecting the temporal relationship, they can prevent the recursive gradient to use a constant value. As a result, LSTM and its variants can avoid the vanishing/exploding gradient problem. To illustrate the usefulness of the LSTM model, we can look at the derivative dm_j/dm_{j-1} which is derived in the supplementary document. In real applications, we will use the LSTM model for obtaining representations of functions.

3.2. Robustness

Robustness is crucial for establishing a representation model. It facilitates the proposed model in real applications where data is not always ideal. We would encounter several data problems preventing a fitted model to be useful in doing predictions. First, observed trajectories are often contaminated by random errors which do not have the same variance nor follow a normal distribution. At some locations, trajectories can be departed from the underlying function by outliers. We can also have random missing observations which forbid us from directly fitting any model. These data problems will become more severe when they are present across all dimensions. One approach for addressing these problems is to smooth functional data before constructing a prediction model. This would separate a single representation model into two and introduce additional estimation burdens. It is also difficult to verify the validity of smoothed functional data. As a result, we indulge the proposed model with robustness so that it can be insensitive to data corruption. This can help the obtained representations be less prone to uncontrollable errors and more reliable for further modeling tasks.

The alternative view of this issue is how to enforce the smoothness of the representation mapping ϕ. One important difference between curves and infinite-dimensional vectors is the smoothness assumption. That is, any random trajectory should behave in a way such that $\boldsymbol{x}(s) \approx \boldsymbol{x}(t)$ given $s \approx t$. If we can ensure the smoothness of reconstructed curves $\psi(\phi(\mathbf{x}(t)))$, then we are able to ensure that both ϕ and ψ are smooth. Traditional methods from FDA will impose some analytic roughness penalties to enforce the smoothness of curves. In our method, we do not express the mapping function as a linear combination of basis functions, the classic FDA treatment. Therefore, we cannot directly obtain the roughness penalty for the mapping functions. The roughness penalty is usually computed by integration. Numeric approximation to such integration depends on both the number of observation points and the completeness of any trajectory. Hence, adding an analytic roughness penalty puts too much burden on the estimation process.

Following the penalizing strategy, we propose to modify the estimation procedure for approximately enforcing the smoothness of mapping functions. This will also ensure the robustness of the proposed representation model. The modification is to introduce a corruption layer on functional data. We will show that this is equivalent to adding an approximate roughness penalty on the reconstructed trajectories. By penalizing the roughness of reconstructions, we can consequently control the smoothness of ϕ and ψ. As a result, the representations will become insensitive to random noises and missing observations. Training neural networks with added noises demonstrates the ability to enforce the smoothness of mapping functions for scalar variables (Bishop 1995). In this work, we will demonstrate that this modified estimation will provide robustness for the

model (1) and subsequently more reliable representations for further modeling tasks.

We will introduce a corruption layer to produce corrupted data before fitting the model. The representation model is then estimated with corrupted inputs while providing reconstruction and classification. The corruption process is a combination of discarding observations and adding random noises for imitating problems in real data. Let $p(x^\star_{1:J}|x_{1:J})$ represent a stochastic mapping for producing the corrupted input $x^\star_{1:J}$. The corruption steps $p(x^\star_{1:J}|x_{1:J})$ can be adding uncorrelated Gaussian errors to $x_{1:J}$, or it can randomly destroy individual elements in $x_{1:J}$ as if they are missing from the observed sequence. For illustration, we will first randomly discard observations and set them to be missing. Then, we will use $x^\star_j = x_j + \epsilon_j$ for obtaining corrupted input given that x_j is not missing. The Gaussian errors $\epsilon_j, j = 1, \ldots, J$, are iid and have a variance of σ^2. We will impose the same corruption step to each dimension if $\boldsymbol{x}(t)$ is not univariate. As a result, the representation model is instead obtaining features through $\phi(\boldsymbol{x}^\star_{1:J})$ based on the corrupted input and alternating the reconstruction loss as $\frac{1}{N}\sum_{i=1}^N l_r(\boldsymbol{x}^i, \psi(\phi(\boldsymbol{x}^{i,\star})))$. Using $\psi(\phi(\boldsymbol{x}^{i,\star}))$ in l_r will enrich the representations with robustness for better denoising capabilities as well as reconstructions based on partially observed inputs. This training strategy is equivalent to adding an approximate roughness penalty on the mapping functions. In the online supplementary document, we demonstrate the correspondence between the modified training process and the roughness penalization of mapping functions.

4. Applications

In this section, we will present the real applications of the FunNoL method to illustrate its performance on curve classification. We compare the FunNoL method with FPCA and the deep neural network proposed in Yao et al. (2021) on multiple datasets presented in Dau et al. (2019). The deep neural network proposed in Yao et al. (2021) is called AdaFNN in this article. AdaFNN uses neural networks for estimating a set of basis functions. These basis functions form a basis to project functional data. The linear projection is used as the representation. The comparison results on the curve reconstruction are presented in Section S7.3 of the supplementary document.

4.1. Applications on Univariate Functional Data

First, we will apply the FunNoL method to univariate functional data. Descriptions and illustrations of the datasets are included in Section S7.1 of the supplementary document. The applications are carried out in the following scheme. For each dataset, we split the data into training and test set. The FunNoL method and FPCA are first estimated based on the training set. We will use corrupted and original observations for fitting the FunNoL model (denoted by FunNoL_c and FunNoL_nc, respectively). Then, we use the estimated models, both FunNoL and FPCA, to predict representations of observations in the test set.

The main assessment for representations is focused on their efficacy in classification. We will use the prediction accuracy on labels from test data for demonstrating the classification performance. The scheme for obtaining the prediction accuracy is as follows. We first use the representations, obtained from the estimated models, to establish a logistic regression model for fitting the labels in the training set. After predicting representations for trajectories in the test set, the logistic model produces labels based on the predicted representations. For each dataset, we will split it into training and test set and repeat data splitting 50 times. For each splitting, we will repeat the estimation and prediction procedure to get the classification accuracy on the test set. The prediction performances from the 50 random splits are recorded for making comparisons.

Table 1. Prediction accuracy from FunNoL (with or without the corruption step, denoted by FunNoL_c and FunNoL_nc, respectively), FPCA and AdaFNN.

	`Starlight`	`Wafer`	`SwedishLeaf`	`Earthquakes`
# Training samples	8236	6164	625	322
FunNoL_c	0.975(0.003)	0.995(0.002)	0.398(0.052)	0.839(0.035)
FunNoL_nc	0.951(0.003)	0.995(0.002)	0.381(0.042)	0.814(0.027)
FPCA	0.847(0.006)	0.927(0.005)	0.471(0.057)	0.810(0.032)
AdaFNN	0.911(0.003)	0.931(0.005)	0.696(0.033)	0.812(0.033)

NOTE: The first row indicates the training sample size for each dataset. Each cell in the last four rows provides the average classification accuracy with standard errors in parentheses on curve labels in the test sets.

We apply the same scheme for both FunNoL and FPCA and compare the portion of corrected labels between these two approaches over all splits. The classification results on the test sets are summarized in Table 1. For making comparisons, we fix the number of features from FunNoL to be the same as the number of FPCs. Table 1 shows that adding a corruption step improves the prediction accuracy across datasets with different training sample sizes. The improvement is most significant when the sample size is large from the dataset `Starlight`. Thus, we can observe that manually corrupting data can prevent the FunNoL model from overfitting the training data and improve the generalization ability of representations. In comparison with FPCA, FunNoL is better in classification when the training sample size is large. The nonlinear mappings can be well approximated by neural networks with a large number of samples. When the training sample size is small, it is not clear which method produces better representations for classification. The AdaFNN method (Yao et al. 2021) uses neural networks to estimate a set of basis functions to project functional data. When linear projections are sufficient or appropriate such as for the `SwedishLeaf` dataset, AdaFNN provides a better basis to represent functional data compared to FPCA. For other cases, AdAFNN does not show a significant improvement from FPCA and has a lower prediction accuracy than the FunNoL method.

For further investigation of FunNoL, we can examine the latent feature space and compare it with the space spanned by the leading FPCs. This can also help us to understand how representations provide information for classification. Figure 2 presents the first two latent features from FunNoL_c and two leading FPC scores for the `Starlight` dataset. In this dataset, we have two classes (plotted in green and blue curves at the top left panel of Figure S1 in the supplementary document) that have similar trajectories. Their projections in the FPC space are not easily separable with a linear classifier. On the other hand, applying a nonlinear mapping produces representations which are showing better separation between these two classes.

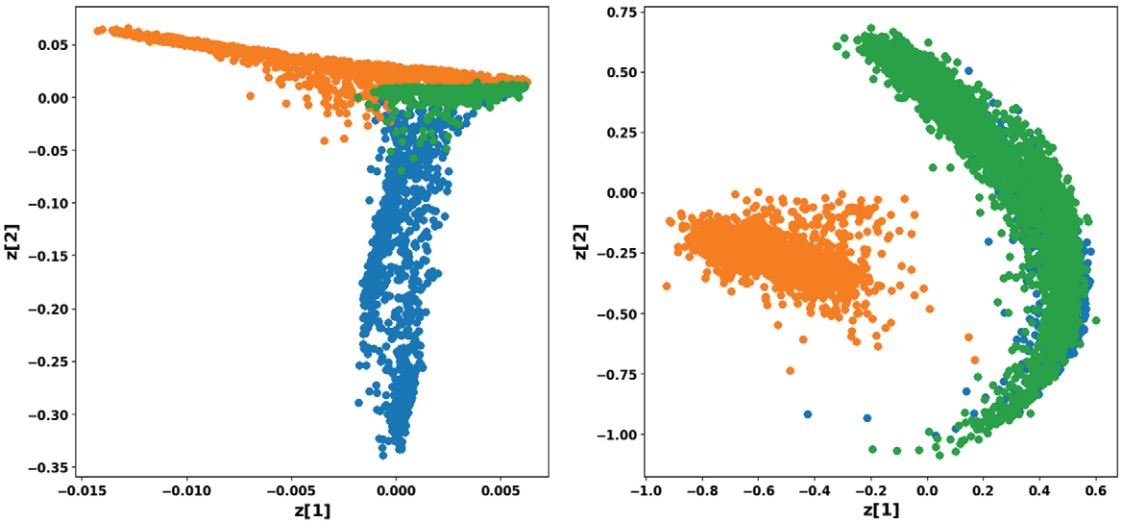

Figure 2. The first two dimensions from the feature space of FunNoL (left) and the two leading functional principal dimensions of FPCA (right) for the `Starlight` dataset. Colors correspond to the classes of individual trajectories.

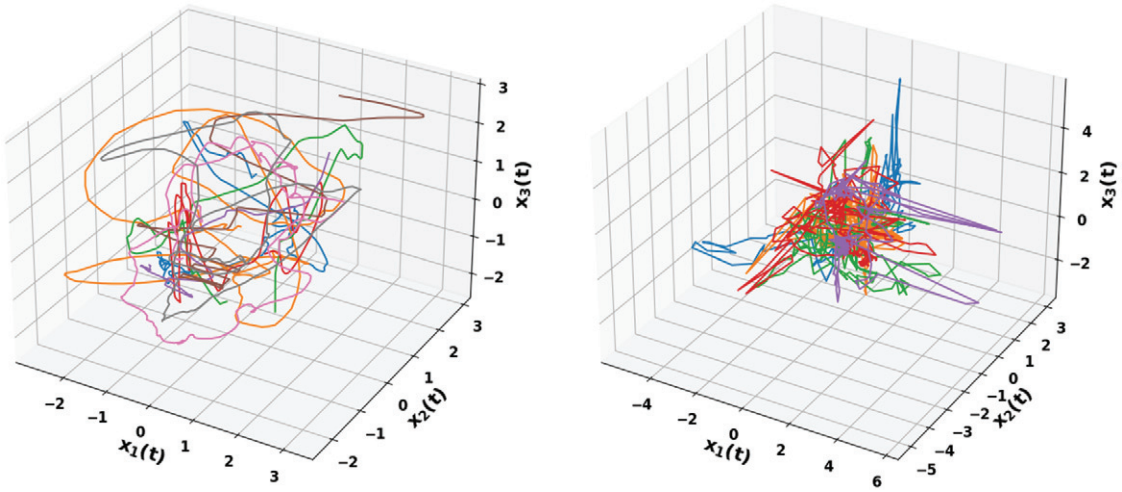

Figure 3. A few selected trajectories from the `Gesture` and `Cricket` dataset. In both graphs, each color indicates a different class of trajectory recorded in a three-dimensional space over time.

In conclusion, FunNoL is able to produce representations that are better suited for the classification of univariate functional data in comparison with FPC scores.

4.2. Applications on Multivariate Functional Data

One motivation of this work is to focus on multivariate functional data that are more common in applications. To show the advantages of FunNoL in learning multivariate functional data, we have applied it to two datasets `Gesture` and `Cricket` from Dau et al. (2019). Figure 3 depicts sample trajectories from both the `Gesture` and `Cricket` dataset.

One advantage of FunNoL is that it can be directly applied to multivariate functional data. This makes it more suitable for addressing the nonlinear structures of multivariate functions without concatenating them into univariate ones. We will repeat the splitting routine as done to the previous univariate functional data. For each split, the prediction accuracy on curve labels is recorded and summarized in Table 2.

Table 2. Prediction accuracy from FunNoL (with or without the corruption step, denoted by FunNoL_c and FunNoL_nc, respectively), FPCA, and AdaFNN, on multivariate functional data.

	`Gesture`	`Cricket`
# Training samples	3582	390
FunNoL_c	0.963(0.0044)	0.316(0.031)
FunNoL_nc	0.920(0.0055)	0.368(0.033)
FPCA	0.880(0.0080)	0.0791(0.023)
mFPCA	0.887(0.0076)	0.0842(0.021)
AdaFNN	0.890(0.011)	0.0949(0.026)

NOTE: mFPCA is to obtain FPC scores by the multivariate FPCA method (Happ and Greven 2018). The first row indicates the training sample size for each dataset. Each cell in the last five rows provides classification accuracy with standard errors in parentheses on curve labels in the test sets.

We observe a more significant improvement in the prediction accuracy than the case of univariate functional data. The nonlinear representations provide a better ground to build classification models for multivariate functional data. FunNoL outperforms FPCA even with a small sample size as in the `Cricket` dataset. Adding a corruption step does not necessarily lead to

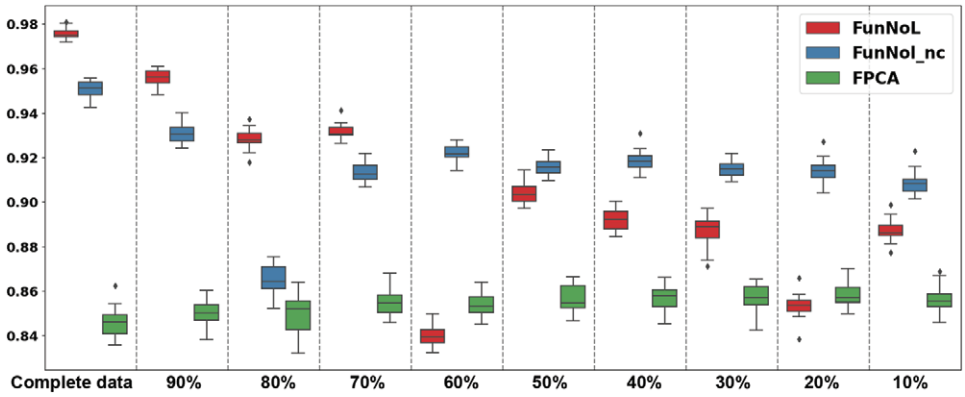

Figure 4. The simulations on `Starlight` dataset. The y-axis represents the classification accuracy of test data whereas the x-axis corresponds to portions of available observations from individual functional data. The simulation is carried out 50 times for each sparsity setting. The distributions of prediction accuracy are represented by boxplots.

an increase in prediction accuracy. From Table 2, the influence of a corruption step depends on the training size. Datasets with more samples (as in the `Gesture` dataset) can benefit from corrupting data for training representation models. The AdaFNN does not directly address multivariate functional data, hence, we observe that it does not lead to a significant improvement in classifying multivariate functional data over FPCA methods. We will examine the classification of multivariate functional data in the next section through simulation studies.

5. Simulations

This section includes simulation studies for assessing the FunNoL method. We examine whether data corruption impairs the performance of FunNoL on the aforementioned modeling tasks. These studies can also be used for verifying the robustness of the proposed method. In real applications, functional data are not always ideal where trajectories may not be fully observed. That is, each observed trajectory may have missing observations. The observations may also contain excessive noises disturbing the resulting representations.

As stated in the previous section, FunNoL uses a corruption step in estimation to reduce influences from corruption in functional data. In the simulation studies, we are aiming to show whether representations from FunNoL are reliable in modeling tasks given different levels of corruption in data. For that matter, we investigate whether the portion of missing observations in trajectories undermines the reliability of the learning model.

The simulation studies are based on real datasets. We use the model estimated by the complete data as the benchmark. We then use a portion of the complete dataset for estimation while setting the remaining portion to be missing. For instance, the setting 90% shown in Figure 4 means that only 90% of observations are available for estimations. Assuming that any individual functional data has J number of observations, the simulation would use $90\% \times J$ number of observations from each trajectory. The remaining 10% of observations are randomly chosen and set to be missing. The performance of various methods is compared based on the downsampled data. The above simulation procedure is replicated 50 times.

5.1. Simulation on Univariate Functional Data

We conduct simulations on univariate functional datasets. The portion of correctly predicted labels given each setting is summarized in Figure 4. Figure 4 shows that the classification performance based on FunNoL representations does not suffer from the increasing portion of missing observations. Using the example of the `Starglight` dataset, when the curves have more than 60% data observed, adding a corruption step prevents overfitting the model and ensures that the representations have better generalization abilities and hence are more useful in building prediction models. In this scenario, the FunNoL method with the corruption step has a larger prediction accuracy. On the other hand, adding manual corruption to training data has a limit on improving the learned representations. When the curves have less than 60% data observed, additional corruptions will overwhelm the estimation process, and the FunNoL method with no corruption step (`FunNOl_nc`) has a higher prediction accuracy. Therefore, we recommend using the FunNoL method with no corruption step in this scenario.

5.2. Simulation on Multivariate Functional Data

We will present simulation studies on multivariate functional data. The following simulations focused on the classification performance of the FunNoL method.

FunNoL is designed to address multivariate functional data, and we expect to see more significant improvement over FPCA as demonstrated in Figure 5. FunNoL exceeds in predicting labels for both datasets. In addition, FunNoL performs equally well given the increasing portion of missing observations. We can conclude that the proposed nonlinear mapping is more suitable for multivariate data where nonlinear structures become overwhelming for linear models. More simulation results are included in Section S8 of the supplementary document.

6. Conclusion

In this work, we propose a nonlinear learning method, FunNoL, for functional data. Its main purpose is to address the nonlinear structures in functional data and produce representa-

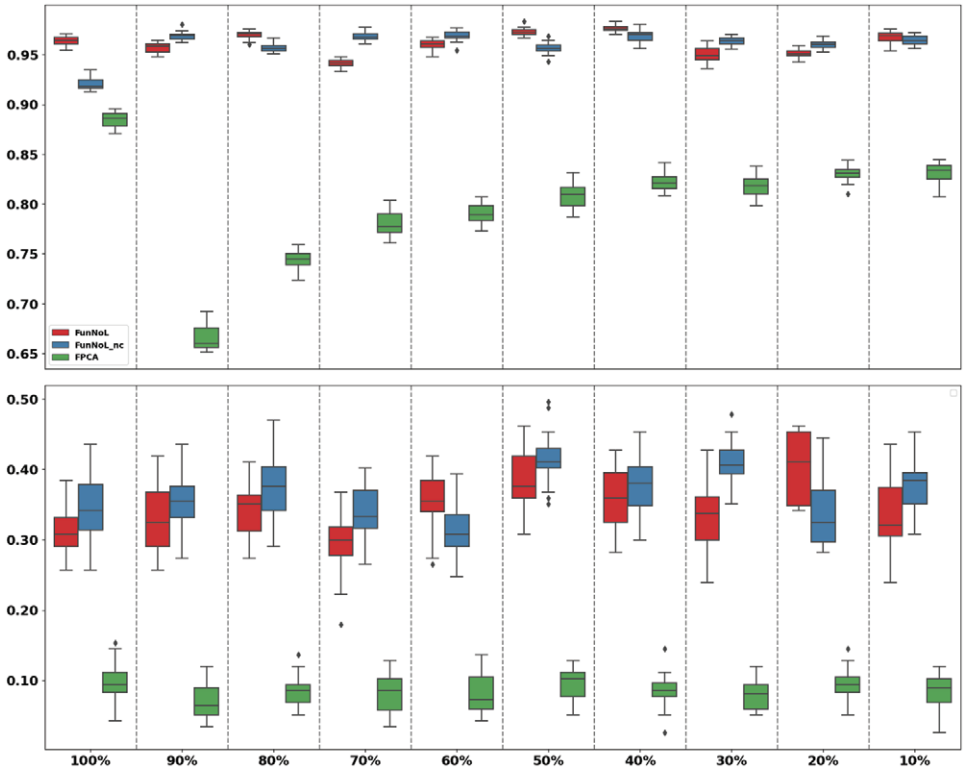

Figure 5. The top panel corresponds to the `gesture` data, and the bottom panel presents the classification of `cricket` data. Red boxes are the FunNoL model trained with corrupted observations, whereas blue boxes are the model trained with original observations. Green boxes represent the prediction accuracy by using FPC scores. Dashed lines separate different data settings where we sample a portion of training data to estimate the model. The leftmost panel corresponds to the case where all trajectories are used. The rightmost panel is the case where 10% of data are sampled for doing estimation.

tions for future modeling tasks. It introduces flexible nonlinear mappings that provide better representations of functional data when linear projections are not sufficient. Also, the method can be directly applied to multivariate functional data. It achieves dimension reduction when the feature space of the mapping is a low-dimensional vector space. As a result, these vector representations reduce the efforts to build prediction models instead of directly using trajectories. The article considers two modeling tasks, curve reconstruction and classification, upon obtaining representations. We have applied FunNoL to several real datasets to examine its performances. In addition, we arranged different data corruption scenarios in simulation studies. The study results show that the FunNoL method is able to handle functional data with missing observations. The nonlinear mapping is also designed to be insensitive to any local disturbances in trajectories, hence, the representations only retain the necessary information for subsequent prediction models. Conclusively, we have demonstrated the superiority of FunNoL as a nonlinear learning method for functional data. In this work, we focus on functional data defined in the time domain, which is well modeled by recurrent neural networks. Future extensions to this work can be made on functional data defined on a multi-dimensional domain. For that matter, we can consider other neural networks to model multi-dimensional functional data.

Supplementary Materials

A supplementary document includes the additional application and simulation results, proofs of the generalization bound, and some technical details.

Acknowledgments

The authors would like to thank the editor, the associate editor, and one referee for many insightful comments. These comments are very helpful for us to improve our work.

Disclosure Statement

The authors report there are no competing interests to declare.

Funding

This research is supported by the Discovery grant (RGPIN-2023-04057) to J.Cao from the Natural Sciences and Engineering Research Council of Canada (NSERC).

ORCID

Jiguo Cao https://orcid.org/0000-0001-7417-6330

References

Bishop, C. M. (1995), "Training with Noise is Equivalent to Tikhonov Regularization," *Neural Computation*, 7, 108–116. [186]

Chen, K., and Lei, J. (2015), "Localized Functional Principal Component Analysis," *Journal of the American Statistical Association*, 110, 1266–1275. [181]

Chiou, J., Chen, Y., and Yang, Y. (2014), "Multivariate Functional Principal Component Analysis: A Normalization Approach," *Statistica Sinica*, 24, 1571–1597. [182]

Dai, X., and Müller, H. (2018), "Principal Component Analysis for Functional Data on Riemannian Manifolds and Spheres," *The Annals of Statistics*, 46, 3334–3361. [181]

Dai, X., Muller, H.-G., and Yao, F. (2017), "Optimal Bayes Classifiers for Functional Data and Density Ratios," *Biometrika*, 104, 545–560. [182]

Dau, H., Bagnall, A., Kamgar, K., Yeh, C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. (2019), "The UCR Time Series Archive," *IEEE/CAA Journal of Automatica Sinica*, 6, 1293–1305. [183,187,188]

Dauxois, J., Pousse, A., and Romain, Y. (1982), "Asymptotic Theory for the Principal Component Analysis of a Vector Random Function: Some Applications to Statistical Inference," *Journal of Multivariate Analysis*, 12, 136–154. [181]

Delaigle, A., and Hall, P. (2012), "Achieving Near Perfect Classification for Functional Data," *Journal of the Royal Statistical Society*, Series B, 74, 267–286. [182]

Hall, P., and Hosseini-Nasab, M. (2005), "On Properties of Functional Principal Components Analysis," *Journal of the Royal Statistical Society*, Series B, 68, 109–126. [181]

Hall, P., Müller, H.-G., and Wang, J.-L. (2006), "Properties of Principal Component Methods for Functional and Longitudinal Data Analysis," *The Annals of Statistics*, 34, 1493–1517. [181]

Happ, C., and Greven, S. (2018), "Multivariate Functional Principal Component Analysis for Data Observed on Different (Dimensional) Domains," *Journal of the American Statistical Association*, 13, 649–659. [182,188]

James, G., Hastie, T., and Sugar, C. (2000), "Principal Component Models for Sparse Functional Data," *Biometrika*, 87, 587–602. [182]

Jiang, S., Cao, J., Colditz, G. A., and Rosner, B. (2023a), "Predicting the Onset of Breast Cancer Using Mammogram Imaging Data with Irregular Boundary," *Biostatistics*, 24, 358–371. [182]

——— (2023b), "Supervised Two-Dimensional Functional Principal Component Analysis with Time-to-Event Outcome on Mammogram Imaging Data," *Biometrics*, 79, 1359–1369. [182]

Kokoszak, P., and Reimherr, M. (2017), *Introduction to Functional Data Analysis*, London: Chapman and Hall/CRC. [181]

Kong, D., Bondell, H., and Shen, W. (2018), "Outlier Detection and Robust Estimation in Nonparametric Regression," in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics* (Vol. 84), pp. 208–216. [182]

Lin, Z., Wang, L., and Cao, J. (2016), "Interpretable Functional Principal Component Analysis," *Biometrics*, 72, 846–854. [181]

Müller, H.-G. (2005), "Functional Modelling and Classification of Longitudinal Data," *Scandinavian Journal of Statistics* 32, 223–240. [181]

Müller, H.-G., and Stadtmüller, U. (2005), "Generalized Functional Linear Models," *The Annals of Statistics*, 33, 774–805. [181]

Nie, Y., and Cao, J. (2020), "Sparse Functional Principal Component Analysis in a New Regression Framework," *Computational Statistics & Data Analysis*, 152, 107016. [181]

Nie, Y., Wang, L., Liu, B., and Cao, J. (2018), "Supervised Functional Principal Component Analysis," *Statistics and Computing*, 28, 713–723. [182]

Nie, Y., Yang, Y., Wang, L., and Cao, J. (2022), "Recovering the Underlying Trajectory from Sparse and Irregular Longitudinal Data," *Canadian Journal of Statistics*, 50, 122–141. [182]

Pascanu, R., Mikolov, T., and Bengio, Y. (2013), "On the Difficulty of Training Recurrent Neural Networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning* (Vol. 28), pp. 1310–1318. [186]

Peng, J., and Müller, H.-G. (2008), "Distance-based Clustering of Sparsely Observed Stochastic Processes, with Applications to Online Auctions," *The Annals of Applied Statistics*, 2, 1056–1077. [181]

Ramsay, J., and Silverman, B. (2005), *Functional Data Analysis*, New York: Springer. [181,182]

Shi, H., and Cao, J. (2022), "Robust Functional Principal Component Analysis based on a New Regression Framework," *Journal of Agricultural, Biological and Environmental Statistics*, 27, 523–543. [182]

Shi, H., Dong, J., Wang, L., and Cao, J. (2021), "Functional Principal Component Analysis for Longitudinal Data with Informative Dropout," *Statistics in Medicine*, 40, 712–724. [182]

Shi, H., Yang, Y., Wang, L., Ma, D., Beg, M. F., Pei, J., and Cao, J. (2022), "Two-Dimensional Functional Orthogonal Approximation Method for Image Feature Extraction," *Journal of Computational and Graphical Statistics*, 31, 1127–1140. [182]

Vieu, P., and Ferraty, F. (2006), *Nonparametric Functional Data Analysis: Theory and Practice*, New York: Springer. [181]

Yao, F., Müller, H.-G., and Wang, J.-L. (2005a), "Functional Data Analysis for Sparse Longitudinal Data," *Journal of the American Statistical Association*, 100, 577–590. [182]

——— (2005b), "Functional Linear Regression Analysis for Longitudinal Data," *The Annals of Statistics*, 33, 2873–2903. [181]

Yao, J., Mueller, J., and Wang, J.-L. (2021), "Deep Learning for Functional Data Analysis with Adaptive Basis Layers," *Proceedings of Machine Learning Research*, 139, 11898–11908. [187]

Zhou, Z., and Sang, P. (2021), "Continuum Centroid Classifier for Functional Data," *Canadian Journal of Statistics*, 50, 200–220. DOI:10.1002/cjs.11624. [182]